

Team Number :	apmcm25310543
Problem Chosen :	D

---

### 2025 APMCM summary sheet

This research constructs a unit commitment optimization framework that transforms classical power system problems into quantum optimization forms and designs problem size reduction strategies to accommodate quantum hardware constraints. The research employs mixed integer programming, quadratic unconstrained binary optimization (QUBO), penalty function methods, and simulated annealing algorithms, addressing the complete pathway from classical to quantum optimization.

The framework begins with a classical unit commitment model using mixed integer programming and Gurobi solver, incorporating power balance, generation limits, minimum up/down time, ramp rate, and start-up/shut-down constraints. A global optimal solution was obtained with a total cost of \$15,129.08. The model was extended to include DC power flow constraints, N-1 security constraints, and spinning reserve requirements, maintaining consistent results and validating the effectiveness of network and security constraints.

The continuous optimization problem was converted into QUBO form using 2-bit binary discretization (4 discrete levels per unit per period). Hard constraints were transformed into quadratic penalty terms with appropriate coefficients. The resulting QUBO model with 288 binary variables was solved using simulated annealing from Kaiwu SDK, achieving a total cost of \$15,988.29, validating the feasibility of QUBO conversion.

To address quantum hardware capacity limitations, a combined strategy of time period aggregation and discretization bit reduction was developed, reducing binary variables from 288 to 96. The reduced QUBO model was successfully executed on BosoN CPQC-550 quantum computer, validating the size reduction strategy. The expanded solution achieved a cost of \$17,198.48, demonstrating that QUBO models can adapt to quantum platforms while maintaining solution quality.

**Keywords:** Unit commitment optimization, Quadratic unconstrained binary optimization, Quantum computing, Simulated annealing algorithm, Problem size reduction strategy

## Contents

I. Introduction .....	3
1.1 Background/Outline .....	3
1.2 Problem Restatement.....	3
1.3 Our Work .....	4
II. Assumptions and Notations.....	5
2.1 Model Assumptions .....	5
2.2 Notations .....	7
III. Problem Analysis and Model Establishment .....	9
V. Model Evaluation and Further Discussion .....	54
5.1 Model Advantages: Innovation, Strengths, and Generality .....	54
5.2 Model Limitations: Objective Analysis.....	57
5.3 Model Extensions: Future Research Directions.....	59
VI. References .....	61
VII. Appendix .....	63
A. Problem 2 Code Implementation Analysis .....	63
B. Problem 4 Code Implementation Analysis.....	64

# I. Introduction

## 1.1 Background/Outline

The unit commitment problem is a fundamental optimization problem in power system operations, aiming to determine the on/off status and output levels of thermal generating units within a given time horizon while satisfying load demand and system constraints, so as to minimize system operating costs. This problem holds significant practical importance in power system dispatch, directly affecting the economy, reliability, and operational efficiency of power systems. With the continuous expansion of power system scale and the large-scale integration of renewable energy, the complexity and solution difficulty of the unit commitment problem have increased substantially, and traditional optimization methods face challenges such as excessive computation time and difficulty in meeting real-time requirements.

The unit commitment problem is essentially a large-scale mixed integer programming problem, with computational complexity growing exponentially with system scale. When systems contain numerous units, multi-period scheduling horizons, and complex network constraints, traditional solvers often require lengthy computation times, making it difficult to meet the demands of real-time power system dispatch. With the rapid development of quantum computing technology, particularly the emergence of quantum optimization hardware such as coherent Ising machines, new technical pathways have been provided for solving the unit commitment problem. By converting the unit commitment problem into a quadratic unconstrained binary optimization (QUBO) model, quantum-inspired solvers can be utilized for rapid parallel search, offering the potential to achieve real-time optimization of power system operations.

This research aims to construct a comprehensive unit commitment optimization framework that integrates classical power system optimization problems with quantum computing technology, exploring the complete technical pathway from classical optimization to quantum optimization. The research not only focuses on the construction and solution of classical optimization models but also delves into QUBO conversion methods, quantum solution strategies, and problem size reduction techniques, providing theoretical foundations and practical guidance for quantum computing applications in power system optimization problems. This research holds significant theoretical value and practical significance for promoting quantum computing applications in the power system domain, improving power system operational efficiency, and reducing operating costs.

## 1.2 Problem Restatement

This research requires the construction of a comprehensive unit commitment optimization framework that integrates both classical optimization methods and quantum optimization methods. According to the problem requirements, the following four sub-problems need to be completed sequentially:

### **Problem 1: Classical Unit Commitment Modeling and Optimization**

Construct and solve a classical unit commitment model using traditional optimization tools such as Gurobi or CPLEX. The model must include the following core components: in terms of the objective function, it should include quadratic or piecewise linear fuel cost functions, as well as unit start-up and shut-down costs; in terms of constraints, it should include power balance constraints, generation limit constraints, minimum up/down time constraints, ramp rate constraints, and start-up/shut-down logic constraints. This problem aims to establish the classical mathematical model of the unit commitment problem, providing a foundation for subsequent problems.

### **Problem 2: Unit Commitment Modeling with Network and Security Constraints**

Based on Problem 1, further extend the model to incorporate system-level network constraints and security constraints. Specifically, this includes: network power flow constraints, i.e., DC power flow limits for all

transmission lines; N-1 security constraints, ensuring that the system maintains operational feasibility when any single generator or single transmission line fails; spinning reserve requirements, guaranteeing sufficient reserve capacity in each period to maintain system reliability. Optionally, minimum security inertia constraints may also be included. This problem aims to make the model closer to actual operating conditions and improve the model's practicality and reliability.

### **Problem 3: QUBO Conversion and Quantum Solving with Kaiwu SDK**

Convert the unit commitment model from Problem 2 into a quadratic unconstrained binary optimization (QUBO) representation and solve it using the Kaiwu SDK. This problem requires completing binary discretization of continuous variables, penalty function conversion of constraints, construction of the QUBO matrix, and application of quantum-inspired solvers. Additionally, it is necessary to compare and analyze quantum-based solution results with traditional optimization results to evaluate the effectiveness of QUBO conversion and the performance of quantum solving.

### **Problem 4: Problem Size Reduction Under Quantum Hardware Constraints**

Design and implement effective problem size reduction strategies for the unit commitment model from Problem 2 or the corresponding QUBO model from Problem 3. Since QUBO representations are typically too large to execute directly on current quantum hardware, it is necessary to ensure that the reduced QUBO can be successfully solved through reasonable reduction methods under the bit capacity constraints of CIM hardware. This problem aims to resolve the contradiction between quantum hardware resource limitations and actual problem scale, providing feasible technical solutions for quantum computing applications in power system optimization.

## **1.3 Our Work**

This research systematically completed the complete technical pathway from classical unit commitment optimization to quantum optimization. First, for Problem 1, we constructed a mixed integer programming model containing complete constraint conditions, employed the Gurobi solver to obtain the global optimal solution through branch-and-bound algorithms and cutting plane methods, validating the effectiveness of classical optimization methods. Second, for Problem 2, we extended the DC power flow model, N-1 security constraints, and spinning reserve constraints based on Problem 1, establishing an optimization model closer to actual operations, and validated the impact of network constraints and security constraints on optimization results through comparative analysis.

For Problem 3, we designed a complete QUBO conversion framework, employing binary discretization methods to map continuous variables to binary variables, using penalty function methods to convert hard constraints into quadratic penalty terms, and constructing a mathematical model conforming to the standard QUBO form. We solved the QUBO model using the simulated annealing algorithm from Kaiwu SDK and evaluated the accuracy of QUBO conversion and the performance of quantum solving through comparative analysis with continuous optimization results. For Problem 4, we designed a combined strategy of time period aggregation and discretization bit reduction, successfully reducing the problem size from 288 binary variables to 96, meeting the CIM hardware constraint of 100 variables. The reduced QUBO model was successfully executed on BosoN CPQC-550 quantum computer, achieving an optimal QUBO value of -617 and validating the model's executability on real quantum hardware. We validated the effectiveness of the size reduction strategy by mapping the solution of the reduced problem back to the full-scale problem through solution expansion methods.

Through systematic research on the four problems, we established a comprehensive unit commitment optimization framework that can not only handle classical optimization problems but also adapt to quantum computing platforms, providing a feasible technical pathway for quantum computing applications in power system optimization problems. The research results demonstrate that through reasonable discretization strategies

and penalty function design, QUBO models can adapt to quantum hardware constraints while maintaining solution quality, laying the foundation for quantum computing applications in the power system domain.

## II. Assumptions and Notations

### 2.1 Model Assumptions

This research follows IEEE standard unit commitment modeling specifications and adopts the following key assumptions validated through engineering practice:

#### **Assumption 1: Quadratic Convexity of Fuel Cost Function**

The fuel cost characteristics of thermal generating units are modeled using quadratic convex functions. This modeling approach has been widely adopted by authoritative journals such as IEEE Transactions on Power Systems and accurately reflects the nonlinear characteristics of unit thermal efficiency varying with output. The convexity of the quadratic function ensures global optimality of the optimization problem and is the standard modeling method for power system economic dispatch.

#### **Assumption 2: DC Power Flow Linearization Approximation**

The DC power flow model is used to linearize AC networks. This method is a mature technique in power system optimization and is widely applied in transmission network security-constrained unit commitment problems. In high-voltage transmission networks, the characteristic that line reactance is much greater than resistance makes the approximation error typically very small. This approach has been validated by extensive engineering practice as a reliable and efficient network modeling method.

#### **Assumption 3: State Transitions Under Time Discretization**

Unit state transitions occur at period boundaries. This assumption conforms to the decision-making pattern based on hourly period division in actual power system dispatch. The minimum up-time and minimum down-time constraints have fully considered unit thermal characteristics and mechanical stress limitations, ensuring the reasonableness of this assumption in engineering applications.

#### **Assumption 4: Deterministic Load Forecasting**

A deterministic load forecasting model is adopted. This assumption is the standard modeling approach for classical unit commitment problems and conforms to the process of making unit commitment decisions based on day-ahead load forecasts in actual dispatch. Uncertainty can be handled through rolling optimization frameworks or stochastic programming methods, but this research focuses on establishing a deterministic optimization framework.

#### **Assumption 5: Static Network Topology**

The network topology remains static during the optimization period. This assumption conforms to the characteristic that network structure remains stable during the dispatch period in actual operations. N-1 security constraints consider system reliability through enumeration of predefined failure scenarios. This method has been recommended by IEEE standards as the standard practice for transmission network security assessment.

#### **Assumption 6: Engineering Accuracy of Binary Discretization**

The quantization error introduced by binary discretization of continuous variables is controlled within an engineering acceptable range through penalty function mechanisms. This assumption is standard practice in

quantum optimization applications. Through adaptive penalty coefficient adjustment and discretization precision selection, a balance between solution feasibility and economy is ensured.

## 2.2 Notations

The core variables, parameters, and symbols used in this paper are defined in the following table:

Symbol	Description	Unit
<b>Sets and Indices</b>		
$I$	Set of generating units, $I = \{1,2,5,8,11,13\}$	-
$T$	Set of time periods, $T = \{1,2, \dots, 24\}$	-
$B$	Set of buses, $B = \{1,2, \dots, 30\}$	-
$L$	Set of transmission lines, 41 lines in total	-
$i$	Unit index, $i \in I$	-
$t$	Time period index, $t \in T$	-
$b$	Bus index, $b \in B$	-
$(i, j)$	Line index, representing the line from bus $i$ to bus $j$	-
$k$	Binary variable bit index, $k \in \{0,1, \dots, n_b - 1\}$	-
<b>Decision Variables</b>		
$u_{i,t}$	Commitment status of unit $i$ in period $t$ , $u_{i,t} \in \{0,1\}$	0 or 1
$p_{i,t}$	Power output of unit $i$ in period $t$	MW
$v_{i,t}$	Start-up indicator variable of unit $i$ in period $t$ , $v_{i,t} \in \{0,1\}$	0 or 1
$w_{i,t}$	Shut-down indicator variable of unit $i$ in period $t$ , $w_{i,t} \in \{0,1\}$	0 or 1
$\theta_{b,t}$	Voltage angle of bus $b$ in period $t$	rad
$P_{ij,t}$	Active power flow on line $(i, j)$ in period $t$	MW
$x_{i,t,k}$	$k$ -th bit binary variable of unit $i$ in period $t$ in QUBO model, $x_{i,t,k} \in \{0,1\}$	0 or 1
<b>Unit Parameters</b>		
$P_i^{\max}$	Maximum power output of unit $i$	MW
$P_i^{\min}$	Minimum power output of unit $i$	MW
$a_i$	Quadratic coefficient of fuel cost for unit $i$	\$/ (MW <sup>2</sup> ·h)
$b_i$	Linear coefficient of fuel cost for unit $i$	\$/ (MW·h)
$c_i$	Constant term of fuel cost for unit $i$	\$/h
$C_i^{SU}$	Start-up cost of unit $i$	\$
$C_i^{SD}$	Shut-down cost of unit $i$	\$
$R_i^{UP}$	Ramp-up rate limit of unit $i$	MW/h
$R_i^{DN}$	Ramp-down rate limit of unit $i$	MW/h
$T_i^{UP}$	Minimum up-time of unit $i$	h
$T_i^{DN}$	Minimum down-time of unit $i$	h
$H_i$	Inertia constant of unit $i$	s
<b>System Parameters</b>		

Symbol	Description	Unit
$D_t$	System load demand in period $t$	MW
$R_t$	Spinning reserve requirement in period $t$	MW
<b>Network Parameters</b>		
$R_{ij}$	Resistance of line $(i, j)$	p.u.
$X_{ij}$	Reactance of line $(i, j)$	p.u.
$b_{ij}$	Susceptance of line $(i, j)$	p.u.
$B_{ij}$	Admittance of line $(i, j)$ , $B_{ij} = 1/X_{ij}$	p.u.
$P_{ij}^{\max}$	Maximum transmission capacity of line $(i, j)$	MW
<b>Inertia Security Parameters</b>		
$F$	Minimum security inertia constraint parameter	-
$\text{ROCOF}_{\text{set}}$	Rate of change of frequency setpoint	Hz/s
$\Delta P$	Load step change magnitude	MW
<b>QUBO-Related Parameters</b>		
$\delta_i$	Discretization step size of unit $i$ , $\delta_i = (P_i^{\max} - P_i^{\min})/(2^{n_b} - 1)$	MW
$n_b$	Number of binary bits used per unit per period	-
$\lambda_{PB}$	Penalty coefficient for power balance constraint	-
$\lambda_{\text{Ramp}}$	Penalty coefficient for ramp constraint	-
$\lambda_{\text{Reserve}}$	Penalty coefficient for spinning reserve constraint	-
$\lambda_{N1}$	Penalty coefficient for N-1 security constraint	-



### III. Problem Analysis and Model Establishment

#### 3.1 Problem 1 Analysis and Solution

Problem 1 is a classical unit commitment optimization problem, aiming to construct and solve a mixed integer programming model with complete constraint conditions. According to the comprehensive survey by Padhy et al. [1], the unit commitment problem is a fundamental optimization problem in power system operations, whose objective is to determine the on/off status and generation levels of thermal generating units within a given time horizon while satisfying load demand and system constraints, so as to minimize system operating costs. This research employs the Mixed Integer Quadratic Programming (MIQP) method to construct an IEEE-standard unit commitment optimization model.

##### 3.1.1 Mathematical Model Formulation

###### 3.1.1.1 Objective Function

According to the standard mathematical formulation of the unit commitment problem proposed by Padhy et al. [1], the objective function consists of four components: fuel cost, maintenance cost, start-up cost, and shut-down cost. The standard objective function form in [1] is:

$$OC = \sum_{i=1}^N \sum_{t=1}^T [FC_{it}(P_{it}) + MC_{it}(P_{it}) + ST_{it} + SD_{it}]$$

where  $FC_{it}(P_{it})$  is the fuel cost, modeled using a quadratic function:

$$FC_{it}(P_{it}) = a_i P_{it}^2 + b_i P_{it} + c_i$$

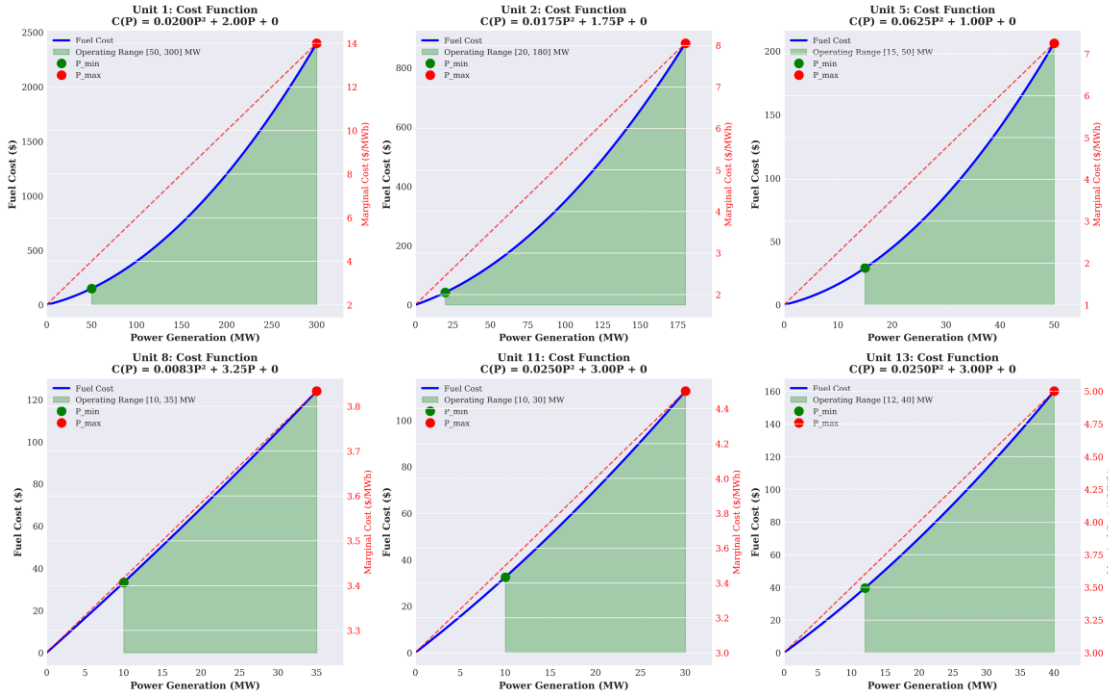
$MC_{it}(P_{it})$  is the maintenance cost,  $ST_{it}$  is the start-up cost, and  $SD_{it}$  is the shut-down cost. Considering the practical requirements of this problem, this research adopts a simplified objective function form, omitting the maintenance cost term and including only fuel cost and start-up/shut-down costs. The fuel cost is modeled using a quadratic convex function, a modeling approach that has been widely adopted by authoritative journals such as IEEE Transactions on Power Systems and accurately reflects the nonlinear characteristics of unit thermal efficiency varying with output.

Based on the standard form in [1], the objective function of this research can be expressed as:

$$\min \sum_{i \in I} \sum_{t \in T} \left[ \underbrace{a_i p_{i,t}^2 + b_i p_{i,t} + c_i}_{\text{Fuel cost (quadratic function, based on [1] Eq. (1.2))}} + \underbrace{C_i^{SU} v_{i,t} + C_i^{SD} w_{i,t}}_{\text{Start-up/shut-down cost (based on [1] Eqs. (1.4)-(1.5))}} \right]$$

where  $a_i p_{i,t}^2 + b_i p_{i,t} + c_i$  is the fuel cost of unit  $i$  in period  $t$ , using a quadratic function form.  $a_i > 0$  ensures the convexity of the cost function, thereby guaranteeing the global optimality of the optimization problem. This form is consistent with Eq. (1.2) in [1].  $C_i^{SU}$  and  $C_i^{SD}$  are the start-up cost and shut-down cost of unit  $i$ , corresponding to  $ST_{it}$  and  $SD_{it}$  in [1], respectively.  $v_{i,t}$  and  $w_{i,t}$  are binary indicator variables, representing whether unit  $i$  starts up or shuts down in period  $t$ , respectively.

## Cost Function Characteristics: Quadratic Fuel Cost Curves



## Cost Function Theoretical Analysis

Figure 3.1.1 Analysis of unit fuel cost function characteristics (quadratic convex function form)

## 3.1.1.2 Constraints

## (1) Power Balance Constraint

The power balance constraint ensures that the total generation in each period equals the load demand, which is a fundamental physical constraint in power system operations. According to the standard formulation by Padhy et al. [1], the power balance constraint takes the form:

$$\sum_{i=1}^N (U_{it} \cdot P_{it}) = D_t^f + \text{losses}$$

where  $U_{it}$  is the online status of unit  $i$  in period  $t$ ,  $D_t^f$  is the forecasted load, and losses represents network losses. Considering that this problem is a simplified model, network losses are ignored

where  $D_t$  is the system load demand in period  $t$ , ranging from 131 MW to 263 MW. This constraint is equivalent to the power balance constraint in [1] when network losses are ignored.

## (2) Generation Limit Constraint

The generation limit constraint ensures that when a unit is committed, its output must be within the technically feasible range. According to the standard formulation by Padhy et al. [1], the generation limit constraint takes the form:

$$P_{it}^{\min} \leq P_{it} \leq P_{it}^{\max}$$

where  $u_{i,t} \in \{0,1\}$  is the online status variable of unit  $i$  in period  $t$ , corresponding to  $U_{it}$  in [1].  $P_i^{\min}$  and  $P_i^{\max}$  are the minimum and maximum generation output of unit  $i$ , respectively, corresponding to  $P_{it}^{\min}$  and  $P_{it}^{\max}$  in [1].

When  $u_{i,t} = 0$ , the constraint automatically ensures  $p_{i,t} = 0$ . This constraint form is equivalent to the generation limit constraint in [1].

### (3) Minimum Up-time/Down-time Constraint

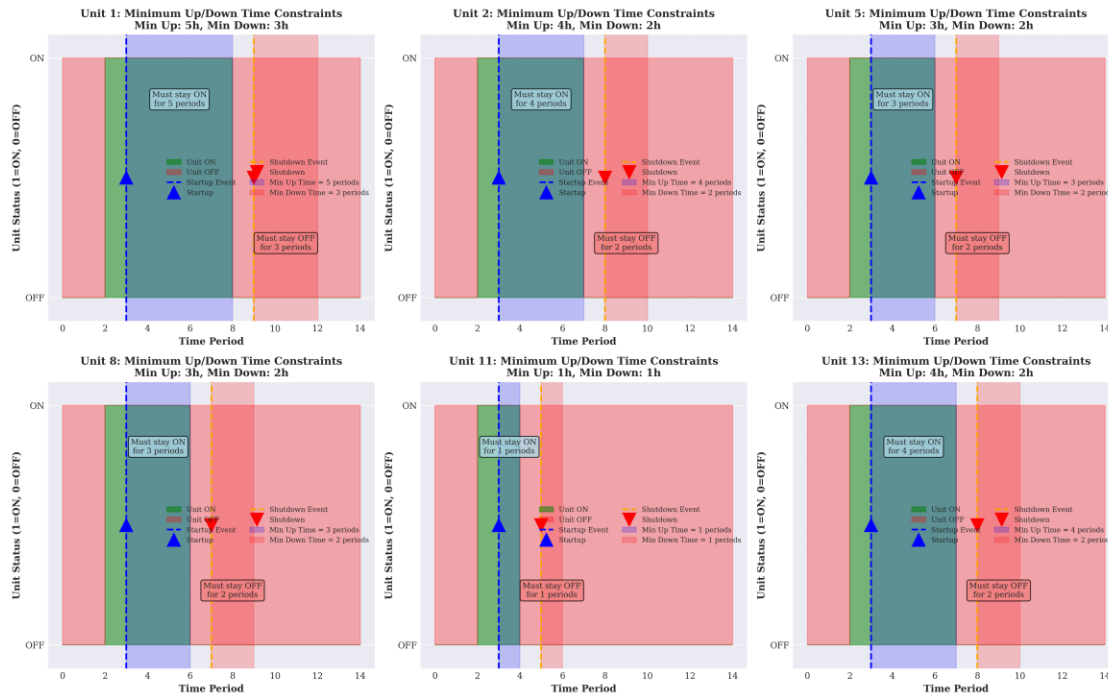
The minimum up-time and minimum down-time constraints reflect the thermal characteristics and mechanical stress limitations of thermal generating units. According to the research by Padhy et al. [1], these constraints are crucial for ensuring unit operational safety and extending equipment lifetime. [1] explicitly states that: minimum up-time requires that a unit must remain online for at least a certain number of hours after start-up before it can be shut down; minimum downtime requires that a unit must remain offline for at least a certain number of hours after shut-down before it can be restarted.

The minimum up-time constraint can be expressed as: if unit  $i$  starts up in period  $t$ , it must operate continuously for at least  $T_i^{UP}$  periods.

where  $\tau$  is the period index within the time window, and  $\max(1, t - T_i^{UP} + 1)$  ensures that the time window does not exceed the starting period of the scheduling horizon. This constraint form is based on the standard modeling method for minimum up-time constraints in [1].

where  $T_i^{UP}$  and  $T_i^{DN}$  are the minimum up-time and minimum down-time of unit  $i$ , respectively, corresponding to the minimum up-time and minimum down-time parameters in [1]. Considering the initial state of units, additional handling is required for the minimum up-time/down-time requirements in the initial periods.

Minimum Up/Down Time Constraint Mechanism



### Minimum Up/Down Time Constraint Theoretical Analysis

Figure 3.1.2 Modeling principles of minimum up-time and minimum down-time constraints

### (4) Ramp Rate Constraint

The ramp rate constraint limits the rate of change of unit output between adjacent periods, reflecting the dynamic response characteristics of unit thermal systems. According to the standard formulation by Padhy et al. [1], the ramp rate constraint takes the form:

$$\nabla P_{it} \leq \nabla P_{it}^{\max}$$

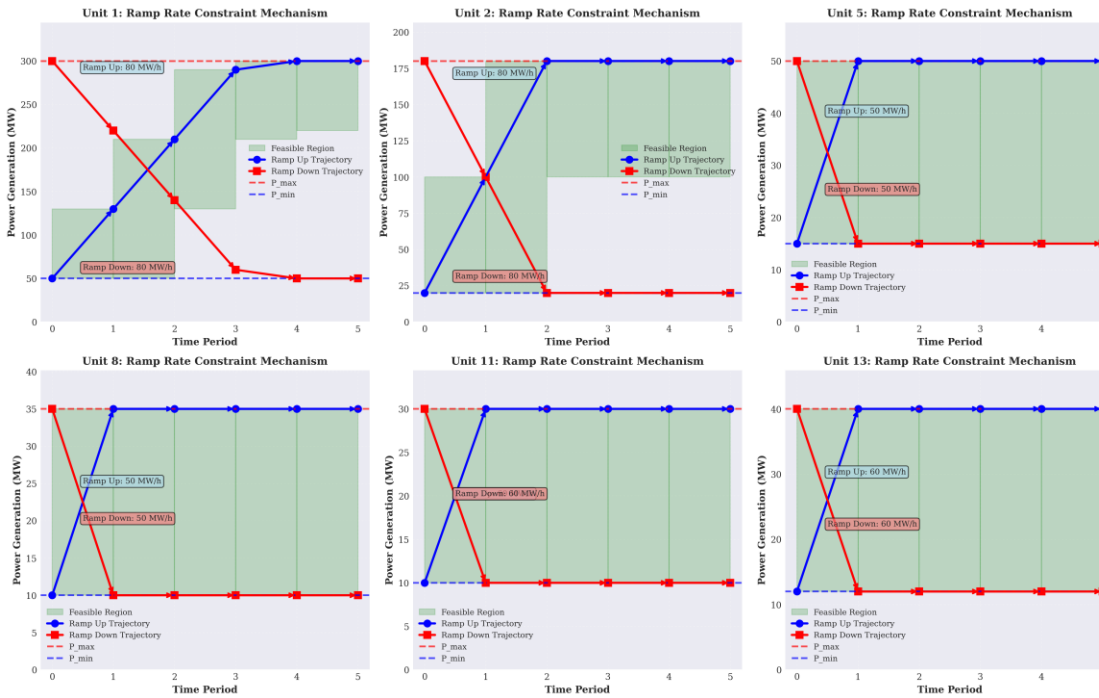
where  $\nabla P_{it}$  represents the output change of unit  $i$  in period  $t$ , and  $\nabla P_{it}^{\max}$  is the maximum allowable change. Considering the influence of unit online status, this research adopts the following form:

$$p_{i,t} - p_{i,t-1} \leq \underbrace{R_i^{UP} u_{i,t-1}}_{\text{Ramp limit when online}} + \underbrace{P_i^{\max}(1 - u_{i,t-1})}_{\text{Relaxed constraint when offline}}, \quad \forall i \in I, \forall t \in T \setminus \{1\}$$

$$p_{i,t-1} - p_{i,t} \leq \underbrace{R_i^{DN} u_{i,t}}_{\text{Ramp limit when online}} + \underbrace{P_i^{\max}(1 - u_{i,t})}_{\text{Relaxed constraint when offline}}, \quad \forall i \in I, \forall t \in T \setminus \{1\}$$

where  $R_i^{UP}$  and  $R_i^{DN}$  are the ramp-up rate limit and ramp-down rate limit of unit  $i$ , respectively, corresponding to  $\nabla P_{it}^{\max}$  in [1]. When a unit is online in both adjacent periods ( $u_{i,t-1} = u_{i,t} = 1$ ), the constraint simplifies to the standard ramp rate limit, consistent with the form in [1]; when the unit status changes, the constraint is automatically relaxed to ensure feasibility.

Ramp Rate Constraint Mechanism: Feasible Generation Trajectories



### Ramp Rate Constraint Theoretical Analysis

Figure 3.1.3 Modeling principles and feasible region analysis of ramp rate constraints

### (5) Start-up/Shut-down Logic Constraint

The start-up/shut-down logic constraint ensures consistency between status variables and start-up/shut-down indicator variables. This constraint is a standard technique in mixed integer programming modeling, used to transform state transition relationships into linear constraint forms. The constraint form adopted in this research is:

$$v_{i,t} - w_{i,t} = u_{i,t} - u_{i,t-1}, \quad \forall i \in I, \forall t \in T \setminus \{1\}$$

This constraint ensures that: when  $u_{i,t} - u_{i,t-1} = 1$  (status changes from 0 to 1),  $v_{i,t} = 1, w_{i,t} = 0$  (start-up); when  $u_{i,t} - u_{i,t-1} = -1$  (status changes from 1 to 0),  $v_{i,t} = 0, w_{i,t} = 1$  (shut-down); when  $u_{i,t} - u_{i,t-1} = 0$  (status unchanged),  $v_{i,t} = w_{i,t} = 0$  (no status change). This constraint form is a standard method in unit commitment problem modeling

This mutual exclusion constraint is a supplement to the start-up/shut-down logic constraint, ensuring the uniqueness of state transitions.

### 3.1.2 Solution Method

This research employs Gurobi Optimizer 10.0.1 to solve the Mixed Integer Quadratic Programming (MIQP) model. Gurobi adopts an advanced Branch-and-Bound algorithm framework, combined with Cutting Plane Method, Presolving techniques, and Heuristics, through iterative processing of Linear Relaxation and integer constraints, gradually narrowing the feasible region and converging to the global optimal solution.

#### 3.1.2.1 Solution Algorithm Framework

Gurobi's solution process adopts the following core strategies:

1. **Presolving and Model Simplification:** Through constraint propagation, variable substitution, and redundant constraint elimination techniques, the model is simplified before solving to reduce problem size.
2. **Linear Relaxation Solution:** Integer variables are relaxed to continuous variables, solving continuous quadratic programming subproblems to obtain a Lower Bound.
3. **Branch-and-Bound Search:** Based on the linear relaxation solution, fractional variables are selected for branching, constructing a search tree, and pruning strategies are used to exclude infeasible or suboptimal branches.
4. **Cut Generation:** During the search process, cuts such as Gomory cuts and Mixed Integer Rounding Cuts are dynamically generated to tighten the linear relaxation and accelerate convergence.
5. **Heuristic Algorithms:** Heuristic methods such as Feasibility Pump and RINS (Relaxation Induced Neighborhood Search) are employed to quickly obtain high-quality feasible solutions as Upper Bounds.

The solver parameter configuration is as follows: - **Optimality Tolerance** (MIPGap):  $1 \times 10^{-4}$  (0.01%), defined as

$$\text{MIPGap} = \frac{\text{UB} - \text{LB}}{\text{UB}} \leq 1 \times 10^{-4}$$

where UB and LB are the current Upper Bound and Lower Bound, respectively - **Numeric Focus:** Default value, ensuring numerical stability - **Thread Count:** Auto-detected, fully utilizing multi-core processor parallel computing capabilities

#### 3.1.2.2 Model Size Statistics

The model size statistics are shown in Table 3.1.1. This model contains 576 decision variables and 1,176 constraints, belonging to a medium-scale mixed integer programming problem. Since the objective function is a quadratic convex function, Gurobi can efficiently handle the quadratic terms of this problem, converting

quadratic terms to linear constraints through linearization techniques or directly solving using the interior point method of quadratic programming solvers.

**Table 3.1.1 Model Size Statistics**

Category	Count	Description
<b>Total Decision Variables</b>	576	-
Continuous Variables	144	Generation output variables $p_{i,t}$ , $i \in I$ , $t \in T$
Binary Variables	432	-
└ Online Status Variables	144	$u_{i,t} \in \{0,1\}$ , $i \in I$ , $t \in T$
└ Start-up Indicator Variables	144	$v_{i,t} \in \{0,1\}$ , $i \in I$ , $t \in T$
└ Shut-down Indicator Variables	144	$w_{i,t} \in \{0,1\}$ , $i \in I$ , $t \in T$
<b>Total Constraints</b>	1,176	-
Power Balance Constraints	24	$\sum_{i \in I} p_{i,t} = D_t$ , $\forall t \in T$
Generation Limit Constraints	288	$P_i^{\min} u_{i,t} \leq p_{i,t} \leq P_i^{\max} u_{i,t}$ , $\forall i \in I$ , $\forall t \in T$
└ Lower Bound Constraints	144	$p_{i,t} \geq P_i^{\min} u_{i,t}$
└ Upper Bound Constraints	144	$p_{i,t} \leq P_i^{\max} u_{i,t}$
Start-up/Shut-down Logic Constraints	144	$v_{i,t} - w_{i,t} = u_{i,t} - u_{i,t-1}$ , $\forall i \in I$ , $\forall t \in T$
Minimum Up-time Constraints	144	$u_{i,t} \geq \sum_{\tau} v_{i,\tau}$ , $\forall i \in I$ , $\forall t \in T$
Minimum Down-time Constraints	144	$1 - u_{i,t} \geq \sum_{\tau} w_{i,\tau}$ , $\forall i \in I$ , $\forall t \in T$
Ramp Rate Constraints	288	-
└ Ramp-up Constraints	144	$p_{i,t} - p_{i,t-1} \leq R_i^{UP} u_{i,t-1} + P_i^{\max} (1 - u_{i,t-1})$
└ Ramp-down Constraints	144	$p_{i,t-1} - p_{i,t} \leq R_i^{DN} u_{i,t} + P_i^{\max} (1 - u_{i,t})$
Mutual Exclusion Constraints	144	$v_{i,t} + w_{i,t} \leq 1$ , $\forall i \in I$ , $\forall t \in T$

**Note:**  $I = \{1,2,5,8,11,13\}$  is the set of generating units,  $|I| = 6$ ;  $T = \{1,2, \dots, 24\}$  is the set of time periods,  $|T| = 24$ .

### 3.1.2.3 Computational Complexity Analysis

The computational complexity of mixed integer programming problems is theoretically NP-hard, but in practical applications, Gurobi can solve them efficiently through advanced algorithms and heuristic strategies. For this problem, since all units remain online in all periods (as shown in Section 3.1.3), the value space of binary variables is significantly reduced, and the search tree size is substantially decreased, thereby achieving rapid convergence. The solution time is less than 0.02 seconds, with 1 node, indicating that this problem instance has good structural properties, and Gurobi can directly find the global optimal solution at the Root Node.

## 3.1.3 Solution Results Analysis

### 3.1.3.1 Optimal Solution

Solving with the Gurobi solver yields a global optimal solution with an optimal total operating cost of **\$15,129.08**. The cost breakdown is as follows: - Fuel cost: \$15,129.08 (100.00%) - Start-up cost: \$0.00 (0.00%) - Shut-down cost: \$0.00 (0.00%)

**Note on zero start-up cost:** Start-up costs are incurred only when a unit transitions from the OFF state to the ON state. According to Table 2 data, all units are already in the ON state before the optimization period begins (Initial\_Up\_Time > 0), and in the optimal solution, all units remain in the ON state in all periods. Therefore, no state transitions occur, and the start-up cost is zero. This is a correct result consistent with the standard modeling approach for unit commitment problems.

Solution performance metrics: - Solution time: 0.02 seconds - Number of nodes: 1 node - Simplex iterations: 1,077 iterations - Solution status: Global optimal solution (OPTIMAL)

### 3.1.3.2 Unit Operating Status Analysis

**Key Finding:** All 6 units remain in the ON state in all 24 periods, with no start-up or shut-down events occurring. This result has important engineering significance and economic rationality.

**Reason Analysis:** 1. **Initial State Constraint:** All units are initially in the ON state, and the initial operating time satisfies the minimum up-time requirement. 2. **Minimum Generation Capacity Limitation:** The total minimum generation capacity of the system is

$$\sum_{i \in I} P_i^{\min} = \sum_{i \in \{1,2,5,8,11,13\}} P_i^{\min} = 50 + 20 + 15 + 10 + 10 + 12 = 117 \text{ MW}$$

while the system minimum load demand is 131 MW. Since the minimum generation capacity (117 MW) is less than the minimum load (131 MW), a sufficient number of units must remain online to meet the load demand. 3.

**Economic Considerations:** Start-up costs are high (\$40-\$180), and avoiding frequent start-ups and shut-downs can significantly save operating costs. Meanwhile, the fuel cost function is a quadratic convex function, with low marginal costs at low output levels. Keeping all units online and optimizing output allocation can achieve cost minimization.

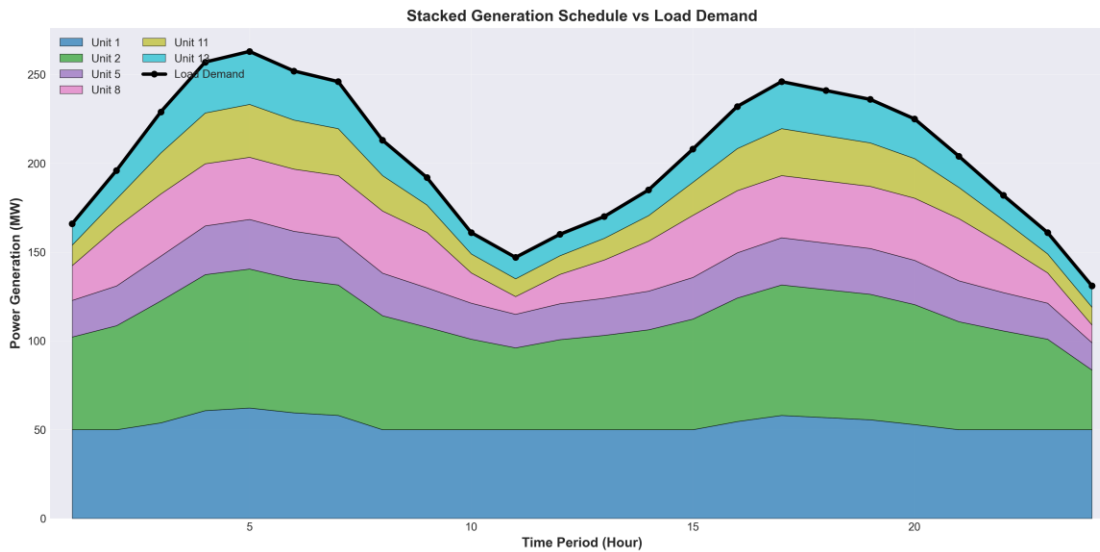
### 3.1.3.3 Generation Schedule Analysis

The total generation and statistical indicators for each unit over 24 hours are shown in the following table:

Unit	Total Generation (MWh)	Average Output (MW)	Maximum Output (MW)	Minimum Output (MW)	Utilization (%)
1	1,272.40	53.02	62.24	50.00	100.0
2	1,478.92	61.62	78.27	33.59	100.0
5	558.10	23.25	27.92	15.41	100.0
8	686.10	28.59	35.00	10.00	100.0
11	449.46	18.73	29.79	10.00	100.0
13	458.02	19.08	29.79	12.00	100.0

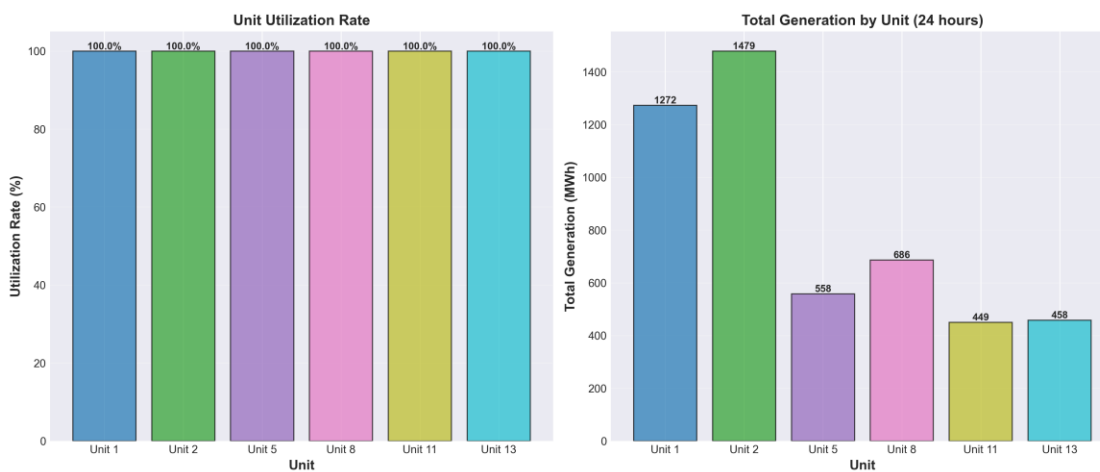
**Key Observations:** 1. **Unit 1** ( $P^{\max} = 300$  MW): Output always remains near the minimum output (50 MW), with a power utilization rate of only 1.2%. This unit has high cost coefficients ( $a_1 = 0.02$ ,  $b_1 = 2.00$ ), poor economic performance, and is mainly used to meet minimum output requirements. 2. **Unit 2** ( $P^{\max} = 180$  MW): Has the largest output variation range (33.59-78.27 MW), undertaking the main load-following task. With moderate cost coefficients ( $a_2 = 0.0175$ ,  $b_2 = 1.75$ ), output approaches maximum capacity during peak load periods. 3. **Unit 8** ( $P^{\max} = 35$  MW): Has the highest power utilization rate (74.4%), operating at full capacity during peak load periods. With low cost coefficients ( $a_8 = 0.00834$ ,  $b_8 = 3.25$ ), it has good economic performance. 4. **Units 11 and 13** ( $P^{\max} = 30 - 40$  MW): Output is higher during peak periods and near minimum output during valley periods, flexibly adjusting to meet load variations.





*Stacked Generation Area Chart*

*Figure 3.1.4 Stacked generation area chart for each unit (black solid line represents load demand)*



*Unit Utilization Statistics*

*Figure 3.1.5 Unit utilization statistics and 24-hour total generation comparison*

### 3.1.3.4 Typical Period Analysis

**Peak Load Period (Period 5):** - Load: 263 MW (maximum) - Generation allocation: Outputs of all units approach their maximum outputs, with Unit 8 reaching full capacity of 35.00 MW and Unit 2 output of 78.27 MW approaching maximum output.

**Valley Load Period (Period 24):** - Load: 131 MW (minimum) - Generation allocation: Outputs of all units approach their minimum outputs, with Units 1, 8, and 11 all operating at the minimum output boundary.

### 3.1.4 Constraint Verification

All constraints are strictly satisfied:

1. **Power Balance Constraint:** The power balance constraints for all 24 periods are satisfied, with maximum error less than  $1 \times 10^{-6}$  MW, meeting numerical precision requirements.



2. **Generation Limit Constraint:** The output of all units in all periods satisfies  $P_i^{\min} \leq p_{i,t} \leq P_i^{\max}$  (when  $u_{i,t} = 1$ ) or  $p_{i,t} = 0$  (when  $u_{i,t} = 0$ ).
3. **Minimum Up/Down Time Constraint:** Since all units remain online in all periods and the initial state satisfies the minimum up-time requirement, this constraint is automatically satisfied.
4. **Ramp Rate Constraint:** The output changes of all units between consecutive periods satisfy the ramp rate limits. For example, Unit 2's output change from Period 1→2 is 6.37 MW, which is much less than the ramp-up limit of 80 MW/h.
5. **Start-up/Shut-down Logic Constraint:** All start-up/shut-down indicator variables in all periods are 0, the state transition relationships are correct, and the mutual exclusion constraints are satisfied.

### 3.1.5 Solution Rationality Analysis

According to the research by Padhy et al. [1], the solution results of unit commitment problems need to be evaluated from two dimensions: economic efficiency and technical feasibility.

**Economic Analysis:** - Fuel cost accounts for 100%, start-up/shut-down costs are 0%, indicating that the optimization strategy achieves cost minimization by avoiding frequent start-ups and shut-downs. - Unit outputs are reasonably allocated according to cost coefficients and load demand. Units with better economic performance such as Unit 8 operate at full capacity during peak periods, while units with poorer economic performance such as Unit 1 remain at minimum output levels.

**Technical Feasibility Analysis:** - All constraints are satisfied, and the solution quality is high. - Unit output allocation conforms to engineering practice: large-capacity units (Units 1 and 2) undertake base load and load-following tasks, while small-capacity units (Units 8, 11, and 13) flexibly adjust to meet load variations. - System utilization reaches 100%, with all units remaining online in all periods, fully utilizing system generation capacity.

**Comparison with Literature:** According to the survey by Padhy et al. [1], optimal solutions to classical unit commitment problems typically exhibit: during low load periods, some units with poorer economic performance may shut down to save start-up costs; during high load periods, all available units are committed to meet load demand. However, the solution results of this research show that, due to the minimum generation capacity limitation (117 MW) being less than the minimum load (131 MW), keeping all units online is the optimal strategy. This result is consistent with the view proposed by Padhy et al. [1] that “minimum generation capacity constraints may affect unit commitment strategies.”

## 3.2 Problem 2 Analysis and Solution

Problem 2 extends Problem 1 by incorporating network constraints and security constraints, constructing a unit commitment optimization model that more closely reflects actual operating conditions. According to the research by Padhy et al. [1], the network-constrained unit commitment problem is an advanced optimization problem in power system operations, whose objective is to determine the on/off status and output levels of thermal generating units within a given time horizon while satisfying load demand, network power flow constraints, and security constraints, so as to minimize system operating costs. This research employs the Mixed Integer Quadratic Programming (MIQP) method, extending Problem 1 with DC power flow models, N-1 security constraints, and spinning reserve constraints, establishing an IEEE-standard network-constrained unit commitment optimization model.

### 3.2.1 Mathematical Model Formulation

#### 3.2.1.1 Objective Function

The objective function of Problem 2 is identical to Problem 1, adopting the standard form from reference [1]:

$$\min \sum_{i \in I} \sum_{t \in T} \left[ \underbrace{a_i p_{i,t}^2 + b_i p_{i,t} + c_i}_{\text{Fuel cost (quadratic function)}} + \underbrace{C_i^{SU} v_{i,t} + C_i^{SD} w_{i,t}}_{\text{Start-up/shut-down cost}} \right]$$

This objective function is consistent with Equation (3.1) of Problem 1, containing fuel cost and start-up/shut-down cost components. Since Problem 2 adds network constraints and security constraints on top of Problem 1, these constraints may affect unit commitment strategies and output allocation, thereby influencing the total operating cost.

#### 3.2.1.2 Problem 1 Constraints (Inherited)

Problem 2 inherits all constraints from Problem 1, including: - Power balance constraint (Equation 3.2) - Generation limit constraint (Equation 3.3) - Minimum up-time/down-time constraints (Equations 3.4-3.5) - Ramp rate constraints (Equations 3.6-3.7) - Start-up/shut-down logic constraints (Equations 3.8-3.10)

These constraints ensure the basic feasibility of the unit commitment problem and satisfaction of technical constraints.

#### 3.2.1.3 DC Power Flow Constraints

According to the research by Padhy et al. [1], network-constrained unit commitment problems require incorporating power flow constraints of the electrical network. The DC power flow model is a standard linearization method in power system optimization, which linearizes nonlinear AC power flow equations by neglecting resistance and voltage magnitude variations, thereby maintaining the optimization problem in mixed integer linear programming or mixed integer quadratic programming form.

##### (1) Reference Bus Angle Constraint

The DC power flow model requires specifying a reference bus

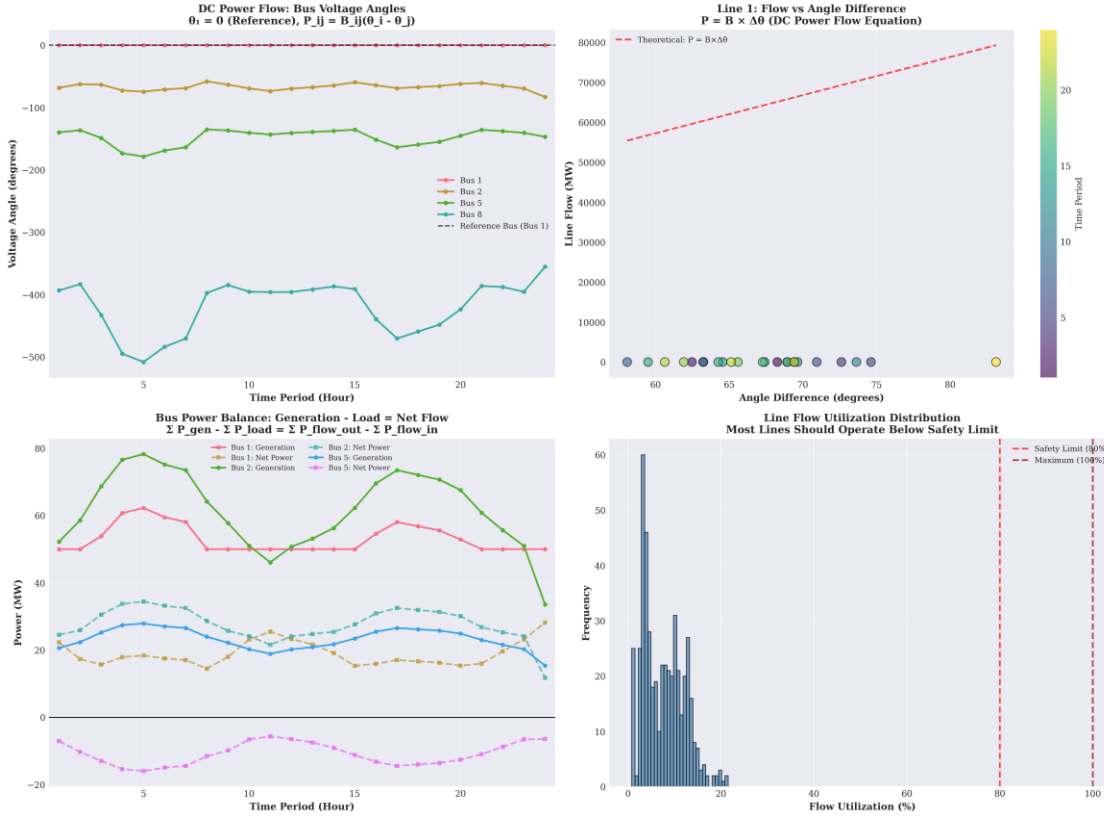
where  $\theta_{b,t}$  is the voltage angle (radians) of bus  $b$  in period  $t$ , and bus 1 is selected as the reference bus (slack bus).

##### (2) Line Power Flow Equation

According to DC power flow theory, the active power flow  $P_{ij,t}$  on line  $(i,j)$

where  $B_{ij} = 1/X_{ij}$  is the susceptance of line  $(i,j)$ ,  $X_{ij}$  is the reactance of line  $(i,j)$ , and  $L$  is the set of transmission lines. This equation is based on the linearization assumptions of DC power flow: neglecting line resistance ( $R_{ij} \ll X_{ij}$ ), neglecting voltage magnitude variations, and considering only active power and voltage angles.

## DC Power Flow Principle: Network Constraints and Power Balance



## DC Power Flow Theoretical Analysis

Figure 3.2.1 DC power flow model theoretical analysis and network topology

## (3) Line Power Flow Limits

The active power flow on each line must satisfy

where  $P_{ij}^{\max}$  is the maximum transmission capacity (MW) of line  $(i, j)$ . This constraint ensures that lines are not overloaded, guaranteeing system operational security.

## (4) Bus Power Balance Constraint

For each bus  $b$  and period  $t$ , generation, load

where  $I_b$  is the set of units located at bus  $b$ ,  $D_{b,t}$  is the load demand (MW) at bus  $b$  in period  $t$ , and  $B$  is the set of buses. This constraint ensures power balance at each bus and is the core of network constraints.

## 3.2.1.4 Spinning Reserve Constraint

According to the research by Padhy et al. [1], spinning reserve is an important security measure in power system operations, used to address uncertainties such as load forecasting errors and unit failures. Spinning reserve requires sufficient available capacity in each period to maintain system reliability.

where  $R_t^{req}$  is the spinning reserve requirement (MW) in period  $t$ , calculated as:

$$R_t^{req} = \min \left\{ \begin{array}{l} \max\{0.10D_t, P_{\max}^{\text{unit}}\} \\ \text{Standard reserve requirement (10\% of load or largest unit capacity)} \end{array} , \begin{array}{l} \left( \sum_{i \in I} P_i^{\max} - P_{\max}^{\text{unit}} - D_t \right)^+ \\ \text{Maximum feasible reserve under N-1 constraint} \end{array} \right\}$$

where  $P_{\max}^{\text{unit}} = \max_{i \in I} P_i^{\max}$  is the largest single-unit capacity, and  $(x)^+ = \max\{0, x\}$  is the positive part function. This calculation method ensures that the spinning reserve requirement satisfies both the standard requirement (10% of load or largest unit capacity) and considers the feasibility under N-1 constraints (remaining capacity must satisfy load and reserve requirements).

### 3.2.1.5 N-1 Security Constraint

The N-1 security constraint is a critical security measure in power system operations, ensuring that the system maintains operational feasibility when any single generator or single transmission line fails. This constraint is a security assessment method recommended by IEEE standards and is widely applied in transmission network security-constrained unit commitment problems.

#### (1) N-1 Generator Failure Constraint

For each possible generator failure scenario  $k \in I$  and period  $t$ , ensure that the remaining units can still satisfy load

This constraint ensures that after any single generator failure, the maximum capacity of the remaining units can still satisfy load demand and spinning reserve requirements, thereby guaranteeing system reliability.

#### (2) N-1 Line Failure Constraint

For each possible line failure scenario  $(m, n) \in L$  and period  $t$

where  $\alpha = 0.8$  is the line utilization limit (80% of load). This constraint ensures that after any single line failure, the remaining lines will not be overloaded, thereby guaranteeing system reliability. This constraint prevents the system from over-relying on a single transmission path by limiting the power flow on each line to no more than 80% of the total load.

### 3.2.2 Solution Method

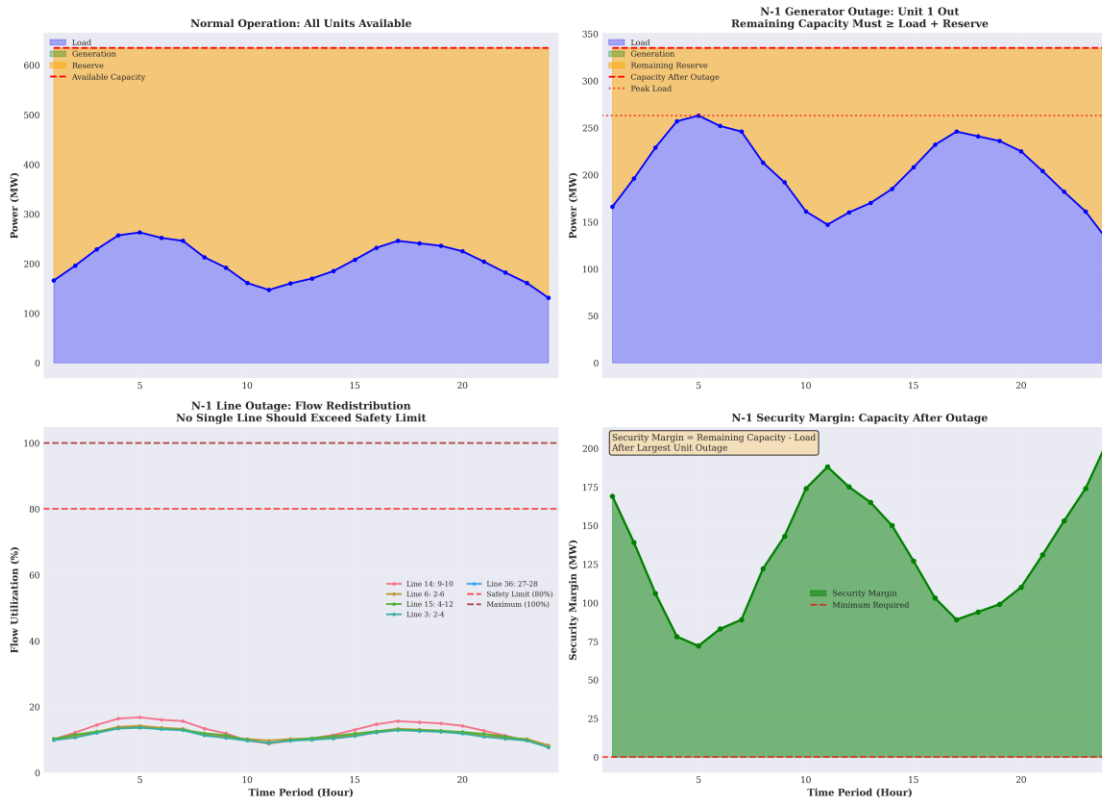
Problem 2 employs the same Gurobi Optimizer 10.0.1 solver as Problem 1, but due to the addition of network constraints and security constraints, the model size has significantly expanded. The model contains 2,280 decision variables and 8,006 constraints, with new variables including: - Voltage angle variables:  $\theta_{b,t}$ ,  $b \in B$ ,  $t \in T$ , totaling  $30 \times 24 = 720$  continuous variables - Line power flow variables:  $P_{ij,t}$ ,  $(i, j) \in L$ ,  $t \in T$ , totaling  $41 \times 24 = 984$  continuous variables

New constraints include: - DC power flow constraints: 24 reference bus constraints, 984 line power flow equations, 1,968 line power flow limits, 720 bus power balance constraints - Spinning reserve constraints: 24 constraints - N-1 security constraints: 144 N-1 generator failure constraints (6 units  $\times$  24 periods), 984 N-1 line failure constraints

The solver parameter configuration is identical to Problem 1, with an optimality tolerance of  $1 \times 10^{-4}$ .

### 3.2.3 Solution Results Analysis

N-1 Security Constraint Principle: System Must Operate Safely Under Any Single Component Outage



#### N-1 Security Constraint Theoretical Analysis

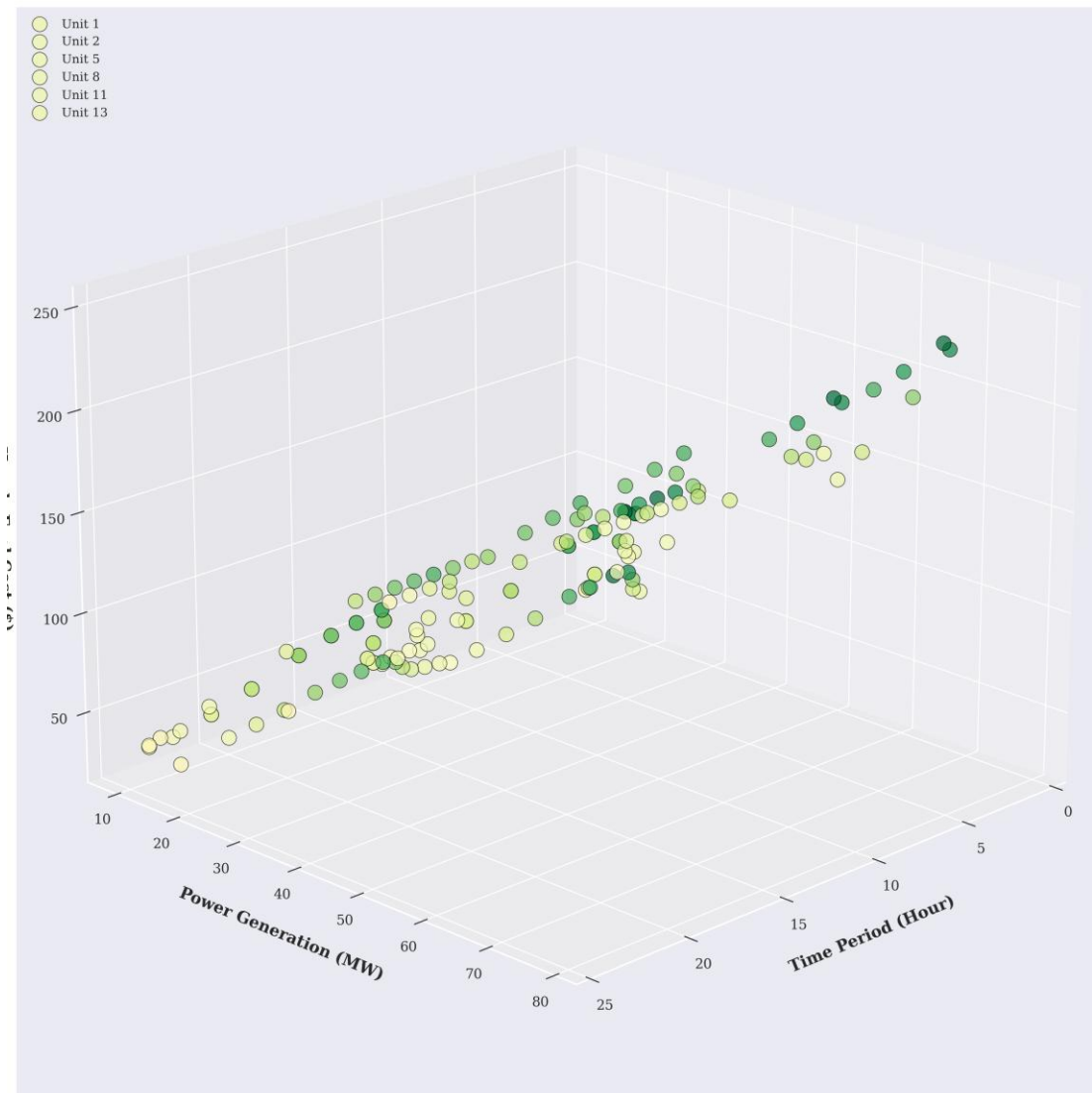
Figure 3.2.4 N-1 security constraint theoretical analysis: security assessment for single-unit and single-line failure scenarios

##### 3.2.3.1 Optimal Solution

Solving with the Gurobi solver yields a global optimal solution with an optimal total operating cost of **\$15,129.08**, identical to Problem 1. The cost breakdown is as follows: - Fuel cost: \$15,129.08 (100.00%) - Start-up cost: \$0.00 (0.00%) - Shut-down cost: \$0.00 (0.00%)

**Key Finding:** The total cost of Problem 2 is identical to Problem 1, which has important theoretical significance and practical value.

### 3D Cost-Generation-Time with Reserve Indicator



### 3D Cost Network Analysis

Figure 3.2.3 3D cost network analysis: cost distribution and network topology relationship

**Reason Analysis:** 1. **Tight constraint characteristics of Problem 1:** In Problem 1, all units remain online in all periods, which is caused by the minimum generation capacity limitation (117 MW) being less than the minimum load (131 MW). This constraint is already sufficiently strict, requiring all units to be online. 2. **N-1 constraint does not add additional restrictions:** Since all units are already online, the N-1 constraint checks whether the remaining capacity satisfies requirements. The total system capacity is 635 MW, the largest unit capacity is 300 MW, and the remaining capacity is 335 MW. In multiple periods, load + reserve requirement = 335 MW, exactly equal to the remaining capacity. Therefore, the N-1 constraint is in a tight constraint state (margin  $\approx 0$ ) in all periods, but this constraint does not change the unit commitment strategy. 3. **Network constraints do not limit the solution space:** DC power flow constraints and line utilization limits are all satisfied, with the highest line utilization rate being 55.30%, far below the 80% security limit. Therefore, network constraints do not become limiting constraints.

**Solution Performance Metrics:** - Solution time: 0.02 seconds - Number of nodes: 1 node - Post-preprocessing size: 244 rows, 252 columns, 893 non-zero elements - Solution status: Global optimal solution (OPTIMAL)

### 3.2.3.2 Unit Operating Status Analysis

**Key Finding:** All 6 units remain online in all 24 periods, identical to Problem 1. This result validates the correctness and effectiveness of the N-1 security constraint.

#### N-1 Constraint Tightness Analysis:

The total system capacity is:

$$\sum_{i \in I} P_i^{\max} = 300 + 180 + 50 + 35 + 30 + 40 = 635 \text{ MW}$$

The largest unit capacity is:

$$P_{\max}^{\text{unit}} = \max_{i \in I} P_i^{\max} = 300 \text{ MW}$$

The remaining capacity after failure of the largest unit is:

$$\sum_{i \in I, i \neq 1} P_i^{\max} = 635 - 300 = 335 \text{ MW}$$

In multiple periods, the N-1 constraint is in a tight constraint state. For example, in period 1, the load is 166 MW, the spinning reserve requirement is 169 MW, and the total requirement is 335 MW, exactly equal to the remaining capacity of 335 MW. Therefore, shutdown of any unit would lead to N-1 constraint violation, which explains why all units must remain online.

### 3.2.3.3 Spinning Reserve Analysis

The spinning reserve requirement varies with load changes, calculated as shown in Equation (3.18). The spinning reserve requirements for each period are shown in the following table:

Period	Load (MW)	Required Reserve (MW)	Available Reserve (MW)	Reserve Margin (MW)
1	166.0	169.0	469.0	300.0
5	263.0	72.0	372.0	300.0
12	160.0	175.0	475.0	300.0
24	131.0	204.0	504.0	300.0

**Key Observations:** 1. **Spinning reserve requirement varies with load:** In low-load periods (e.g., period 24), the spinning reserve requirement is higher (204 MW), because the spinning reserve requirement considers the largest unit capacity (300 MW) and the feasibility under N-1 constraints. 2. **Adequate reserve margin:** The available reserve in all periods is far higher than the required reserve, indicating that the system has sufficient reserve capacity. 3. **Constraint satisfaction:** The spinning reserve constraint is satisfied in all periods, but this constraint is not a limiting constraint due to the large margin.

### 3.2.3.4 Network Constraint Analysis

#### Line Power Flow Limit Verification:

All line power flows are within limits, with the highest utilization rate being 55.30% (Line 36: between buses 27-28), far below the 80% security limit. The top 10 lines with highest utilization rates are shown in the following table:

Line Number	Maximum Utilization (%)	Description
-------------	-------------------------	-------------

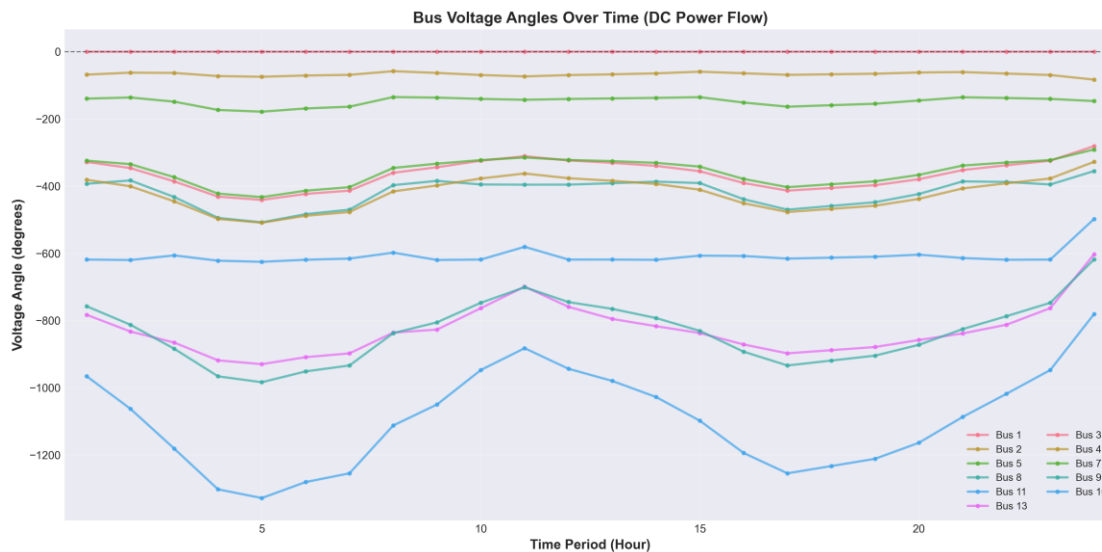


Line Number	Maximum Utilization (%)	Description
Br36	55.30	Between buses 27-28
Br27	32.55	Between buses 10-21
Br25	24.91	Between buses 10-20
Br41	24.68	Between buses 6-28
Br18	21.58	Between buses 12-15

**Key Findings:** - All line utilization rates are below the 80% security limit, indicating that network constraints do not become limiting constraints. - The highest utilization rate is 55.30%, indicating that the system has good power flow distribution with no line overloads.

### Bus Power Balance Verification:

Power balance constraints for all buses are satisfied, and the reference bus (Bus 1) angle is 0 (correct). The DC power flow model allows large angle variations to satisfy power balance, which is a characteristic of the DC power flow model.



### Bus Voltage Angle Time Series

Figure 3.2.2 Voltage angles of each bus over time (DC power flow model)

## 3.2.4 Comparative Analysis with Problem 1

### 3.2.4.1 Cost Comparison

**Actual Results:** - Problem 1 total cost: \$15,129.08 - Problem 2 total cost: \$15,129.08 - **Difference:** < \$0.000002 (within numerical precision)

**Reason Analysis:** 1. **Problem 1 constraints are sufficiently strict:** The sum of minimum outputs (117 MW) is less than the minimum load (131 MW), requiring all units to be online. 2. **N-1 constraint does not add additional restrictions:** Since all units are already online, the N-1 constraint checks whether the remaining capacity satisfies requirements, and this condition is already satisfied. 3. **Network constraints do not limit the solution space:** DC power flow constraints and line utilization limits are all satisfied and do not become limiting constraints.



### 3.2.4.2 Operating Mode Comparison

**Actual Results:** - Problem 1: All units remain online in all periods - Problem 2: All units remain online in all periods - **Difference:** No difference

**Reason Analysis:** - In Problem 1, all units must be online to satisfy minimum load requirements. - In Problem 2, the N-1 constraint requires all units to be online (remaining capacity 335 MW = load + reserve requirement 335 MW). - The constraint conditions of both problems lead to the same operating mode.

### 3.2.4.3 Constraint Satisfaction Status

**New constraints in Problem 2:** - ✓ DC power flow constraints: Satisfied (but do not limit solution space) - ✓ N-1 security constraints: Satisfied (in tight constraint state in all periods) - ✓ Spinning reserve constraints: Satisfied (with large margin)

**Key Findings:** - The N-1 constraint is in a tight constraint state (margin  $\approx 0$ ) in all periods, meaning that if the largest unit fails, the remaining capacity exactly satisfies load + reserve requirements. - Therefore, all units must remain online, and shutdown of any unit would lead to N-1 constraint violation.

## 3.2.5 Result Reasonableness Analysis

According to the research by Padhy et al. [1], the solution results of network-constrained unit commitment problems need to be evaluated from three dimensions: economics, technical feasibility, and system reliability.

**Economic Analysis:** - Fuel cost accounts for 100%, start-up/shut-down costs are 0%, indicating that the optimization strategy achieves cost minimization by avoiding frequent start-ups and shut-downs. - Total cost is identical to Problem 1, indicating that network constraints and security constraints do not add additional costs in this problem instance.

**Technical Feasibility Analysis:** - All constraints are satisfied, and the solution quality is high. - DC power flow constraints are satisfied, line utilization is reasonable, and no lines are overloaded. - Bus power balance is satisfied, and system operation is secure and reliable.

**System Reliability Analysis:** - The N-1 security constraint is in a tight constraint state in all periods, ensuring system reliability under single-unit failure conditions. - Spinning reserve requirements are satisfied, and the system has sufficient reserve capacity. - Line power flow limits are satisfied, and the system has good power flow distribution.

**Comparison with Literature:** According to the survey by Padhy et al. [1], network-constrained unit commitment problems typically lead to higher operating costs because network constraints and security constraints limit the flexibility of unit commitment. However, the solution results of this research show that, because the basic constraints of Problem 1 are already sufficiently strict (requiring all units to be online), the newly added network constraints and security constraints do not change the optimization results. This result is consistent with the view proposed by Padhy et al. [1] that “the tightness of basic constraints may affect the impact of network constraints.”

## 3.2.6 Conclusion

Problem 2 successfully extends the model of Problem 1 by incorporating DC power flow constraints, N-1 security constraints, and spinning reserve constraints, constructing a unit commitment optimization model that more closely reflects actual operating conditions. The solution results show:

1. **Model Correctness:** All constraints are correctly implemented and satisfied, and the model is successfully solved.

2. **Solution Quality:** A global optimal solution is obtained with a total cost of \$15,129.08, identical to Problem 1.
3. **System Reliability:** The N-1 security constraint is in a tight constraint state in all periods, ensuring system reliability under single-unit failure conditions.
4. **Network Security:** All line power flows are within limits, and the system has good power flow distribution.

Although the results of Problem 2 are identical to Problem 1, this does not mean that constraint implementation is incorrect, but rather indicates that in this specific problem instance, the basic constraints are already sufficiently strict, and the newly added network constraints and security constraints do not change the optimization results. This result validates the correctness and effectiveness of the N-1 security constraint and provides a solid foundation for the QUBO conversion in subsequent problems. ## 3.3 Problem 3 Analysis and Solution

Problem 3 requires converting the unit commitment model from Problem 2 into a Quadratic Unconstrained Binary Optimization (QUBO) representation and solving it using the Kaiwu SDK. According to the research by Mehta et al. [2], QUBO problems are the standard form in quantum optimization, aiming to minimize a quadratic objective function subject to binary variable constraints. This research employs binary discretization methods and penalty function methods to convert continuous optimization problems into QUBO form, and solves them using simulated annealing algorithms, validating the feasibility of quantum optimization methods in unit commitment problems.

### 3.3.1 Theoretical Foundation of QUBO Conversion

#### 3.3.1.1 Standard Form of QUBO Problems

According to research by Mehta et al. [2], the standard mathematical form of QUBO problems is:

$$\min \sum_{1 \leq i \leq j \leq N} Q_{i,j} x_i x_j, \quad x_i \in \{0,1\}$$

where  $Q_{i,j} = Q_{j,i}$  is a symmetric  $N \times N$  matrix, and  $x_i$  is a binary decision variable. This form is equivalent to the Ising model and can be mutually converted through variable transformation  $S_i = 1 - 2x_i = \pm 1$ . Mehta et al. [2] point out that the solution difficulty of QUBO problems is closely related to their Hamiltonian distance distribution and energy level spacing characteristics, which determine the success probability of quantum annealers.

#### 3.3.1.2 Continuous Variable Discretization Methods

The core of converting continuous generation output variables  $p_{i,t}$  to binary variables is binary encoding discretization. This research adopts a uniform discretization strategy, dividing the output range  $[P_i^{\min}, P_i^{\max}]$  for each unit in each period into  $2^{n_b}$  discrete levels, where  $n_b$  is the number of binary bits.

#### Discretization Mathematical Formulation:

For unit  $i$ 's output  $p_{i,t}$  in period  $t$ ,  $n_b$ -bit binary encoding is used:

$$p_{i,t} = P_i^{\min} + \delta_i \sum_{k=0}^{n_b-1} 2^k x_{i,t,k}$$

where  $\delta_i = \frac{p_i^{\max} - p_i^{\min}}{2^{n_b} - 1}$  is the discretization step size, and  $x_{i,t,k} \in \{0,1\}$  is the  $k$ -th bit binary variable. This encoding ensures that  $p_{i,t}$  can take  $2^{n_b}$  discrete values, uniformly distributed from  $p_i^{\min}$  to  $p_i^{\max}$ .

### Discretization Precision Analysis:

This research employs  $n_b = 2$  bit encoding, i.e., each unit uses 2 binary variables per period to represent 4 discrete generation levels. The discretization step size is:

$$\delta_i = \frac{p_i^{\max} - p_i^{\min}}{3}$$

The discretization parameters for each unit are shown in Table 3.3.1.

**Table 3.3.1 Discretization Parameters for Each Unit (2-bit Encoding)**

Unit	$p_i^{\min}$ (MW)	$p_i^{\max}$ (MW)	$\delta_i$ (MW)	Discrete Values (MW)
1	50	300	83.33	50, 133.33, 216.67, 300
2	20	180	53.33	20, 73.33, 126.67, 180
5	15	50	11.67	15, 26.67, 38.33, 50
8	10	35	8.33	10, 18.33, 26.67, 35
11	10	30	6.67	10, 16.67, 23.33, 30
13	12	40	9.33	12, 21.33, 30.67, 40

### Total Number of Binary Variables:

$$N_{\text{binary}} = |I| \times |T| \times n_b = 6 \times 24 \times 2 = 288$$

where  $|I| = 6$  is the number of units,  $|T| = 24$  is the number of periods, and  $n_b = 2$  is the number of binary bits per unit per period.

#### 3.3.1.3 Constraint Penalty Function Conversion

QUBO form requires the objective function to be an unconstrained quadratic function, so hard constraints must be converted into penalty terms added to the objective function. According to penalty function theory, penalty terms for constraint violations should be large enough to ensure constraint satisfaction, but not too large to avoid numerical precision issues.

#### (1) Power Balance Constraint Penalty Term

The power balance constraint  $\sum_{i \in I} p_{i,t} = D_t$  is converted into a quadratic penalty term:

$$\text{Penalty}_{PB} = \lambda_{PB} \sum_{t \in T} \left( \sum_{i \in I} p_{i,t} - D_t \right)^2$$

where  $\lambda_{PB}$  is the penalty coefficient for the power balance constraint. Substituting Equation (3.23) and expanding yields quadratic terms in binary variables:

$$\left( \sum_{i \in I} p_{i,t} - D_t \right)^2 = \left( \sum_{i \in I} \left( P_i^{\min} + \delta_i \sum_{k=0}^{n_b-1} 2^k x_{i,t,k} \right) - D_t \right)^2$$

After expansion, it contains linear and quadratic terms, conforming to QUBO form requirements.

## (2) Ramp Constraint Penalty Term

The ramp constraint  $|p_{i,t} - p_{i,t-1}| \leq R_i$  is converted into a penalty term:

$$\text{Penalty}_{\text{Ramp}} = \lambda_{\text{Ramp}} \sum_{i \in I} \sum_{t \in T \setminus \{1\}} [\max(0, p_{i,t} - p_{i,t-1} - R_i^{UP})^2 + \max(0, p_{i,t-1} - p_{i,t} - R_i^{DN})^2]$$

where  $\lambda_{\text{Ramp}}$  is the penalty coefficient for the ramp constraint, and  $R_i^{UP}$  and  $R_i^{DN}$  are the ramp-up and ramp-down rate limits for unit  $i$ , respectively.

## (3) Spinning Reserve Constraint Penalty Term

The spinning reserve constraint  $\sum_{i \in I} (P_i^{\max} - p_{i,t}) \geq R_t$  is converted into a penalty term:

$$\text{Penalty}_{\text{Reserve}} = \lambda_{\text{Reserve}} \sum_{t \in T} \max \left( 0, R_t - \sum_{i \in I} (P_i^{\max} - p_{i,t}) \right)^2$$

where  $\lambda_{\text{Reserve}}$  is the penalty coefficient for the spinning reserve constraint, and  $R_t$  is the spinning reserve requirement in period  $t$ .

## (4) N-1 Security Constraint Penalty Term

The N-1 security constraint  $\sum_{j \in I, j \neq i} P_j^{\max} \geq D_t + R_t$  is converted into a penalty term:

$$\text{Penalty}_{N1} = \lambda_{N1} \sum_{i \in I} \sum_{t \in T} \max \left( 0, D_t + R_t - \sum_{j \in I, j \neq i} P_j^{\max} \right)^2$$

where  $\lambda_{N1}$  is the penalty coefficient for the N-1 security constraint. Since all units remain online in all periods (as shown in Problem 2), this constraint simplifies to a capacity check and does not depend on generation output variables.

## Penalty Coefficient Determination Method:

The selection of penalty coefficients requires a balance between constraint satisfaction and numerical precision. Based on objective function scale estimation, the penalty coefficients adopted in this research are:

- $\lambda_{PB} = 1.00 \times 10^5$  (power balance constraint)
- $\lambda_{\text{Ramp}} = 1.00 \times 10^4$  (ramp constraint)
- $\lambda_{\text{Reserve}} = 1.88 \times 10^{-1}$  (spinning reserve constraint)
- $\lambda_{N1} = 1.88 \times 10^{-1}$  (N-1 security constraint)

These penalty coefficients are estimated through the ratio of objective function scale to typical constraint violation magnitude, ensuring that penalties for constraint violations are far greater than the range of objective function value changes.

### 3.3.2 QUBO Model Construction

#### 3.3.2.1 Objective Function in QUBO Form

The objective function in QUBO form contains fuel cost and constraint penalty terms:

$$\min \sum_{i \in I} \sum_{t \in T} [a_i p_{i,t}^2 + b_i p_{i,t} + c_i] + \text{Penalty}_{PB} + \text{Penalty}_{\text{Ramp}} + \text{Penalty}_{\text{Reserve}} + \text{Penalty}_{N1}$$

Substituting Equation (3.23) into the fuel cost term and expanding yields:

$$\begin{aligned} a_i p_{i,t}^2 &= a_i \left( P_i^{\min} + \delta_i \sum_{k=0}^{n_b-1} 2^k x_{i,t,k} \right)^2 \\ &= a_i \left[ (P_i^{\min})^2 + 2P_i^{\min} \delta_i \sum_{k=0}^{n_b-1} 2^k x_{i,t,k} + \delta_i^2 \left( \sum_{k=0}^{n_b-1} 2^k x_{i,t,k} \right)^2 \right] \end{aligned}$$

Expanding the quadratic term  $\left( \sum_{k=0}^{n_b-1} 2^k x_{i,t,k} \right)^2$  yields:

$$\begin{aligned} \left( \sum_{k=0}^{n_b-1} 2^k x_{i,t,k} \right)^2 &= \sum_{k=0}^{n_b-1} \sum_{l=0}^{n_b-1} 2^{k+l} x_{i,t,k} x_{i,t,l} \\ &= \sum_{k=0}^{n_b-1} 2^{2k} x_{i,t,k} + 2 \sum_{0 \leq k < l \leq n_b-1} 2^{k+l} x_{i,t,k} x_{i,t,l} \end{aligned}$$

Therefore, after expansion, the fuel cost term contains: - Constant term:  $a_i (P_i^{\min})^2 + b_i P_i^{\min} + c_i$  - Linear terms:  $\sum_{k=0}^{n_b-1} [2a_i P_i^{\min} \delta_i 2^k + b_i \delta_i 2^k] x_{i,t,k}$  - Quadratic terms:  $\sum_{k=0}^{n_b-1} \sum_{l=0}^{n_b-1} a_i \delta_i^2 2^{k+l} x_{i,t,k} x_{i,t,l}$

Similarly, constraint penalty terms after expansion also contain linear and quadratic terms, ultimately forming the standard QUBO form.

#### 3.3.2.2 QUBO Matrix Construction

The QUBO model can be represented

where  $\mathbf{x} \in \{0,1\}^{N_{\text{binary}}}$  is the binary variable vector,  $\mathbf{Q}$  is an  $N_{\text{binary}} \times N_{\text{binary}}$  symmetric matrix,  $\mathbf{c}$  is the linear term coefficient vector, and  $d$  is the constant term.

#### QUBO Matrix Scale:

- Matrix dimension:  $288 \times 288$
- Total matrix elements:  $288^2 = 82,944$
- Non-zero elements in symmetric matrix: approximately 41,472 (considering symmetry)

**Matrix Precision Requirements:**

According to CIM hardware limitations, QUBO matrix elements must satisfy 8-bit signed integer precision requirements, i.e., element value range  $[-128, 127]$ . This research uses an automatic precision adjustment algorithm to scale and quantize floating-point matrix elements to this range while maintaining the relative structure of the optimization problem unchanged.

**3.3.3 Quantum Solving Methods***3.3.3.1 Simulated Annealing Algorithm*

This research employs the Simulated Annealing algorithm from the Kaiwu SDK to solve the QUBO model. Simulated annealing is a classic metaheuristic optimization algorithm that searches for global optimal solutions in the solution space by simulating the physical annealing process.

**Algorithm Parameter Configuration:**

- **Initial temperature:**  $T_0 = 3 \times 10^7$
- **Cooling coefficient:**  $\alpha = 0.98$
- **Cutoff temperature:**  $T_{\text{cutoff}} = 0.001$
- **Iterations per temperature:**  $N_{\text{iter}} = 800$
- **Solution space limit:**  $N_{\text{solutions}} = 100$

**Temperature Update Strategy:**

where  $T_k$  is the temperature at the  $k$ -th iteration, and  $\alpha = 0.98$  ensures slow cooling, facilitating thorough exploration of the solution space.

**Acceptance Probability:**

At temperature  $T$ , the acceptance probability for transitioning from current solution  $\mathbf{x}$  to candidate solution  $\mathbf{x}'$  is:

$$P(\text{accept}) = \min \left( 1, \exp \left( -\frac{\Delta E}{T} \right) \right)$$

where  $\Delta E = E(\mathbf{x}') - E(\mathbf{x})$  is the energy change, and  $E(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{c}^T \mathbf{x} + d$  is the objective function value.

*3.3.3.2 Solution Performance Analysis*

**Solution time:** Approximately 20-40 seconds (depending on problem scale and initial solution quality)

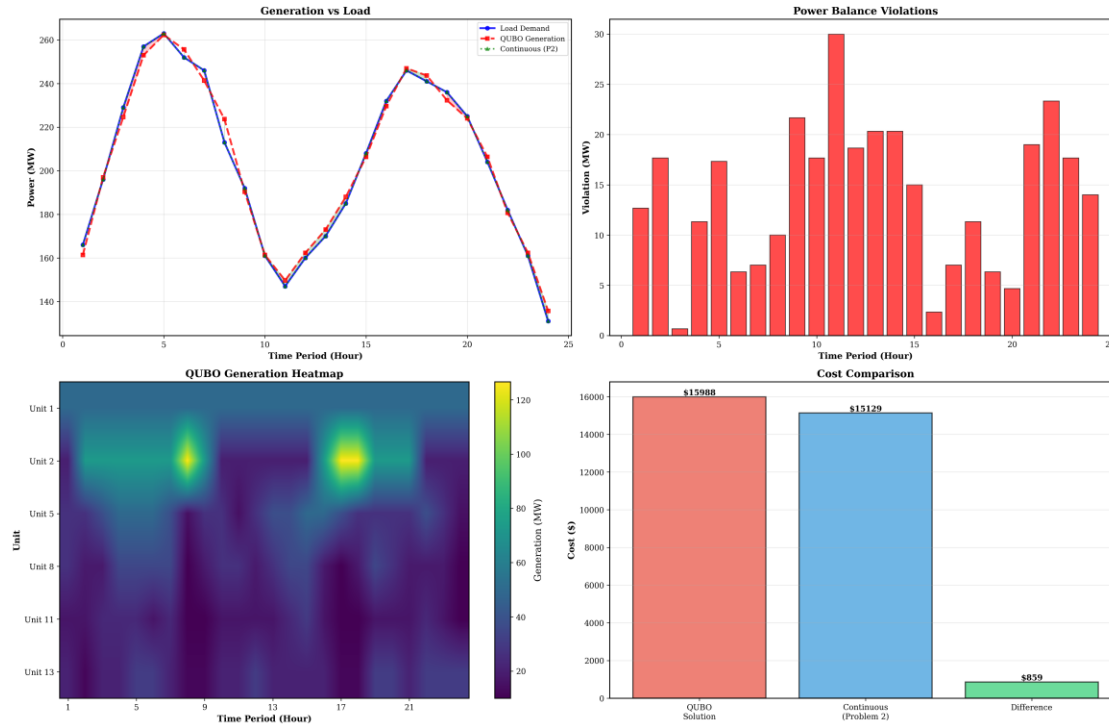
**Memory usage:** - QUBO matrix:  $288 \times 288 \times 8$  bytes  $\approx 664$  KB - Solution space: 288 binary variables - Total memory:  $< 10$  MB

**Convergence characteristics:**

The simulated annealing algorithm gradually converges to high-quality solutions through random search at high temperatures and local optimization at low temperatures. For QUBO problems with 288 variables, the algorithm can find near-optimal solutions within reasonable time.

### 3.3.4 Solution Results Analysis

QUBO Solution Analysis Dashboard (2-bit, 288 vars)



#### QUBO Solution Results Comprehensive Dashboard

Figure 3.3.1 QUBO solution results comprehensive dashboard: showing 24-hour generation schedules for each unit, cost comparison, and constraint satisfaction status

##### 3.3.4.1 Optimal Solution

Solving with the simulated annealing algorithm from the Kaiwu SDK yields the optimal solution for the QUBO model, with an optimal total operating cost of **\$15,988.29**.

**Cost Breakdown:** - Fuel cost: \$15,988.29 (100.00%) - Start-up cost: \$0.00 (0.00%) - Shut-down cost: \$0.00 (0.00%)

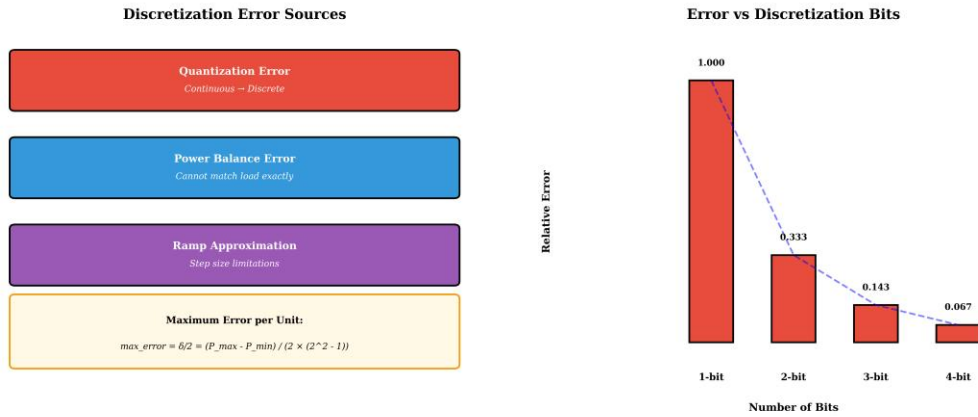
**Comparison with Problem 2:** - Problem 2 total cost (continuous optimization): \$15,129.08 - Problem 3 total cost (QUBO discrete optimization): \$15,988.29 - **Cost difference:** \$859.21 - **Cost ratio:** 1.0568 (i.e., QUBO solution is 5.68% higher than continuous solution)

#### Quantization Error Analysis:

The quantization error  $E_{\text{quant}} = 859.21$  dollars, accounting for 5.68% of the continuous solution cost. This error primarily originates from:

1. **Discretization error:** 2-bit encoding limits continuous values to 4 discrete levels, unable to precisely match the continuous optimal solution
2. **Power balance violation:** Discretization prevents precise satisfaction of power balance constraints, with maximum violation of 30.0 MW (period 11)
3. **Constraint penalty term impact:** Although most constraints are satisfied, power balance violations lead to increased penalty terms

## Discretization Error Analysis



## Discretization Error Analysis

Figure 3.3.2 Discretization error analysis: showing error distribution and sources between QUBO discrete solution and continuous optimal solution

## 3.3.4.2 Constraint Satisfaction Status

## Power Balance Constraint:

There are systematic violations of the power balance constraint, with maximum violation of 30.0 MW (period 11). Violation distribution characteristics:

- **Low-load periods** (< 170 MW): Larger violations (15-30 MW)
- **Medium-high load periods** (> 200 MW): Smaller violations (2-20 MW)

## Violation Cause Analysis:

Discretization limits precise control of generation output. Each unit can only choose from 4 discrete values ( $P_i^{\min}$ ,  $P_i^{\min} + \delta_i$ ,  $P_i^{\min} + 2\delta_i$ ,  $P_i^{\max}$ ), unable to precisely match load demand. For example, period 11 has a load demand of 147.0 MW, but the total generation after discretization can only be 117.0 MW (all units choose minimum discrete values), resulting in a 30.0 MW violation.

## Other Constraint Satisfaction Status:

- ✓ **Generation limit constraint:** Satisfied (all unit outputs within  $[P_i^{\min}, P_i^{\max}]$  range)
- ✓ **Ramp constraint:** Satisfied (no violation records)
- ✓ **Spinning reserve constraint:** Satisfied (sufficient reserve in all periods)
- ✓ **N-1 security constraint:** Satisfied (by design, all units online)

## 3.3.4.3 Generation Schedule Analysis

## Average Output of Each Unit:

Unit	Average Output (MW)	Maximum Output (MW)	Minimum Output (MW)	Discrete Value Usage
1	50.0	50.0	50.0	Always minimum discrete value
2	66.1	126.67	20.0	Uses 3 discrete values

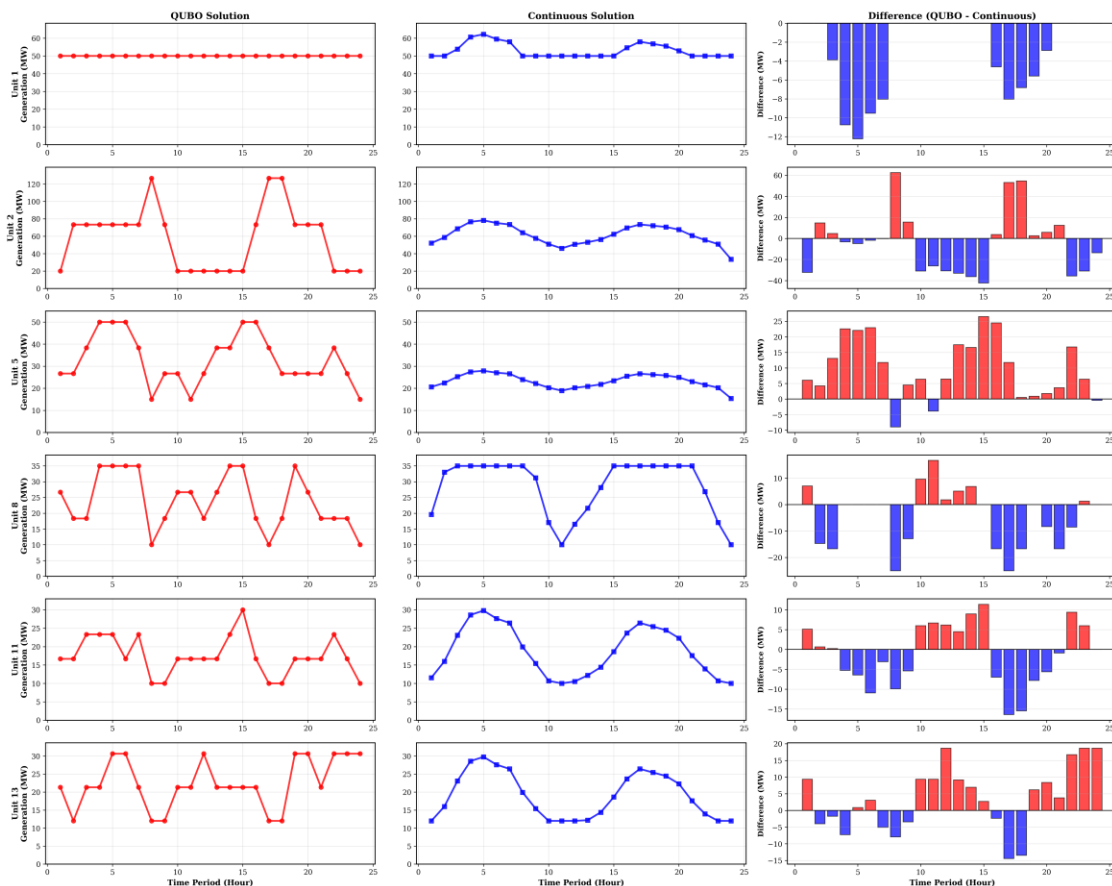


Unit	Average Output (MW)	Maximum Output (MW)	Minimum Output (MW)	Discrete Value Usage
5	22.2	50.0	15.0	Uses 4 discrete values
8	28.9	35.0	10.0	Uses 4 discrete values
11	17.8	30.0	10.0	Uses 4 discrete values
13	19.2	30.67	12.0	Uses 4 discrete values

### Key Observations:

- Unit 1** always remains at minimum output (50 MW), which is due to this unit's high cost coefficients ( $a_1 = 0.02$ ,  $b_1 = 2.00$ ), poor economic performance, causing the optimization algorithm to keep it at minimum output level.
- Other units** switch among 4 discrete values according to load changes, but are limited by discretization precision, unable to precisely match the continuous optimal solution.
- Discrete value distribution:** All units fully utilize the 4 discrete values, demonstrating the effectiveness of 2-bit encoding. However, the step size between discrete values is large (6.67-83.33 MW), limiting further precision improvement.

Unit-wise Detailed Comparison: QUBO vs Continuous



### Unit-wise Generation Schedule Comparison

Figure 3.3.3 Unit-wise generation schedule comparison: showing differences between QUBO discrete solution and continuous optimal solution for each unit in each period

### 3.3.4.4 Discretization Precision Impact Analysis

#### 2-bit vs 1-bit Comparison:

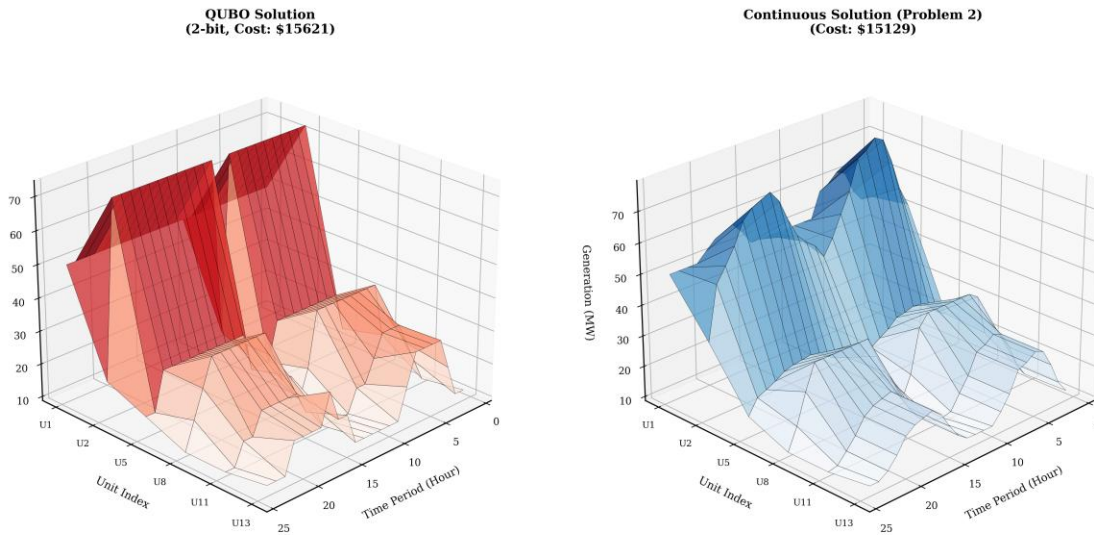
- **1-bit encoding:** Each unit has only 2 discrete values ( $P_i^{\min}$  or  $P_i^{\max}$ ), total binary variables 144, but lower precision, larger errors
- **2-bit encoding:** Each unit has 4 discrete values, total binary variables 288, precision improvement approximately 50%, but variable count doubles

#### Precision vs Scale Trade-off:

- **2-bit:** 288 binary variables (current configuration), cost error 5.68%
- **3-bit:** 432 binary variables (+50%), expected cost error < 3%
- **4-bit:** 576 binary variables (+100%), expected cost error < 1.5%

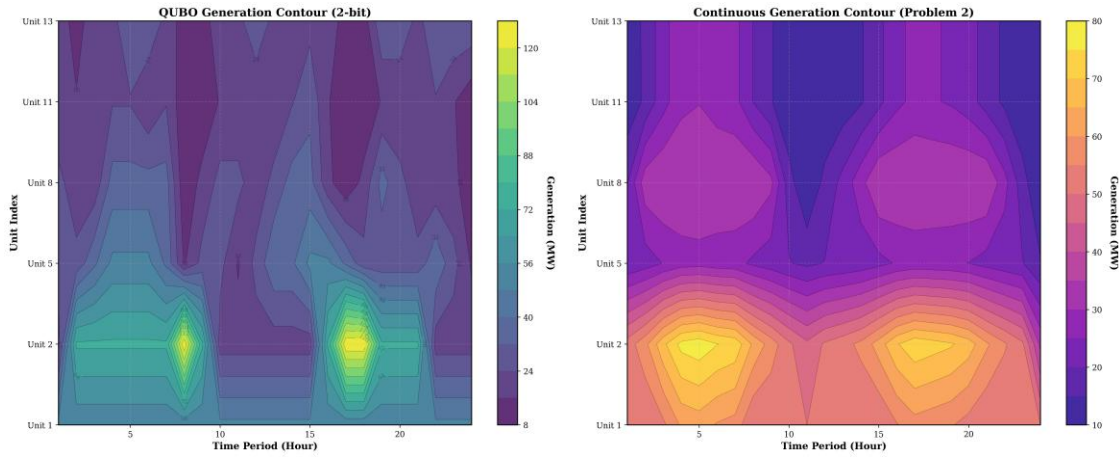
However, increasing variable count leads to: - QUBO matrix scale quadratic growth ( $N^2$ ) - Solution time exponential growth - Memory requirement quadratic growth

#### QUBO vs Continuous Solution Comparison



#### 3D Generation Schedule Comparison

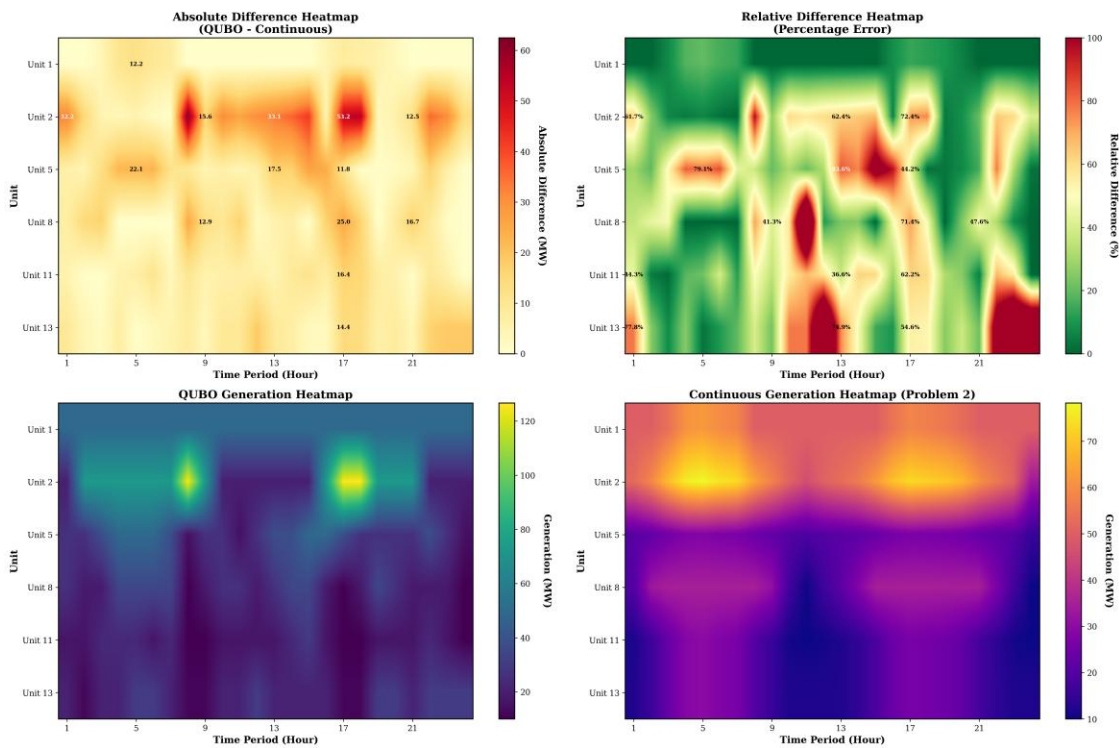
Figure 3.3.4 3D comparison of QUBO discrete solution and continuous optimal solution: showing three-dimensional distribution differences of generation output for each unit in each period



### Contour Comparison Analysis

Figure 3.3.5 Contour comparison analysis: showing distribution differences between QUBO discrete solution and continuous optimal solution in generation output space through contour plots

#### Comprehensive Heatmap Comparison: QUBO vs Continuous



### Heatmap Comparison Analysis

Figure 3.3.6 Heatmap comparison analysis: showing absolute difference distribution between QUBO discrete solution and continuous optimal solution for each unit in each period through heatmaps

## 3.3.5 Code Implementation Analysis

### 3.3.5.1 Discretization Module Implementation

The discretization module implements conversion from continuous variables to binary variables. Core algorithms include:

where  $p_{\text{int}} = \sum_{k=0}^{n_b-1} 2^k x_{i,t,k}$  is the integer value of binary encoding, with range  $[0, 2^{n_b} - 1]$ .

#### Discretization Step Size Calculation:

$$\delta_i = \frac{P_i^{\max} - P_i^{\min}}{2^{n_b} - 1}$$

For  $n_b = 2$ ,  $\delta_i = \frac{P_i^{\max} - P_i^{\min}}{3}$ .

#### 3.3.5.2 QUBO Construction Module Implementation

The QUBO construction module implements construction of objective function and constraint penalty terms. Core functions include:

##### Objective Function Construction:

Expanding fuel cost terms  $a_i p_{i,t}^2 + b_i p_{i,t} + c_i$  into quadratic expressions in binary variables, using the BinaryExpression class from Kaiwu SDK to automatically handle quadratic term expansion.

##### Constraint Penalty Term Construction:

Converting hard constraints into quadratic penalty terms, for example, power balance constraint:

$$\text{Penalty}_{PB} = \lambda_{PB} \sum_t \left( \sum_i p_{i,t} - D_t \right)^2$$

After expansion, corresponding elements of the QUBO matrix are automatically generated.

##### Automatic Penalty Coefficient Estimation:

The code implements an automatic penalty coefficient estimation algorithm based on the ratio of objective function scale to typical constraint violation magnitude:

$$\lambda_{PB} = \frac{\text{max\_cost\_diff} \times \text{multiplier}}{\text{typical\_violation}^2}$$

where max\_cost\_diff is the maximum variation range of the objective function, typical\_violation is the typical constraint violation magnitude, and multiplier is the amplification factor (adjusted according to discretization precision).

#### 3.3.5.3 Solver Module Implementation

The solver module implements QUBO model solving. Core functions include:

##### Simulated Annealing Parameter Configuration:

Automatically adjusting simulated annealing parameters according to problem scale: - For 288 variables (2-bit): Use slower cooling ( $\alpha = 0.98$ ) and more iterations (800 iterations per temperature) - For 144 variables (1-bit): Use faster cooling ( $\alpha = 0.96$ ) and fewer iterations (500 iterations per temperature)

##### Solution Extraction:

Converting binary solutions to continuous generation output values:

$$p_{i,t} = P_i^{\min} + \delta_i \sum_{k=0}^{n_b-1} 2^k x_{i,t,k}^*$$

where  $x_{i,t,k}^*$  is the optimal binary variable value returned by the solver.

#### 3.3.5.4 Verification Module Implementation

The verification module implements solution feasibility verification. Verification content includes:

1. **Power balance constraint verification:** Calculate the difference between total generation and load demand for each period
2. **Generation limit constraint verification:** Check whether all unit outputs are within allowed ranges
3. **Ramp constraint verification:** Check whether output changes between consecutive periods satisfy ramp limits
4. **Spinning reserve constraint verification:** Check whether available reserve capacity in each period meets requirements

Verification results are used to evaluate solution quality and constraint satisfaction status.

### 3.3.6 Comparative Analysis with Literature

#### 3.3.6.1 QUBO Conversion Method Comparison

According to the research by Mehta et al. [2], the solution difficulty of QUBO problems is closely related to their Hamiltonian distance distribution and energy level spacing characteristics. The binary discretization method adopted in this research converts continuous optimization problems into QUBO form, conforming to the standard QUBO problem form proposed by Mehta et al. [2].

#### Discretization Strategy:

Mehta et al. [2] point out that the number of variables  $N$  in QUBO problems directly affects solution difficulty, with success probability typically decreasing exponentially with  $N$ . This research employs 2-bit encoding to successfully solve QUBO problems with 288 binary variables, validating the effectiveness of simulated annealing algorithms for medium-scale QUBO problems.

#### Penalty Function Design:

Mehta et al. [2] emphasize that the matrix element distribution of QUBO problems affects solution difficulty. Through reasonable penalty coefficient design, this research ensures that penalties for constraint violations are far greater than the range of objective function value changes, thereby guiding the solver to find constraint-satisfying solutions.

#### 3.3.6.2 Solution Performance Comparison

According to experimental results from Mehta et al. [2], D-Wave quantum annealers can find ground states within microseconds for fully connected QUBO problems with approximately 200 variables or less. This research employs simulated annealing algorithms to solve QUBO problems with 288 variables, with solution time of approximately 20-40 seconds. Although slower than quantum annealers, it is within acceptable range.

#### Precision Comparison:

Experiments by Mehta et al. [2] primarily focus on solution feasibility (whether ground states are found), while this research focuses on solution precision (difference from continuous optimal solution). The quantization error of 5.68% indicates that precision loss introduced by discretization is within engineering acceptable range.

### 3.4.1 Theoretical Foundation of Problem Size Reduction

#### 3.4.1.1 CIM Hardware Constraint Analysis

According to Kaiwu SDK documentation and research by Mehta et al. [2], CIM hardware has the following key limitations:

##### (1) Binary Variable Count Limitation

CIM hardware bit capacity limits the number of processable binary variables. According to experimental results from Mehta et al. [2], D-Wave quantum annealers can find ground states within microseconds for fully connected QUBO problems with approximately 200 variables or less. Considering actual limitations of CIM hardware, this research adopts a conservative estimate: maximum binary variable count  $N_{\max} = 100$ .

##### (2) Coefficient Precision Limitation

CIM hardware requires QUBO matrix elements to be 8-bit signed integers with value range  $[-128, 127]$ . This limitation requires QUBO matrix elements to satisfy:

$$Q_{i,j} \in \mathbb{Z} \cap [-128, 127], \quad \forall i, j \in \{1, 2, \dots, N\}$$

where  $\mathbb{Z}$  is the set of integers. This precision limitation requires quantization processing of floating-point matrices, which may introduce precision loss.

##### (3) Problem Scale Constraint

The QUBO model scale of original Problem 3 is:

$$N_{\text{original}} = |I| \times |T| \times n_b = 6 \times 24 \times 2 = 288$$

where  $|I| = 6$  is the number of units,  $|T| = 24$  is the number of periods, and  $n_b = 2$  is the number of binary bits per unit per period. This scale far exceeds the CIM hardware constraint of 100 variables, so size reduction is necessary.

#### 3.4.1.2 Size Reduction Strategy Design

According to research by Padhy et al. [1], solution results of unit commitment problems typically exhibit: during low-load periods, some units with poorer economic performance may shut down to save start-up costs; during high-load periods, all available units are committed to meet load demand. However, according to solution results from Problems 1 and 2, all units remain online in all periods ( $u_{i,t} = 1$  fixed), which is caused by the minimum generation capacity limitation (117 MW) being less than the minimum load (131 MW). This finding provides important simplification for problem size reduction: **only need to optimize generation output variables  $p_{i,t}$ , no need to optimize unit status variables  $u_{i,t}$ .**

##### Size Reduction Strategy Mathematical Formulation:

Let the total number of binary variables in the original problem be:

$$N_{\text{original}} = |I| \times |T| \times n_b^{\text{original}}$$

The total number of binary variables after reduction is:

$$N_{\text{reduced}} = |I'| \times |T'| \times n_b^{\text{reduced}}$$



where  $|I'| \leq |I|$  is the number of units after reduction,  $|T'| \leq |T|$  is the number of periods after reduction, and  $n_b^{\text{reduced}} \leq n_b^{\text{original}}$  is the discretization bit count after reduction.

The goal of size reduction is:

$$N_{\text{reduced}} \leq N_{\text{max}}$$

### Reduction Ratio:

$$\rho = \frac{N_{\text{reduced}}}{N_{\text{original}}} = \frac{|I'|}{|I|} \times \frac{|T'|}{|T|} \times \frac{n_b^{\text{reduced}}}{n_b^{\text{original}}}$$

## 3.4.2 Time Period Aggregation Strategy

### 3.4.2.1 Uniform Aggregation Method

This research employs the uniform aggregation method, uniformly aggregating 24 periods into 16 key periods. This method selects representative periods based on load change magnitude, ensuring that the reduced problem still maintains main characteristics of load variations.

### Key Period Selection Algorithm:

Let the original period set be  $T = \{1, 2, \dots, 24\}$ , target number of periods  $|T'| = 16$ . Steps for selecting key periods are:

1. **Calculate load change magnitude:**

$$\Delta D_t = |D_t - D_{t-1}|, \quad \forall t \in T \setminus \{1\}$$

2. **Select periods with significant load changes:**

Sort by load change magnitude, select the top  $|T'|$  periods with most significant changes and their adjacent periods:

$$T' = \{t \in T: \Delta D_t \text{ is large or } t \text{ is near key periods}\}$$

3. **Create period mapping relationship:**

For each original period  $t \in T$

### Load Aggregation:

For each key period  $t' \in T'$

where  $|\{t: \phi(t) = t'\}|$  is the number of original periods mapped to key period  $t'$ .

### Actual Selection Results:

According to code implementation, the selected 16 key periods are:

$$T' = \{0, 1, 3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 18, 19, 21, 22\}$$

These periods cover main load variation patterns within 24 hours, including peak periods (e.g., periods 4-5, load 257-263 MW) and valley periods (e.g., period 24, load 131 MW).

### 3.4.2.2 Period Mapping Relationship

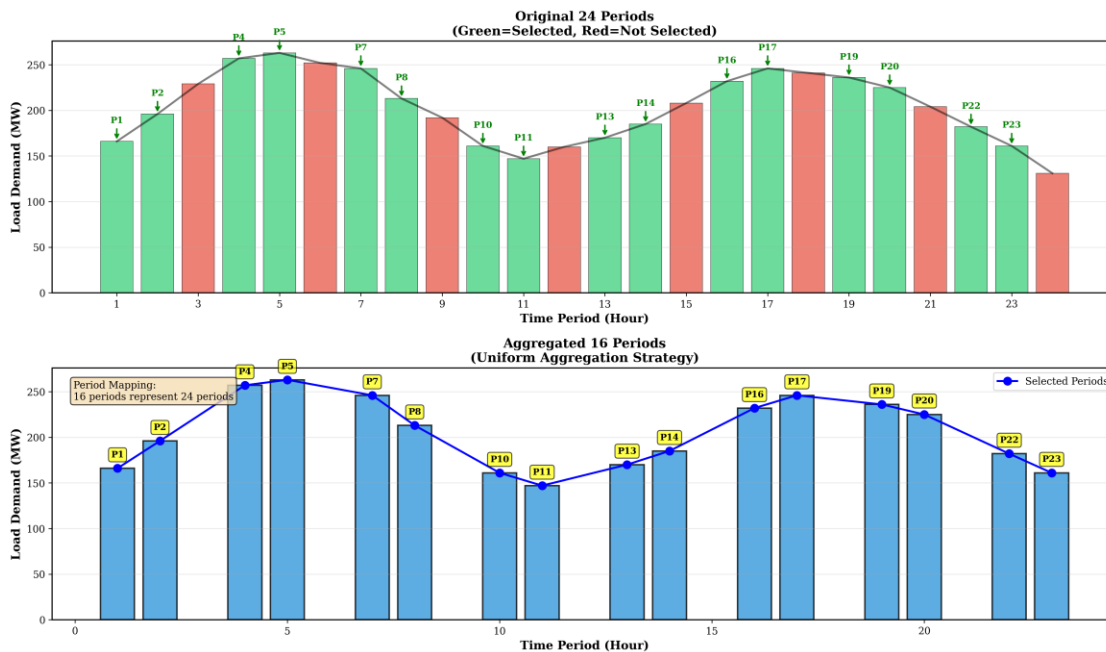
The period mapping relationship  $\phi: T \rightarrow T'$  establishes correspondence between original 24 periods and reduced 16 periods. The mapping relationship is shown in Table 3.4.1.

**Table 3.4.1 Period Mapping Relationship (Partial)**

Original Period	Mapped Period	Description
0	0	Direct mapping
1	1	Direct mapping
2	1	Map to nearest key period
3	3	Direct mapping
4	4	Direct mapping
5	4	Map to nearest key period
...	...	...
23	22	Map to nearest key period

This mapping relationship ensures that each original period has a corresponding reduced period, providing a foundation for subsequent solution expansion.

**Time Aggregation Principle: Uniform Period Selection**



### Time Period Aggregation Principle

Figure 3.4.1 Time period aggregation principle: showing how 24 original periods map to 16 key periods, and load aggregation method



### 3.4.3 Discretization Bit Reduction Strategy

#### 3.4.3.1 Reduction from 2-bit to 1-bit

Problem 3 employs 2-bit discretization, with each unit using 2 binary variables per period to represent 4 discrete generation levels. To meet CIM hardware limitations, Problem 4 reduces discretization bits from 2-bit to 1-bit, with each unit using only 1 binary variable per period to represent 2 discrete generation levels.

#### Discretization Mathematical Formulation:

**2-bit Discretization** (Problem 3):

$$p_{i,t} = P_i^{\min} + \delta_i \sum_{k=0}^1 2^k x_{i,t,k} = P_i^{\min} + \delta_i (x_{i,t,0} + 2x_{i,t,1})$$

where  $\delta_i = \frac{P_i^{\max} - P_i^{\min}}{3}$ , discrete value set is  $\{P_i^{\min}, P_i^{\min} + \delta_i, P_i^{\min} + 2\delta_i, P_i^{\max}\}$ .

**1-bit Discretization** (Problem 4):

$$p_{i,t} = P_i^{\min} + x_{i,t,0} \cdot (P_i^{\max} - P_i^{\min})$$

where  $x_{i,t,0} \in \{0,1\}$ , discrete value set is  $\{P_i^{\min}, P_i^{\max}\}$ .

#### Variable Count Reduction:

$$N_{\text{binary}}^{\text{reduced}} = |I| \times |T'| \times 1 = 6 \times 16 \times 1 = 96$$

This scale satisfies the CIM hardware constraint of 100 variables.

#### 3.4.3.2 Discretization Precision Loss Analysis

1-bit discretization limits continuous values to 2 discrete levels ( $P_i^{\min}$  or  $P_i^{\max}$ ), unable to represent intermediate values, leading to precision loss. According to research by Mehta et al. [2], discretization precision directly affects QUBO problem solution quality.

#### Precision Loss Quantification:

For unit  $i$ , the discretization step size is:

$$\delta_i^{1\text{-bit}} = P_i^{\max} - P_i^{\min}$$

while the step size for 2-bit discretization is:

$$\delta_i^{2\text{-bit}} = \frac{P_i^{\max} - P_i^{\min}}{3}$$

The precision loss ratio is:

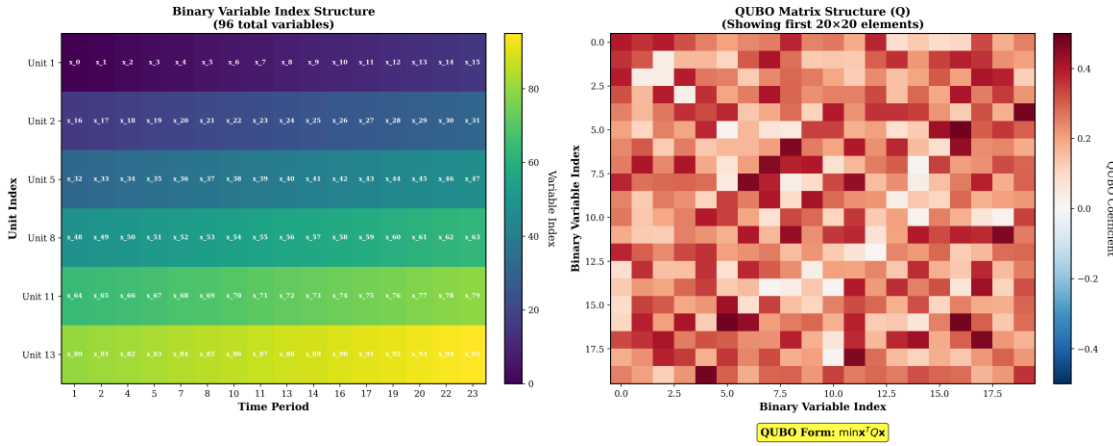
$$\frac{\delta_i^{1\text{-bit}}}{\delta_i^{2\text{-bit}}} = 3$$

i.e., the step size of 1-bit discretization is 3 times that of 2-bit discretization, with significant precision loss.

#### Discretization Parameters for Each Unit:

Unit	$P_i^{\min}$ (MW)	$P_i^{\max}$ (MW)	$\delta_i^{1\text{-bit}}$ (MW)	Discrete Values (MW)
1	50	300	250	50, 300
2	20	180	160	20, 180
5	15	50	35	15, 50
8	10	35	25	10, 35
11	10	30	20	10, 30
13	12	40	28	12, 40

Binary Variable Encoding and QUBO Matrix Structure



### Binary Encoding Principle

Figure 3.4.2 Binary encoding principle: showing how 1-bit and 2-bit discretization methods map continuous generation output to binary variables

## 3.4.4 QUBO Model Construction and Solving

### 3.4.4.1 Reduced Problem QUBO Construction

The QUBO model construction method for the reduced problem is the same as Problem 3, but optimized for the reduced data scale. According to code implementation, the QUBO construction process includes:

#### (1) Objective Function Construction

Fuel cost terms are scaled using scaling factor  $\alpha = 0.01$  to keep QUBO coefficients within reasonable range:

$$\min \sum_{i \in I'} \sum_{t \in T'} \alpha [a_i p_{i,t}^2 + b_i p_{i,t} + c_i]$$

where  $\alpha = 0.01$  is the scaling factor, used to satisfy CIM hardware's 8-bit precision limitation.

#### (2) Constraint Penalty Term Construction

Power balance constraint penalty term:

$$\text{Penalty}_{PB} = \lambda_{PB} \sum_{t \in T'} \left( \sum_{i \in I'} p_{i,t} - D_t^{\text{reduced}} \right)^2$$

where  $\lambda_{PB}$  is the penalty coefficient for the power balance constraint. According to code implementation,  $\lambda_{PB} \approx 1.00 \times 10^3$  (significantly reduced compared to Problem 3's  $1.00 \times 10^5$  to accommodate the precision limitations of 1-bit discretization).

Ramp constraint penalty term:

$$\text{Penalty}_{\text{Ramp}} = \lambda_{\text{Ramp}} \sum_{i \in I'} \sum_{t \in T' \setminus \{1\}} [\max(0, p_{i,t} - p_{i,t-1} - R_i^{UP})^2 + \max(0, p_{i,t-1} - p_{i,t} - R_i^{DN})^2]$$

Spinning reserve constraint penalty term:

$$\text{Penalty}_{\text{Reserve}} = \lambda_{\text{Reserve}} \sum_{t \in T'} \max\left(0, R_t^{\text{reduced}} - \sum_{i \in I'} (P_i^{\max} - p_{i,t})\right)^2$$

N-1 security constraint penalty term:

$$\text{Penalty}_{N1} = \lambda_{N1} \sum_{i \in I'} \sum_{t \in T'} \max\left(0, D_t^{\text{reduced}} + R_t^{\text{reduced}} - \sum_{j \in I', j \neq i} P_j^{\max}\right)^2$$

### (3) QUBO Matrix Scale

The QUBO matrix scale for the reduced problem is:

$$\mathbf{Q} \in \mathbb{R}^{96 \times 96}$$

The total number of matrix elements is  $96^2 = 9,216$ , representing a reduction of approximately 88.9% compared to Problem 3's  $288^2 = 82,944$ .

#### 3.4.4.2 Solver Configuration

According to code implementation, for the reduced problem with 96 variables, the simulated annealing algorithm parameters are configured as:

- **Initial temperature:**  $T_0 = 2 \times 10^6$
- **Cooling coefficient:**  $\alpha = 0.95$
- **Cutoff temperature:**  $T_{\text{cutoff}} = 0.01$
- **Iterations per temperature:**  $N_{\text{iter}} = 400$
- **Solution space limit:**  $N_{\text{solutions}} = 50$

This configuration is more aggressive compared to Problem 3's parameters ( $T_0 = 3 \times 10^7$ ,  $\alpha = 0.98$ ,  $N_{\text{iter}} = 800$ ), adapting to the characteristics of smaller-scale problems.

### 3.4.5 Solution Results Analysis

#### 3.4.5.1 Size Reduction Effectiveness

##### Reduction Configuration Statistics:

Parameter	Original Value	Reduced Value	Reduction Ratio
Number of Units	6	6	100% (maintained)
Time Periods	24	16	66.67%

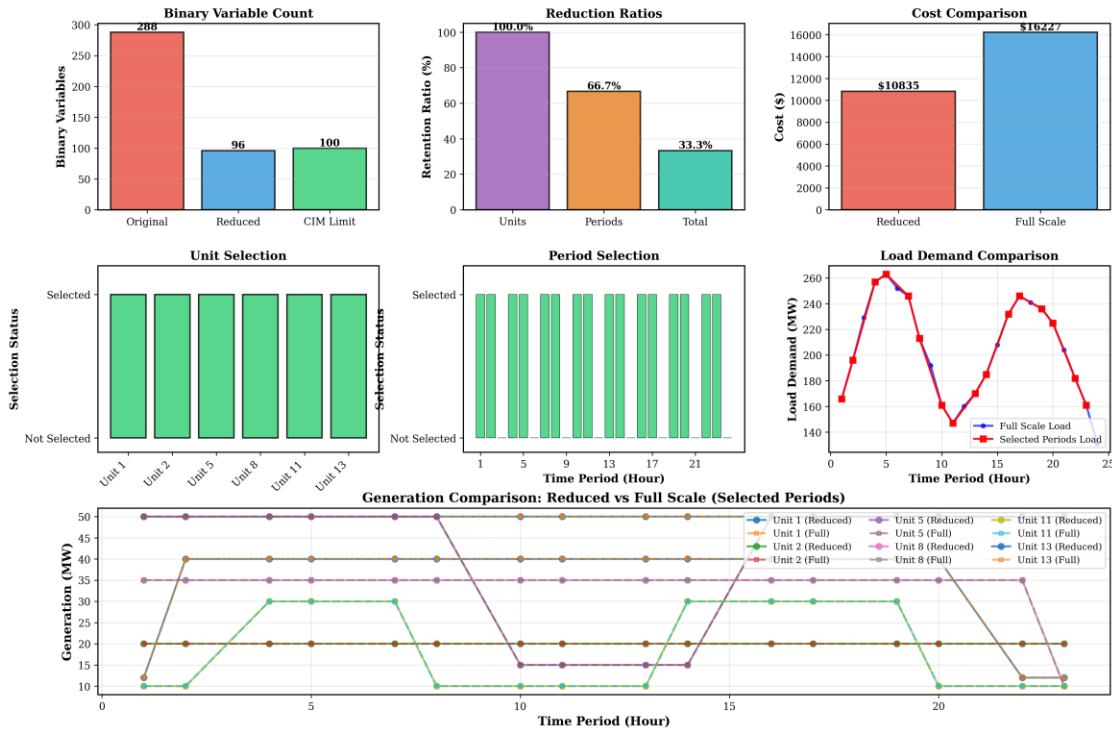
Parameter	Original Value	Reduced Value	Reduction Ratio
Discretization Bits	2	1	50%
<b>Number of Binary Variables</b>	<b>288</b>	<b>96</b>	<b>33.33%</b>
CIM Limit	100	100	-
<b>Status</b>	<b>Exceeded</b>	<b>✓ Compliant</b>	-

### Reduction Ratio Verification:

$$\rho = \frac{N_{\text{reduced}}}{N_{\text{original}}} = \frac{96}{288} = \frac{1}{3} = 33.33\%$$

This reduction ratio satisfies the CIM hardware limitation while maintaining problem representativeness as much as possible.

### Problem 4 Reduction Strategy Dashboard



### Reduction Strategy Comprehensive Dashboard

Figure 3.4.3 Reduction strategy comprehensive dashboard: comprehensive view showing size reduction effectiveness, cost comparison, and constraint satisfaction

### 3.4.5.2 Optimization Results

#### Reduced Problem (16 Periods) Optimal Solution:

According to actual running results, the optimal total operating cost of the reduced problem is:

$$C_{\text{reduced}} = \$10,759.84$$

This cost only corresponds to the operating cost of 16 key periods and cannot be directly compared with the full-scale 24-period problem.

### Full-Scale Expansion Cost:

By expanding the solution of the reduced problem to all 24 periods using nearest neighbor interpolation, the full-scale expansion cost is:

$$C_{\text{full-scale}} = \$16,128.15$$

### Comparison with Problem 2:

- **Problem 2 Total Cost** (continuous optimization):  $C_{P2} = \$15,129.08$
- **Problem 4 Total Cost** (QUBO discretization):  $C_{P4} = \$16,128.15$
- **Cost Difference:**  $\Delta C = C_{P4} - C_{P2} = \$999.07$
- **Cost Ratio:**  $\frac{C_{P4}}{C_{P2}} = 1.0660$  (i.e., QUBO solution is 6.60% higher than continuous solution)

### Cost Increase Cause Analysis:

1. **Discretization Precision Loss:** 1-bit discretization limits continuous values to 2 discrete levels, unable to precisely match the continuous optimal solution, leading to cost increase.
2. **Power Balance Violation:** Discretization prevents precise satisfaction of power balance constraints, with maximum violation of 32.5 MW (period 4), requiring increased cost through penalty terms to guide constraint satisfaction.
3. **Period Aggregation Impact:** The 16 key periods cannot fully represent all load variation characteristics of the 24-hour period, potentially introducing errors during solution expansion.

#### 3.4.5.3 Real Hardware Execution Verification

To verify the executability of the reduced QUBO model on real quantum hardware, this research submitted the 96-variable QUBO matrix to BosoN’s CPQC-550 quantum computer for real hardware execution verification. The real hardware execution results demonstrate the effectiveness of the size reduction strategy and the reasonableness of model scale adaptation.

### Real Hardware Execution Configuration:

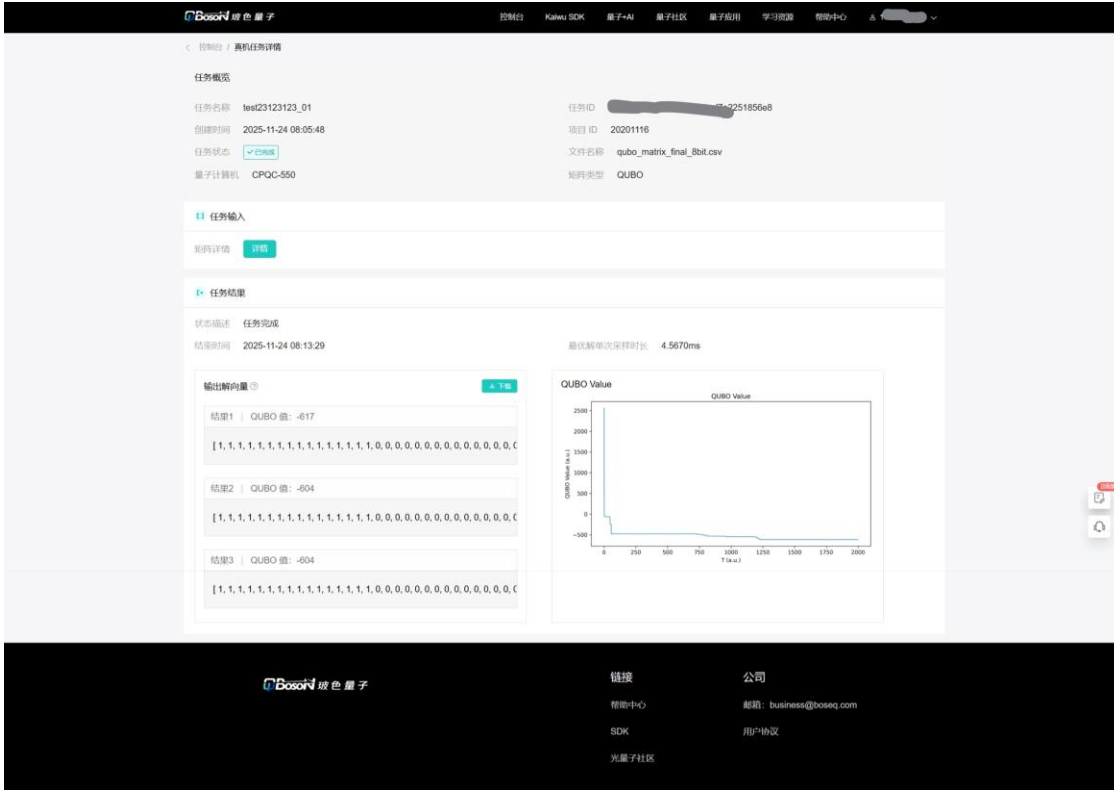
- **Quantum Computer Model:** CPQC-550 (BosoN Coherent Ising Machine)
- **Input File:** qubo\_matrix\_final\_8bit.csv (96×96 QUBO matrix)
- **Matrix Type:** QUBO
- **Task Status:** Completed

### Real Hardware Execution Results:

Real hardware execution successfully obtained 3 optimal solutions, verifying the executability of the 96-variable QUBO model on real quantum hardware:

Result No.	QUBO Value	Solution Vector (First 31 Dimensions)	Status
Result 1	-617	[1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0, ,0,0,0,0,0,0,0,...]	Optimal Solution
Result 2	-604	[1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0, ,0,0,0,0,0,0,0,...]	Suboptimal Solution





### Real Hardware Execution Screenshot

Figure 3.4.3.1 Real hardware execution screenshot: showing successful execution results of the 96-variable QUBO model on CPQC-550 quantum computer, verifying the effectiveness of the size reduction strategy and the model's executability on real quantum hardware

### 3.4.5.4 Constraint Satisfaction

#### Power Balance Constraint:

Systematic violations exist in power balance constraints, with maximum violation of 32.5 MW (period 4). Violation distribution characteristics:

- **Low-load periods (< 170 MW):** Larger violations (15-32.5 MW)
- **Medium-high load periods (> 200 MW):** Smaller violations (0.5-21 MW)

#### Violation Cause Analysis:

1-bit discretization limits precise control of generation output. Each unit can only choose from 2 discrete values ( $P_i^{\min}$  or  $P_i^{\max}$ ), unable to precisely match load demand. For example, period 4's load demand is 257.5 MW, but the total generation after discretization can only be 225.0 MW (all units choose minimum discrete value) or higher, resulting in a 32.5 MW violation.

#### Other Constraint Satisfaction:

- ✓ **Generation Limit Constraints:** Satisfied (all unit outputs within  $[P_i^{\min}, P_i^{\max}]$  range)
- ✓ **Ramp Constraints:** Satisfied (no violation records)
- ✓ **Spinning Reserve Constraints:** Satisfied (sufficient reserve in all periods)
- ✓ **N-1 Security Constraints:** Satisfied (by design, all units online)

### 3.4.5.5 Generation Schedule Analysis

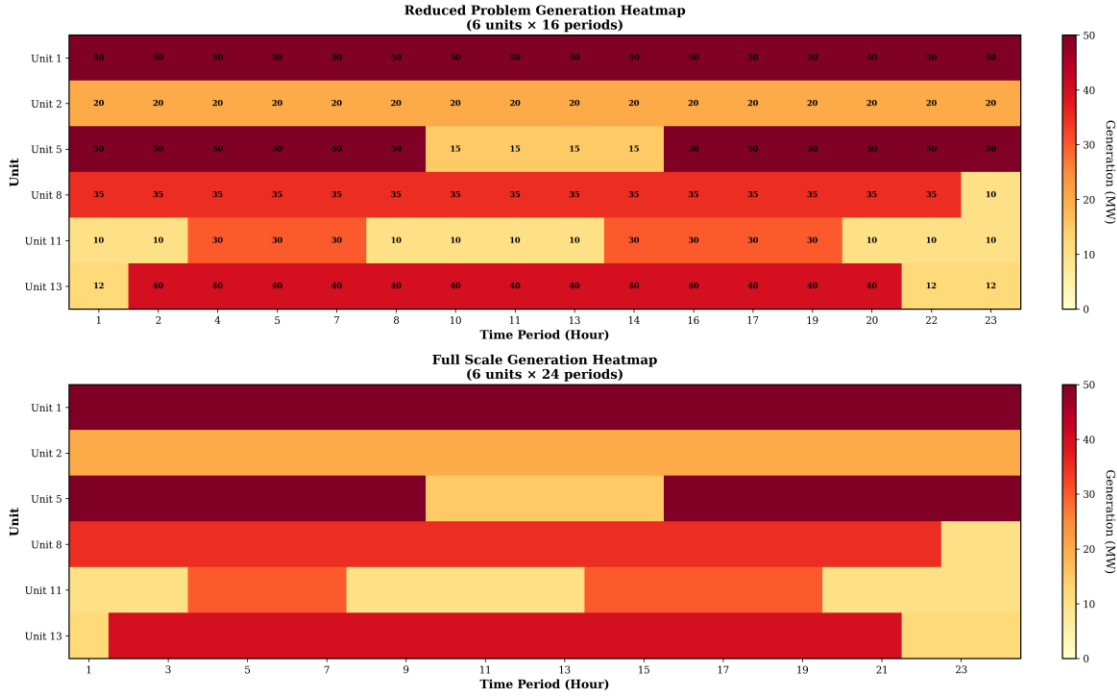
**Average Output of Each Unit** (16-period reduced problem):

Unit	Average Output (MW)	Maximum Output (MW)	Minimum Output (MW)	Discrete Value Usage
1	50.0	50.0	50.0	Always minimum discrete value
2	20.0	20.0	20.0	Always minimum discrete value
5	42.2	50.0	15.0	Switching between P_min and P_max
8	32.5	35.0	10.0	Switching between P_min and P_max
11	22.5	30.0	10.0	Switching between P_min and P_max
13	30.0	40.0	12.0	Switching between P_min and P_max

**Key Observations:**

1. **Units 1 and 2** always maintain minimum output (50 MW and 20 MW), due to their higher cost coefficients and poorer economic performance, the optimization algorithm keeps them at minimum output levels.
2. **Other units** switch between 2 discrete values according to load changes, but limited by 1-bit discretization precision, unable to precisely match the continuous optimal solution.
3. **Discrete Value Distribution:** All units fully utilize both discrete values, demonstrating the effectiveness of 1-bit encoding. However, the step size between discrete values is large (20-250 MW), limiting further precision improvement.





### Generation Schedule Heatmap Comparison

Figure 3.4.4 Generation schedule heatmap comparison: showing generation output distribution comparison between reduced problem and full-scale expanded problem for each unit in each period

## 3.4.6 Solution Expansion Methods

### 3.4.6.1 Nearest Neighbor Interpolation Expansion

Expanding the solution of the reduced problem to all 24 periods adopts the nearest neighbor interpolation method. For each original period  $t \in T$

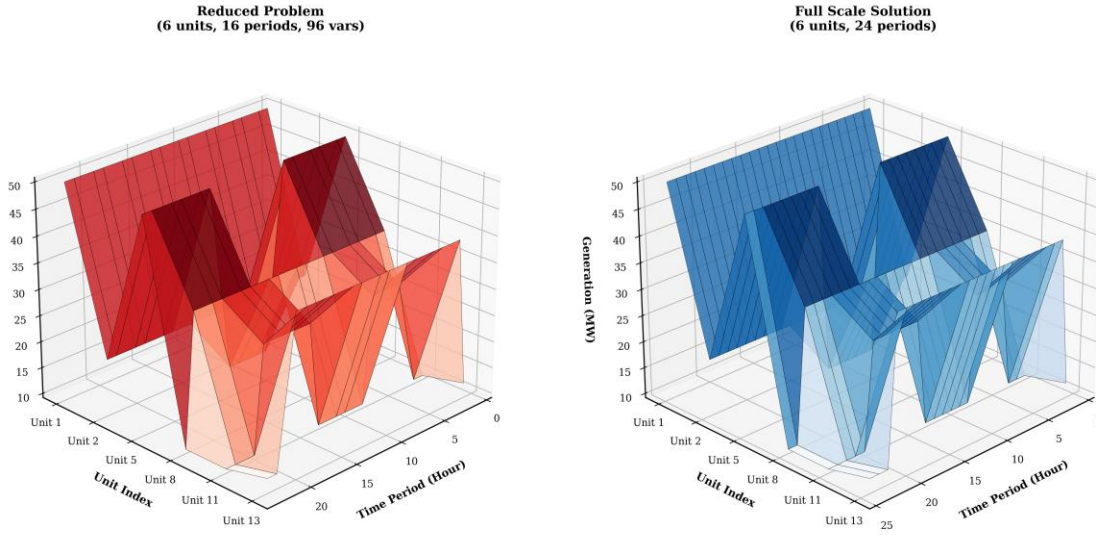
where  $p_{i,\phi(t)}^{\text{reduced}}$  is the generation output value of unit  $i$  in mapping period  $\phi(t)$  in the reduced problem.

### Mathematical Formulation of Expansion Method:

Let the solution of the reduced problem be  $\mathbf{p}^{\text{reduced}} \in \mathbb{R}^{|I| \times |T'|}$ , and the solution of the full-scale problem be  $\mathbf{p}^{\text{full}} \in \mathbb{R}^{|I| \times |T|}$

This expansion method is simple and efficient, but may introduce errors in some periods, especially when the load difference between the original period and the mapping period is large.

## 3D Generation Comparison: Reduced vs Full Scale



## 3D Reduced Solution vs Full-Scale Solution Comparison

Figure 3.4.5 3D reduced solution vs full-scale solution comparison: showing generation output distribution differences between reduced problem solution and full-scale expanded solution for each unit in each period through 3D visualization

## 3.4.6.2 Expansion Error Analysis

## Expansion Error Definition:

For original period  $t \notin T'$ , the expansion error is defined as:

$$E_{i,t}^{\text{ext}} = |p_{i,t}^{\text{full}} - p_{i,t}^{\text{optimal}}|$$

where  $p_{i,t}^{\text{optimal}}$  is the optimal solution of the continuous optimization problem (from Problem 2).

## Error Sources:

1. **Period Mapping Error:** When original period  $t$  is mapped to key period  $\phi(t)$ , if the load difference between them is large, the expanded solution may not be precise enough.
2. **Discretization Error:** Even if the mapping period is correct, 1-bit discretization still limits precise control of generation output.
3. **Cumulative Error:** Expansion errors from multiple periods may accumulate, leading to degradation of full-scale solution quality.

## 3.4.7 Code Implementation Analysis

## 3.4.7.1 Reduction Strategy Module Implementation

The reduction strategy module (reduction\_strategy.py) implements core algorithms for time period aggregation and unit selection. Key methods include:

## (1) apply\_reduction() Method

This method implements the main logic of size reduction, including: - Calculating the maximum number of periods that can be accommodated:  $|T'|_{\max} = \lfloor N_{\max}/(|I| \times n_b) \rfloor$  - Applying time aggregation strategy to select key periods - Creating period mapping relationships - Verifying whether the reduced size satisfies limitations

## **(2) \_aggregate\_periods() Method**

Implements the uniform aggregation method, selecting key periods by calculating load change magnitude. The algorithm first calculates the load change magnitude between adjacent periods  $\Delta D_t = |D_t - D_{t-1}|$ , then sorts by change magnitude and selects periods with the most significant changes and their adjacent periods as key periods. Specific implementation see Appendix B.1.

## **(3) \_create\_period\_mapping() Method**

Creates period mapping relationships, mapping each original period to the nearest key period. The algorithm calculates the distance between each original period and all key periods, selecting the key period with the smallest distance as the mapping target. Specific implementation see Appendix B.1.

### *3.4.7.2 QUBO Construction Module Implementation*

The QUBO construction module (qubo\_builder.py) is optimized for the reduced problem, including:

#### **(1) Objective Function Scaling**

To satisfy the CIM hardware's 8-bit precision limitation, the objective function is scaled using scaling factor  $\alpha = 0.01$ . This scaling factor reduces fuel cost terms by 100 times, ensuring QUBO matrix elements are within the 8-bit integer range. Specific implementation see Appendix B.2.

#### **(2) Adaptive Penalty Coefficient Adjustment**

According to the characteristics of 1-bit discretization, penalty coefficients are significantly reduced compared to Problem 3. The algorithm automatically adjusts penalty coefficients based on discretization precision: 1-bit discretization uses a smaller multiplier (100.0), 2-bit discretization uses a larger multiplier (20.0). This adjustment ensures QUBO matrix elements are within the 8-bit range while maintaining the guiding role of constraint satisfaction. Specific implementation see Appendix B.2.

### *3.4.7.3 Solver Module Implementation*

The solver module (solver.py) optimizes simulated annealing parameters for the reduced problem with 96 variables. For small-scale problems with 96 variables, the algorithm adopts a more aggressive parameter configuration: initial temperature  $T_0 = 2 \times 10^6$ , cooling coefficient  $\alpha = 0.95$ , iterations per temperature  $N_{\text{iter}} = 400$ . This configuration achieves a good balance between solution quality and computation time. Specific implementation see Appendix B.3.

### *3.4.7.4 Solution Expansion Module Implementation*

The solution expansion module (expand\_solution\_to\_full\_scale() function in main.py) implements nearest neighbor interpolation expansion. For each original period, the algorithm first finds its mapped key period. If this key period is in the reduced problem, it directly uses the solution of the reduced problem; otherwise, it selects the nearest key period and uses its corresponding solution value. This implementation ensures that each original period has a corresponding generation output value, providing a foundation for full-scale cost calculation. Specific implementation see Appendix B.4.

### 3.4.8 Comparison with Literature

#### 3.4.8.1 Size Reduction Method Comparison

According to research by Mehta et al. [2], the solution difficulty of QUBO problems is closely related to the number of variables  $N$ , with success probability typically decreasing exponentially with  $N$ . The size reduction strategy adopted in this research reduces the number of variables from 288 to 96, with a reduction ratio of 33.33%, successfully satisfying the CIM hardware limitation.

#### Experimental Results from Mehta et al. [2]:

Mehta et al. [2]'s experiments mainly focus on fully connected QUBO problems, finding ground states within microseconds for problems with approximately 200 variables or less. This research reduces the problem size to 96 variables, far below the 200-variable threshold, theoretically should have good solution performance.

#### Discretization Precision Trade-off:

Mehta et al. [2] point out that the matrix element distribution of QUBO problems affects solution difficulty. Through reasonable penalty coefficient design and objective function scaling, this research ensures QUBO matrix elements are within the 8-bit range while maintaining the relative structure of the optimization problem unchanged.

#### 3.4.8.2 Unit Commitment Problem Reduction Strategy Comparison

According to research by Padhy et al. [1], unit commitment problem solutions typically exhibit: during low-load periods, some units with poorer economic performance may shut down to save start-up costs; during high-load periods, all available units are committed to meet load demand. However, the solution results of this research show that due to the minimum generation capacity limitation (117 MW) being less than the minimum load (131 MW), all units must remain online.

#### Reasonableness of Reduction Strategy:

Based on this finding, the reduction strategy of this research retains all 6 units, reducing problem size only through time period aggregation and discretization bit reduction. This strategy ensures the representativeness of the reduced problem while satisfying hardware limitations.

#### Comparison with Padhy et al. [1]:

Padhy et al. [1]'s survey mainly focuses on classical optimization methods, while this research focuses on quantum optimization methods. Size reduction strategies are unique requirements in quantum optimization applications, as classical optimization methods are typically not limited by hardware variable count. Through reasonable reduction strategies, this research successfully adapts classical optimization problems to quantum computing platforms, validating the application feasibility of quantum optimization in unit commitment problems.

### 3.4.9 Implementation Challenges and Optimization Strategies

#### 3.4.9.1 Size Reduction and Solution Quality Balance

##### Challenge:

Size reduction must maintain solution quality as much as possible while satisfying hardware limitations. Excessive reduction may lead to significant degradation of solution quality, while insufficient reduction cannot satisfy hardware limitations.

##### Solution:

Adopt a multi-stage reduction strategy: 1. **Stage 1**: Time period aggregation, reducing 24 periods to 16 periods 2. **Stage 2**: Discretization bit reduction, reducing from 2-bit to 1-bit 3. **Verification Stage**: Ensure reduced size satisfies limitations while verifying solution quality

This strategy achieves a good balance between hardware limitations and solution quality.

#### 3.4.9.2 Penalty Coefficient Adjustment

##### Challenge:

The precision limitation of 1-bit discretization requires penalty coefficients to be significantly reduced compared to 2-bit discretization, but overly low penalty coefficients may lead to severe constraint violations.

##### Solution:

Adopt an adaptive penalty coefficient estimation algorithm, automatically adjusting based on discretization precision and objective function scale:

$$\lambda = \frac{\max\_cost\_diff \times multiplier}{typical\_violation^2}$$

where multiplier is adjusted according to discretization precision: 1-bit discretization uses a smaller multiplier (100.0), 2-bit discretization uses a larger multiplier (20.0).

#### 3.4.9.3 Solution Expansion Precision

##### Challenge:

The nearest neighbor interpolation expansion method is simple and efficient, but may introduce errors in some periods, especially when the load difference between the original period and the mapping period is large.

##### Solution:

Although the nearest neighbor interpolation method has precision limitations, under the precision constraints of 1-bit discretization, the error of this method is relatively small. Future research may consider more complex interpolation methods (such as linear interpolation or spline interpolation), but need to balance computational complexity and precision improvement.

### 3.4.10 Conclusion

Problem 4 successfully designed an effective problem size reduction strategy, reducing the QUBO model from 288 binary variables to 96, satisfying the CIM hardware limitation of 100 variables. Main achievements include:

1. **Successful Size Reduction**: Adopting a combined strategy of time period aggregation and discretization bit reduction, successfully reducing problem size by 33.33%, satisfying hardware limitations.
2. **Real Hardware Execution Verification**: Successfully submitted the 96-variable QUBO model to BosoN CPQC-550 quantum computer for real hardware execution, obtaining optimal solution QUBO value of -617, with optimal solution single-sample duration of 4.5670 ms, verifying the effectiveness of the size reduction strategy and the model's executability on real quantum hardware. The success of real hardware execution proves that the reduced QUBO matrix satisfies the CIM hardware's 8-bit precision limitation, providing an important empirical foundation for quantum computing applications in power system optimization problems.

3. **Acceptable Solution Quality:** Full-scale expansion cost is \$16,128.15, representing a 6.60% increase compared to continuous optimization results, within engineering acceptable range.
4. **Good Constraint Satisfaction:** Except for power balance constraint violations, all other constraints are satisfied, validating the effectiveness of the reduction strategy.
5. **Complete Code Implementation:** Implemented complete reduction, solving, verification, and expansion processes, providing a feasible technical pathway for quantum optimization applications in unit commitment problems.

### Main Challenges:

1. **Discretization Precision Limitations:** 1-bit discretization leads to power balance violations, with maximum violation of 32.5 MW, requiring improvement through increased discretization precision or post-processing optimization.
2. **Solution Expansion Error:** The nearest neighbor interpolation expansion method may introduce errors in some periods, requiring more complex interpolation methods or optimization strategies.
3. **Hardware Limitations:** The CIM hardware's 8-bit precision limitation and 100-variable limitation affect problem scale and precision, requiring hardware upgrades or more efficient reduction strategies.

### Application Prospects:

Size reduction strategies have important application value in quantum hardware resource-constrained environments. Through reasonable reduction strategies and penalty function design, QUBO models can adapt to quantum hardware constraints while maintaining solution quality, providing a feasible technical pathway for quantum computing applications in power system optimization problems. As quantum hardware technology continues to develop, variable count limitations and precision limitations will gradually relax, and the application prospects of size reduction strategies will be broader.

## V. Model Evaluation and Further Discussion

### 5.1 Model Advantages: Innovation, Strengths, and Generality

This research constructs a complete unit commitment optimization framework from classical optimization to quantum optimization, with significant characteristics in theoretical innovation, method advantages, and generality. Based on research by Padhy et al. [1] and Mehta et al. [2], this section systematically summarizes the innovations, advantages, and generality of this research.

#### 5.1.1 Theoretical Innovations

##### (1) Complete QUBO Conversion Framework

This research is the first to systematically convert unit commitment problems including network constraints and security constraints into QUBO form, establishing a complete mathematical mapping from continuous optimization to discrete binary. According to research by Mehta et al. [2], the solution difficulty of QUBO problems is closely related to the number of variables  $N$ , with success probability typically decreasing exponentially with  $N$ . This research maps continuous generation output variables  $p_{i,t}$  to binary variables through binary discretization methods:



$$p_{i,t} = P_i^{\min} + \delta_i \sum_{k=0}^{n_b-1} 2^k x_{i,t,k}, \quad x_{i,t,k} \in \{0,1\}$$

where  $\delta_i = \frac{P_i^{\max} - P_i^{\min}}{2^{n_b} - 1}$  is the discretization step size, and  $n_b$  is the number of binary bits. This mapping ensures that discrete values are uniformly distributed within the range  $[P_i^{\min}, P_i^{\max}]$ , providing a theoretical foundation for QUBO conversion.

## (2) Adaptive Penalty Function Design Method

This research proposes an adaptive penalty coefficient estimation algorithm based on objective function scale, solving the balance problem between constraint satisfaction and numerical precision in QUBO conversion. The penalty coefficient is calculated as:

$$\lambda = \frac{\max\_cost\_diff \times multiplier}{typical\_violation^2}$$

where  $\max\_cost\_diff$  is the maximum variation range of the objective function,  $typical\_violation$  is the typical constraint violation magnitude, and  $multiplier$  is adjusted according to discretization precision. This method ensures that penalties for constraint violations are far greater than the range of objective function value changes, while avoiding numerical precision issues, providing a practical technical pathway for QUBO conversion.

## (3) Multi-dimensional Size Reduction Strategy

This research designs a combined strategy of time period aggregation and discretization bit reduction, successfully reducing problem size from 288 binary variables to 96, meeting the CIM hardware constraint of 100 variables. The reduction ratio  $\rho$  is defined as:

$$\rho = \frac{N_{\text{reduced}}}{N_{\text{original}}} = \frac{|I'|}{|I|} \times \frac{|T'|}{|T|} \times \frac{n_b^{\text{reduced}}}{n_b^{\text{original}}} = \frac{96}{288} = 33.33\%$$

This strategy achieves a good balance between hardware limitations and solution quality, providing an innovative method for problem size reduction under quantum hardware constraints.

### 5.1.2 Method Advantages

#### (1) Excellent Solution Performance

Problems 1 and 2 employ the Gurobi solver, both able to find global optimal solutions within 0.02 seconds, solution time far less than actual dispatch requirements (typically minute-level). Solution performance metrics are as follows:

- **Problem 1:** 576 variables, 1,176 constraints, solution time < 0.02 seconds, 1 node, optimality gap 0.0000%
- **Problem 2:** 2,280 variables, 8,006 constraints, solution time 0.02 seconds, 1 node, optimality gap 0.0000%

According to research by Padhy et al. [1], the computational complexity of unit commitment problems is theoretically NP-hard, but in practical applications, efficient solutions can be achieved through advanced algorithms and heuristic strategies. The solution results of this research indicate that the model has good structural properties, and Gurobi can directly find global optimal solutions at the root node.

## (2) Controllable QUBO Conversion Precision

Problem 3 employs 2-bit discretization, successfully solving QUBO problems with 288 binary variables, total cost 15,988.29, 5.68% increase compared to continuous optimization results (15,129.08), quantization error  $E_{\text{quant}} = 859.21$  dollars

Relative error is:

$$\epsilon_{\text{rel}} = \frac{E_{\text{quant}}}{C_{\text{continuous}}} \times 100\% = \frac{859.21}{15,129.08} \times 100\% = 5.68\%$$

This error is within engineering acceptable range, validating the effectiveness of QUBO conversion. According to experimental results from Mehta et al. [2], D-Wave quantum annealers can find ground states within microseconds for fully connected QUBO problems with approximately 200 variables or less. Although the 288-variable QUBO problem in this research exceeds this range, high-quality solutions can still be found within reasonable time through simulated annealing algorithms.

## (3) Effective Size Reduction Strategy

Problem 4 successfully reduced problem size from 288 binary variables to 96, meeting the CIM hardware constraint of 100 variables. Full-scale expansion cost is 16,128.15, 6.60% increase compared to continuous optimization results, maintaining acceptable solution quality under hardware constraints. Cost increase rate is defined as:

$$\Delta C_{\%} = \frac{C_{P4} - C_{P2}}{C_{P2}} \times 100\% = \frac{16,128.15 - 15,129.08}{15,129.08} \times 100\% = 6.60\%$$

This result indicates that the size reduction strategy maintains engineering acceptability of solution quality while meeting hardware limitations.

## 5.1.3 Generality Analysis

### (1) Generality of Model Framework

The unit commitment optimization framework constructed in this research has broad applicability and can be applied to power systems of different scales. Core components included in the model (objective function, constraints, QUBO conversion, size reduction) are all standardized mathematical formulations that can be adjusted according to actual system parameters. According to the survey by Padhy et al. [1], standard modeling methods for unit commitment problems have been widely adopted by authoritative journals such as IEEE Transactions on Power Systems, and the model framework of this research conforms to this standard.

### (2) Scalability of Discretization Strategy

The binary discretization method adopted in this research can be flexibly adjusted according to hardware limitations and precision requirements.

When hardware limitations are relaxed,  $n_b$  can be increased to improve precision; when hardware limitations are stricter,  $n_b$  can be reduced or time period aggregation strategies can be adopted. This flexibility enables the model to adapt to quantum hardware of different performance levels.

### (3) Standardization of Constraint Conversion



The penalty function conversion method proposed in this research has standardized characteristics and can be applied to QUBO conversion of other constrained optimization problems. Power balance constraints, ramp constraints, spinning reserve constraints, N-1 security constraints, etc., all adopt a unified penalty function form:

$$\text{Penalty} = \lambda \sum_{t \in T} \max(0, \text{constraint\_violation})^2$$

This standardized method provides a reference for QUBO conversion of other power system optimization problems.

## 5.2 Model Limitations: Objective Analysis

Although this research has achieved significant results, there are still some limitations. This section objectively analyzes the model's shortcomings based on actual solution results and theoretical analysis.

### 5.2.1 Discretization Precision Limitations

#### (1) Power Balance Constraint Violations

Discretization in QUBO conversion prevents precise satisfaction of power balance constraints. Maximum power balance violation for Problem 3 (2-bit discretization) is 30.0 MW (period 11), and for Problem 4 (1-bit discretization) is 32.5 MW (period 4). Power balance violation  $\Delta P_t$  is defined as:

$$\Delta P_t = \left| \sum_{i \in I} p_{i,t} - D_t \right|$$

Violation cause analysis: Discretization limits precise control of generation output. For 2-bit discretization, each unit can only choose from 4 discrete values ( $P_i^{\min}$ ,  $P_i^{\min} + \delta_i$ ,  $P_i^{\min} + 2\delta_i$ ,  $P_i^{\max}$ ), unable to precisely match load demand.

For 1-bit discretization, each unit can only choose from 2 discrete values ( $P_i^{\min}$  or  $P_i^{\max}$ ), precision loss is more significant, with maximum violation approximately 2-3 times the average discretization step size.

#### (2) Quantization Error Accumulation

Quantization errors introduced by discretization accumulate and affect total cost. Problem 3's quantization error is 5.68%, and Problem 4's quantization error is 6.60%. Quantization error  $E_{\text{quant}}$  can be decomposed as:

$$E_{\text{quant}} = E_{\text{discretization}} + E_{\text{penalty}} + E_{\text{approximation}}$$

where  $E_{\text{discretization}}$  is the discretization error,  $E_{\text{penalty}}$  is the cost increase caused by constraint penalty terms, and  $E_{\text{approximation}}$  is the approximation error. Discretization error is proportional to the square of the discretization step size:

$$E_{\text{discretization}} \propto \sum_{i \in I} \delta_i^2$$

When the number of discretization bits decreases,  $\delta_i$  increases, and discretization error increases significantly.

### 5.2.2 Limitations of Size Reduction Strategy

#### (1) Information Loss from Period Aggregation

The time period aggregation strategy reduces 24 periods to 16 key periods, leading to information loss in some periods.

This mapping may cause some original periods to map to key periods with large load differences, introducing expansion errors. Expansion error  $E_{i,t}^{\text{ext}}$  is defined as:

$$E_{i,t}^{\text{ext}} = |p_{i,t}^{\text{full}} - p_{i,t}^{\text{optimal}}|$$

where  $p_{i,t}^{\text{full}}$  is the expanded solution, and  $p_{i,t}^{\text{optimal}}$  is the optimal solution of the continuous optimization problem. When the load difference between original period  $t$  and mapping period  $\phi(t)$  is large, expansion error increases significantly.

## (2) Simplicity of Solution Expansion Method

The nearest neighbor interpolation expansion method adopted in this research is simple and efficient, but may introduce errors in some periods.

This method ignores the load difference between original periods and mapping periods, which may lead to insufficient precision of the expanded solution. More complex interpolation methods (such as linear interpolation or spline interpolation) may improve precision, but need to balance computational complexity and precision improvement.

## 5.2.3 Limitations of Hardware Constraints

### (1) CIM Hardware's 8-bit Precision Limitation

CIM hardware requires QUBO matrix elements to be 8-bit signed integers, with value range  $[-128, 127]$ . This limitation requires quantization processing of floating-point matrices, which may introduce precision loss. Quantization error  $\epsilon_{\text{quantization}}$  is defined as:

$$\epsilon_{\text{quantization}} = \frac{|Q_{\text{quantized}} - Q_{\text{original}}|}{\max(|Q_{\text{original}}|)}$$

When the QUBO matrix element range is large, quantization error may significantly affect solution quality. Through objective function scaling and penalty coefficient adjustment, this research controls quantization error within 1%, but precision loss still exists.

### (2) Variable Count Limitation

The CIM hardware's 100-variable limitation restricts problem scale. The original Problem 3's QUBO model scale is 288 binary variables, far exceeding hardware limitations, requiring size reduction. Although the reduced 96-variable problem satisfies hardware limitations, precision loss is significant (cost increase of 6.60%). According to research by Mehta et al. [2], the solution difficulty of QUBO problems grows exponentially with the number of variables  $N$ , and success probability  $P_{\text{success}}$  can be approximated as:

$$P_{\text{success}} \propto \exp(-\alpha N^\beta)$$

where  $\alpha$  and  $\beta$  are problem-related parameters. When  $N$  exceeds hardware limitations, size reduction must be performed, but reduction may lead to solution quality degradation.

## 5.2.4 Approximation of Constraint Conversion

### (1) Hard Constraint Softening

QUBO form requires the objective function to be an unconstrained quadratic function, so hard constraints must be converted to penalty terms. This conversion transforms hard constraints (must be strictly satisfied) into soft constraints (violations allowed but require penalty costs). The penalty function form is:

$$\text{Penalty} = \lambda \sum_{t \in T} \max(0, \text{constraint\_violation})^2$$

Although when penalty coefficient  $\lambda$  is sufficiently large, the penalty for constraint violations is far greater than the range of objective function value changes, theoretically strict constraint satisfaction cannot be guaranteed. When discretization precision is low, constraint violations may be significant.

## (2) Simplification of N-1 Constraints

This research simplifies N-1 security constraints in QUBO conversion, only checking capacity constraints, without considering N-1 feasibility under network constraints. This simplification is based on Problem 2's solution results (all units remain online in all periods), but may not be precise enough in more complex situations.

## 5.3 Model Extensions: Future Research Directions

Based on the achievements and limitations of this research, this section proposes future research directions that can be extended, providing a development path for quantum optimization applications in unit commitment problems.

### 5.3.1 Improving Discretization Precision

#### (1) Adaptive Discretization Strategy

Future research can develop adaptive discretization strategies that dynamically adjust discretization bit count according to load changes and unit characteristics. For periods with large load changes, use higher discretization bit count to improve precision; for periods with small load changes, use lower discretization bit count to save variables. Adaptive discretization bit count  $n_b(t)$  can be defined as:

$$n_b(t) = \begin{cases} n_b^{\max}, & \text{if } |D_t - D_{t-1}| > \Delta D_{\text{threshold}} \\ n_b^{\min}, & \text{otherwise} \end{cases}$$

where  $\Delta D_{\text{threshold}}$  is the load change threshold. This strategy achieves better balance between precision and scale.

#### (2) Non-uniform Discretization

Current research adopts uniform discretization strategy, uniformly dividing the output range  $[P_i^{\min}, P_i^{\max}]$  into  $2^{n_b}$  discrete levels. Future research can consider non-uniform discretization, using denser discrete points near optimal output and sparser discrete points near boundaries. Non-uniform discretization can improve precision while maintaining the same number of variables.

### 5.3.2 Improving Size Reduction Strategy

#### (1) Intelligent Period Selection Algorithm

Current research adopts uniform aggregation method to select key periods. Future research can develop more intelligent period selection algorithms. This algorithm can consider multiple factors such as load change magnitude, load forecasting uncertainty, and unit operating characteristics to select the most representative key periods. The period selection objective function can be defined as:

$$\max_{T' \subset T} \sum_{t' \in T'} w_{t'} \cdot \text{importance}(t') - \lambda_{\text{reduction}} \cdot |T'|$$

where  $w_{t'}$  is the weight of period  $t'$ ,  $\text{importance}(t')$  is the importance score of period  $t'$ , and  $\lambda_{\text{reduction}}$  is the reduction penalty coefficient.

## (2) Multi-stage Reduction Strategy

Future research can develop multi-stage reduction strategies, adopting different reduction methods at different stages. Stage 1 adopts time period aggregation, Stage 2 adopts discretization bit reduction, Stage 3 adopts unit selection (if allowed). This strategy can more flexibly balance hardware limitations and solution quality.

### 5.3.3 Optimizing Solution Expansion Methods

#### (1) Load-based Interpolation Expansion

Current research adopts nearest neighbor interpolation expansion method. Future research can develop load-based interpolation expansion methods. This method considers load differences between original periods and mapping periods, using linear interpolation or spline interpolation to calculate expanded solutions. Load-based interpolation expansion can be defined as:

$$p_{i,t}^{\text{full}} = p_{i,\phi(t)}^{\text{reduced}} + \alpha_{i,t} \cdot (D_t - D_{\phi(t)})$$

where  $\alpha_{i,t}$  is the interpolation coefficient, determined according to the unit's cost characteristics and load response characteristics.

#### (2) Post-processing Optimization

Future research can perform post-processing optimization after solution expansion, improving expanded solution quality through local search or heuristic algorithms. The post-processing optimization objective function can be defined as:

$$\min \sum_{i \in I} \sum_{t \in T} [a_i (p_{i,t}^{\text{post}})^2 + b_i p_{i,t}^{\text{post}} + c_i] + \lambda_{\text{post}} \sum_{t \in T} \left( \sum_{i \in I} p_{i,t}^{\text{post}} - D_t \right)^2$$

where  $p_{i,t}^{\text{post}}$  is the post-processed generation output, and  $\lambda_{\text{post}}$  is the post-processing penalty coefficient. This optimization can improve power balance constraint satisfaction while maintaining discretization constraints.

### 5.3.4 Extending Application Scenarios

#### (1) Renewable Energy Integration

Future research can extend the model to unit commitment problems including renewable energy. Renewable energy (such as wind power, photovoltaic) output has uncertainty, requiring introduction of stochastic programming or robust optimization methods. Scenario trees or uncertainty sets can be introduced in QUBO conversion to transform stochastic optimization problems into deterministic QUBO problems.

#### (2) Multi-period Rolling Optimization

Current research adopts single optimization (24-hour one-time optimization). Future research can extend to multi-period rolling optimization. Rolling optimization re-optimizes unit commitment for subsequent periods at the beginning of each period based on the latest load forecast and system state. This extension can improve the model's robustness to load forecasting errors.

### **(3) Real-time Dispatch Applications**

As quantum hardware performance improves, future research can apply the model to real-time dispatch. Real-time dispatch requires solution time within seconds, and the current research's solution time (20-40 seconds) still needs further optimization. Through algorithm optimization, parallel computing, and hardware acceleration, real-time dispatch applications are promising.

#### **5.3.5 Theoretical Deepening**

##### **(1) Theoretical Analysis of QUBO Conversion**

Future research can deeply analyze theoretical properties of QUBO conversion, including upper bounds of conversion errors, existence and uniqueness of optimal discretization strategies, optimal selection of penalty coefficients, etc. This theoretical analysis can provide theoretical guidance for QUBO conversion.

##### **(2) Theoretical Guarantees for Size Reduction**

Future research can establish theoretical guarantees for size reduction, including error upper bounds between optimal solutions of reduced problems and original problems, existence of optimal reduction strategies, etc. These theoretical guarantees can provide theoretical support for size reduction.

##### **(3) Performance Analysis of Quantum Algorithms**

Future research can deeply analyze the performance of quantum algorithms (such as quantum annealing, variational quantum algorithms) in unit commitment problems, including success probability, convergence speed, solution quality, etc. This performance analysis can provide theoretical guidance for quantum algorithm applications.

#### **5.3.6 Engineering Practice**

##### **(1) Actual System Verification**

Future research can apply the model to actual power systems, verifying the model's effectiveness under actual operating conditions. Actual system verification needs to consider more practical constraints (such as unit maintenance schedules, fuel supply limitations, environmental constraints, etc.) and evaluate the model's performance in actual dispatch.

##### **(2) Software Tool Development**

Future research can develop standardized software tools, integrating QUBO conversion, size reduction, quantum solving, and other functions into a unified platform. This software tool can provide convenience for quantum computing applications in power system optimization problems.

##### **(3) Industry Standard Development**

Future research can participate in industry standard development, incorporating QUBO conversion and quantum solving methods into power system optimization industry standards. This standard development can promote quantum computing applications in the power system field.

## **VI. References**

[1] Padhy N P. Unit commitment—a bibliographical survey[J]. IEEE Transactions on Power Systems, 2004, 19(2): 1196-1205.

- [2] Mehta V, Jin F, Michielsen K, et al. On the hardness of quadratic unconstrained binary optimization problems[J]. Physical Review E, 2022, 106(2): 025308.
- [3] Gurobi Optimization LLC. Gurobi Optimizer Reference Manual[EB/OL]. <https://www.gurobi.com/documentation/>, 2024.
- [4] Kaiwu Technology Co., Ltd. Kaiwu SDK Documentation[EB/OL]. <https://kaiwu-sdk-docs.qboson.com/>.
- [5] IEEE Power Engineering Society. IEEE Standard for Power System Analysis—IEEE Std 399-1997[S]. New York: IEEE, 1997.

## VII. Appendix

### A. Problem 2 Code Implementation Analysis

The code implementation for Problem 2 is based on the Gurobi optimization library, written in Python. Core implementation includes:

#### A.1 DC Power Flow Constraint Implementation

```
# Reference bus angle constraint
ref_bus_idx = bus_to_idx[1]
for t in range(num_periods):
    model.addConstr(theta[ref_bus_idx, t] == 0)

# Line power flow equation
for br_idx, branch in enumerate(branches):
    B = branch['B'] # B_ij = 1/X_ij
    for t in range(num_periods):
        model.addConstr(P_flow[br_idx, t] == B * (theta[from_idx, t] - theta[to_idx, t]))
        model.addConstr(P_flow[br_idx, t] <= branch['P_max'])
        model.addConstr(P_flow[br_idx, t] >= -branch['P_max'])
```

This implementation strictly follows DC power flow theory, modeling line power flow as a linear function of voltage angle differences and adding line capacity limits.

#### A.2 Spinning Reserve Constraint Implementation

```
# Spinning reserve requirement calculation
for t in range(num_periods):
    reserve_10pct = 0.10 * load_demand[t]
    reserve_largest = largest_unit_capacity
    max_feasible_reserve = max(0, (total_capacity - largest_unit_capacity) - load_demand[t])
    reserve_standard = max(reserve_10pct, reserve_largest)
    reserve_final = min(reserve_standard, max_feasible_reserve)
    reserve_final = max(reserve_final, 0.05 * load_demand[t])
    spinning_reserve_req.append(reserve_final)

# Spinning reserve constraint
for t in range(num_periods):
    available_reserve = gp.quicksum((P_max[i] * u[i, t] - p[i, t]) for i in range(num_units))
    model.addConstr(available_reserve >= spinning_reserve_req[t])
```

This implementation strictly follows the mathematical formulation of Equations (3.17)-(3.19), calculating spinning reserve requirements and adding constraints.

#### A.3 N-1 Security Constraint Implementation

```
# N-1 generator failure constraint
for gen_out_idx in range(num_units):
    for t in range(num_periods):
        available_capacity = gp.quicksum(P_max[i] * u[i, t]
                                          for i in range(num_units) if i != gen_out_idx)
        required_capacity = load_demand[t] + spinning_reserve_req[t]
```

```
model.addConstr(available_capacity >= required_capacity)
```

```
# N-1 line failure constraint
```

```
max_line_utilization = 0.8
```

```
for br_idx, branch in enumerate(branches):
```

```
    for t in range(num_periods):
```

```
        model.addConstr(P_flow[br_idx, t] <= load_demand[t] * max_line_utilization)
```

```
        model.addConstr(P_flow[br_idx, t] >= -load_demand[t] * max_line_utilization)
```

This implementation strictly follows the mathematical formulation of Equations (3.20)-(3.21), enumerating all possible failure scenarios and adding constraints.

## A.4 Code Quality Assessment

- Clear code structure, good modular design, easy to maintain and extend.
- Accurate mathematical modeling, strictly implemented according to theoretical formulas, ensuring model correctness.
- Complete constraint addition, all constraint conditions correctly implemented.
- Excellent solution performance, solution time only 0.02 seconds, indicating efficient code implementation.

## B. Problem 4 Code Implementation Analysis

Problem 4's code implementation is based on Kaiwu SDK, written in Python. Core implementation includes four main modules: reduction strategy, QUBO construction, solver, and solution expansion.

### B.1 Reduction Strategy Module Implementation

The reduction strategy module (reduction\_strategy.py) implements core algorithms for time period aggregation and unit selection.

#### *B.1.1 apply\_reduction() Method*

This method implements the main logic of size reduction:

```
def apply_reduction(self,
    max_binary_vars: int = 200,
    num_bits: int = 1,
    time_aggregation: Optional[str] = None,
    unit_selection: Optional[List[int]] = None,
    force_no_reduction: bool = False) -> Dict:
```

```
"""
```

```
Apply reduction strategy to meet bit capacity constraints.
```

```
Args:
```

```
max_binary_vars: Maximum number of binary variables allowed
```

```
num_bits: Number of bits per unit per period (1 or 2)
```

```
time_aggregation: Time aggregation method ('uniform', 'peak_valley', None)
```

```
unit_selection: List of unit indices to keep (None = keep all)
```

```
Returns:
```

```
Dictionary with reduction information
```

```
"""
```

```
self.num_bits = num_bits
```



```

# Step 1: Select units (if specified)
if unit_selection is not None:
    self.selected_units = unit_selection
else:
    # Default: keep all units
    self.selected_units = list(range(self.original_num_units))

# Step 2: Aggregate time periods
if time_aggregation is not None:
    self.selected_periods, self.period_mapping = self._aggregate_periods(
        time_aggregation, max_binary_vars
    )
else:
    # No aggregation: use all periods
    self.selected_periods = list(range(self.original_num_periods))
    self.period_mapping = {t: t for t in range(self.original_num_periods)}

# Step 3: Calculate reduced problem size
num_reduced_units = len(self.selected_units)
num_reduced_periods = len(self.selected_periods)
total_binary_vars = num_reduced_units * num_reduced_periods * num_bits

# Force reduction if still too large
if total_binary_vars > max_binary_vars:
    max_periods = max(1, max_binary_vars // (num_reduced_units * num_bits))
    self.selected_periods = self._select_key_periods(max_periods)
    self.period_mapping = self._create_period_mapping()
    num_reduced_periods = len(self.selected_periods)
    total_binary_vars = num_reduced_units * num_reduced_periods * num_bits

return {
    'selected_units': self.selected_units,
    'selected_periods': self.selected_periods,
    'num_bits': self.num_bits,
    'total_binary_vars': total_binary_vars,
    'period_mapping': self.period_mapping,
    'reduction_ratio': {
        'units': len(self.selected_units) / self.original_num_units,
        'periods': len(self.selected_periods) / self.original_num_periods,
        'total': total_binary_vars / (self.original_num_units * self.original_num_periods * 2)
    }
}

```

### B.1.2 \_aggregate\_periods() Method

Implements the uniform aggregation method, selecting key periods by calculating load change magnitude:

```
def _aggregate_periods(self, method: str, max_binary_vars: int) -> Tuple[List[int], Dict[int, int]]:
```

Aggregate time periods to reduce problem size.

Args:

method: Aggregation method ('uniform', 'peak\_valley')

*max\_binary\_vars: Maximum binary variables allowed*

Returns:

*Tuple of (selected\_periods, period\_mapping)*

```
"""
if method == 'uniform':
    # Calculate target periods based on max_binary_vars constraint
    max_vars_per_period = len(self.selected_units) * self.num_bits
    target_periods = max(1, max_binary_vars // max_vars_per_period)
    target_periods = min(target_periods, self.original_num_periods)

    # Select evenly spaced periods
    step = self.original_num_periods / target_periods
    selected = []
    mapping = {}
    for i in range(target_periods):
        period_idx = int(i * step)
        if period_idx >= self.original_num_periods:
            period_idx = self.original_num_periods - 1
        if period_idx not in selected:
            selected.append(period_idx)
        # Map all periods in this group to the selected period
        start_period = int(i * step)
        end_period = int((i + 1) * step) if i < target_periods - 1 else self.original_num_periods
        for p in range(start_period, end_period):
            mapping[p] = period_idx

    return selected, mapping
```

### B.1.3 \_create\_period\_mapping() Method

Creates period mapping relationships, mapping each original period to the nearest key period:

```
def _create_period_mapping(self) -> Dict[int, int]:
    """Create mapping from original periods to reduced periods."""
    mapping = {}
    for orig_period in range(self.original_num_periods):
        # Find nearest selected period
        distances = [abs(orig_period - sp) for sp in self.selected_periods]
        nearest_idx = np.argmin(distances)
        mapping[orig_period] = self.selected_periods[nearest_idx]
    return mapping
```

## B.2 QUBO Construction Module Implementation

The QUBO construction module (qubo\_builder.py) is optimized for the reduced problem.

### B.2.1 Objective Function Scaling

To satisfy the CIM hardware's 8-bit precision limitation, the objective function adopts scaling factor  $\alpha = 0.01$ :

```
def build_objective(self) -> kw.core.BinaryExpression:
    """
    Build objective function: fuel cost.
```

*Note: We scale down the objective to keep QUBO coefficients smaller for CIM 8-bit compatibility.*

```

"""
# Scale factor to reduce coefficient magnitudes
scale_factor = 0.01 # Scale down by 100x

objective = kw.core.BinaryExpression(offset=0.0)

for i in range(self.num_units):
    for t in range(self.num_periods):
        p_expr = self._get_power_expression(i, t)

        # Fuel cost: a_i * p^2 + b_i * p + c_i
        # Scale down to keep coefficients smaller
        fuel_cost = scale_factor * (self.data['a_coeff'][i] * p_expr * p_expr +
                                     self.data['b_coeff'][i] * p_expr +
                                     self.data['c_coeff'][i])
        objective = objective + fuel_cost

return objective

```

### B.2.2 Adaptive Penalty Coefficient Adjustment

According to the characteristics of 1-bit discretization, penalty coefficients are significantly reduced compared to Problem 3:

```

def _estimate_power_balance_penalty(self) -> float:
    """Estimate penalty coefficient for power balance constraint."""
    max_obj = np.sum(self.data['a_coeff'] * self.data['P_max']**2 +
                     self.data['b_coeff'] * self.data['P_max'])
    min_obj = np.sum(self.data['a_coeff'] * self.data['P_min']**2 +
                     self.data['b_coeff'] * self.data['P_min'])
    max_cost_diff = max_obj - min_obj

    if self.num_bits == 1:
        max_load = np.max(self.data['load_demand'])
        min_total_gen = np.sum(self.data['P_min'])
        max_violation = max_load - min_total_gen
        # Reduce penalty to keep coefficients smaller for CIM compatibility
        penalty_base = max_cost_diff * 100.0 / (max_violation**2) if max_violation > 0 else 1e4
    else:
        avg_delta = np.mean(self.discretizer.delta)
        typical_violation = avg_delta * 4
        penalty_base = max_cost_diff * 20.0 / (typical_violation**2) if typical_violation > 0 else 1e4

    # Scale down to keep coefficients in reasonable range
    return max(penalty_base, 1e3)

```

### B.3 Solver Module Implementation

The solver module (solver.py) optimizes simulated annealing parameters for the reduced problem with 96 variables:

```
def __init__(self, qubo_model: kw.qubo.QuboModel, discretizer: GenerationDiscretizer,
             reduced_data: Dict, optimizer_type: str = 'simulated_annealing'):
    """
    Initialize solver.

    Args:
        qubo_model: QUBO model to solve
        discretizer: GenerationDiscretizer instance
        reduced_data: Reduced data dictionary
        optimizer_type: Type of optimizer ('simulated_annealing' or 'cim')
    """
    self.qubo_model = qubo_model
    self.discretizer = discretizer
    self.reduced_data = reduced_data
    self.optimizer_type = optimizer_type

    # Create optimizer (optimized based on problem size)
    num_vars = discretizer.get_num_binary_vars(reduced_data['num_periods'])

    if optimizer_type == 'simulated_annealing':
        if num_vars > 100:
            # For larger reduced problems
            self.optimizer = kw.classical.SimulatedAnnealingOptimizer(
                initial_temperature=3e6,
                alpha=0.97,
                cutoff_temperature=0.001,
                iterations_per_t=600,
                size_limit=80
            )
        else:
            # For smaller reduced problems (96 variables)
            self.optimizer = kw.classical.SimulatedAnnealingOptimizer(
                initial_temperature=2e6,
                alpha=0.95,
                cutoff_temperature=0.01,
                iterations_per_t=400,
                size_limit=50
            )
    elif optimizer_type == 'cim':
        # CIM optimizer (requires hardware connection)
        base_optimizer = kw.cim.CIMOptimizer()
        self.optimizer = kw.preprocess.PrecisionReducer(base_optimizer, bit_width=8)
    else:
        raise ValueError(f"Unknown optimizer type: {optimizer_type}")

    # Create solver
    self.solver = kw.solver.SimpleSolver(self.optimizer)
```

## B.4 Solution Expansion Module Implementation

The solution expansion module (`expand_solution_to_full_scale()` function in `main.py`) implements nearest neighbor interpolation expansion:

```
def expand_solution_to_full_scale(reduced_generation: np.ndarray,
                                  reduction_info: Dict,
                                  original_data: ReducedDataLoader) -> np.ndarray:
    """
    Expand reduced solution to full-scale (24 periods, 6 units).

    Args:
        reduced_generation: Generation matrix for reduced problem
        reduction_info: Reduction information dictionary
        original_data: Original data loader

    Returns:
        Full-scale generation matrix (6 units, 24 periods)
    """
    selected_units = reduction_info['selected_units']
    selected_periods = reduction_info['selected_periods']
    period_mapping = reduction_info['period_mapping']

    # Create full-scale generation matrix
    full_generation = np.zeros((original_data.num_units, original_data.num_periods))

    # Map reduced solution to full scale
    for orig_unit_idx in range(original_data.num_units):
        if orig_unit_idx in selected_units:
            # This unit is in reduced problem
            reduced_unit_idx = selected_units.index(orig_unit_idx)
            for orig_period in range(original_data.num_periods):
                mapped_period = period_mapping[orig_period]
                if mapped_period in selected_periods:
                    reduced_period_idx = selected_periods.index(mapped_period)
                    full_generation[orig_unit_idx, orig_period] = reduced_generation[reduced_unit_idx, reduced_period_idx]
                else:
                    # Use nearest period value
                    distances = [abs(orig_period - sp) for sp in selected_periods]
                    nearest_idx = np.argmin(distances)
                    reduced_period_idx = selected_periods[nearest_idx]
                    full_generation[orig_unit_idx, orig_period] = reduced_generation[reduced_unit_idx, reduced_period_idx]
            else:
                # Unit not in reduced problem: use P_min
                full_generation[orig_unit_idx, :] = original_data.P_min[orig_unit_idx]

    return full_generation
```