JAIST Repository
https://dspace.jaist.ac.jp/

Title	JSSPのBQM生成におけるボトルネック分析と直接構築手続きの整理による高速化およびQPUの性能調査 [課題研究報告書]
Author(s)	河本, 大貴
Citation	
Issue Date	2026-03
Type	Thesis or Dissertation
Text version	author
URL	https://hdl.handle.net/10119/20548
Rights	
Description	Supervisor:井口　寧, 先端科学技術研究科, 修士(情報科学)

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

課題研究報告書

JSSP の BQM 生成におけるボトルネック分析と直接構築手続きの整理による高速化および QPU の性能調査

河本　大貴

主指導教員　井口 寧

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和 8 年 3 月

概要

量子アニーリングは組合せ最適化問題への適用が期待されているが，実問題に対する性能は十分に解明されていない．ジョブショップスケジューリング問題（Job Shop Scheduling Problem：JSSP）への量子アニーリング適用において，Carugno らは dwavebinarycsp ライブラリを用いた実装を報告しているが，$n = 26$ 程度の問題では BQM（Binary Quadratic Model）生成に 8 時間以上を要するため，大規模問題での評価が困難な状況にある．しかし，ライブラリ内部のどの処理が指数的増大を引き起こすかは特定されていない．

本研究では，この BQM 生成のボトルネックを分析し，先行研究で散在していた手法を整理することでその解消を図った．まず，Carugno らの手法を再現実装し処理時間を詳細に計測した結果，dwavebinarycsp ライブラリ内部の一意性制約処理において $O(2^T)$（T はタイムスパン）の列挙処理がボトルネックであることを特定した．20×20 インスタンスでは約 7.5 分を要し，それ以上の問題サイズでは実質的に BQM 生成が困難となることがわかった．

次に，dwavebinarycsp を経由せずに BQM を直接構築する手法を整理した．Venturelli らの論文では明示されていなかった 4 つの手続き，すなわち (a) 変数インデックス割り当て，(b)QUBO 係数への変換，(c) 係数の累積，(d)BQM オブジェクト構築を明示化し，列挙処理を経ずに Venturelli の定式化から直接 QUBO 係数を計算する手法を整理した．

本手法の評価実験では，20×20 インスタンスにおいて従来手法と比較して約 5200 倍の高速化を達成した．高速化に伴う解品質の劣化は観測されなかった．さらに，ボトルネック解消により可能となった QPU 性能評価を実施し，$n = 30$ までの問題サイズで量子アニーリングの性能限界を調査した．$n \leq 15$ では安定して最適解が得られ，$n \geq 26$ で達成率が低下することを確認した．

Abstract

Quantum annealing is expected to be applied to combinatorial optimization problems, but its performance on real-world problems has not been sufficiently clarified. In the application of quantum annealing to the Job Shop Scheduling Problem (JSSP), Carugno et al. reported an implementation using the dwavebinarycsp library; however, BQM (Binary Quadratic Model) generation requires more than 8 hours for problems of approximately $n = 26$, making evaluation of large-scale problems difficult. Yet, which specific process within the library causes the exponential growth has not been identified.

In this study, we analyzed the bottleneck in BQM generation and consolidated previously dispersed methods from prior research to resolve it. First, we reproduced Carugno's method and measured processing times in detail. As a result, we identified that $O(2^T)$ enumeration processing (where T is the timespan) in the uniqueness constraint processing within the dwavebinarycsp library is the bottleneck. For a 20×20 instance, approximately 7.5 minutes were required, making BQM generation practically infeasible for larger problem sizes.

Next, we developed a method to construct BQM directly without using dwavebinarycsp. We formalized four procedures that were not explicitly stated in Venturelli's paper: (a) variable index assignment, (b) conversion to QUBO coefficients, (c) coefficient accumulation, and (d) BQM object construction. This approach calculates QUBO coefficients directly from Venturelli's formulation without enumeration processing.

In the evaluation experiments, this method achieved approximately 5200 times speedup compared to the conventional method for a 20×20 instance. The speedup did not degrade solution quality. Furthermore, we conducted QPU performance evalua-

tion, which became possible by resolving the bottleneck, and investigated the performance limits of quantum annealing for problem sizes up to $n = 30$. We confirmed that optimal solutions are reliably obtained for $n \leq 15$, while the success rate decreases for $n \geq 26$.

目次

概要 i

Abstract ii

第 1 章　緒言 1

1.1　本論文の構成 2

1.2　表記方法 3

第 2 章　関連技術 4

2.1　はじめに 4

2.2　ジョブショップスケジューリング問題 4

2.3　時間インデックスモデル 6

2.4　量子アニーリング 11

2.5　QUBO と BQM 12

2.6　D-Wave システム 17

2.7　D-Wave Ocean SDK 19

2.8　おわりに 21

第 3 章　関連研究 22

3.1　はじめに 22

3.2　Venturelli らの研究 23

3.3　Carugno らの研究 34

3.4　Robinbux らの実装 40

3.5　本研究の位置づけ 42

3.6　おわりに 43

第 4 章　ボトルネック分析 45
4.1　はじめに 45
4.2　測定箇所 46
4.3　測定方法 48
4.4　測定結果 51
4.5　ボトルネック部分の調査 55
4.6　考察 58
4.7　まとめ 60

第 5 章　直接 BQM 構築 62
5.1　はじめに 62
5.2　ペナルティ係数の設定 75
5.3　時間窓による変数削減 78
5.4　h_3 制約の正確な係数 82
5.5　h_2 制約の双方向判定 86
5.6　手続き (a)〜(d) を統合した BQM 構築手順 89
5.7　まとめ 94

第 6 章　実験・評価 96
6.1　はじめに 96
6.2　実験 1：BQM 生成時間の比較 96
6.3　実験 2：QPU 性能評価 105
6.4　課題 114
6.5　まとめ 115

第 7 章　結言 116
7.1　まとめ 116
7.2　今後の課題 117

謝辞 118

研究業績 119

参考文献 120

付録 A　具体例 124
A.1　第 2 章の具体例 124
A.2　第 4 章の具体例 124

第 1 章

緒言

半導体の微細化によるムーアの法則は限界に近づきつつあり，従来のコンピュータアーキテクチャによる性能向上には制約が生じている．このような背景から，次世代の計算機として量子コンピュータへの注目が高まっている．量子コンピュータは大きく分けて量子ゲート方式と量子アニーリング方式があり，D-Wave Systems 社は後者の商用機を提供している．量子アニーリングは組合せ最適化問題への適用が期待されているが，実問題に対してどの程度の性能を発揮できるかは十分に解明されていない．

量子アニーリングの適用先として頻繁に検討されるベンチマーク問題の一つに，巡回セールスマン問題（Traveling Salesman Problem：TSP）がある．TSP への量子アニーリング適用が試みられてきたが，Jain ら [2] の研究では，8 ノード程度が単体 QPU 実行の上限であり，古典計算機と比較して優位な結果は報告されていない．著者らも最新の D-Wave Advantage および Advantage2 プロトタイプを用いて TSP の求解実験を行ったが（著者らの予備実験：D-Wave Advantage system4.1 使用，7〜8 都市，num_reads=1000），厳密解は得られず，古典手法と比較して有効性が低いことを確認した [3]．同じく組合せ最適化問題であるジョブショップスケジューリング問題（Job Shop Scheduling Problem：JSSP）については，Carugno ら [5] が量子アニーリングの性能評価を行い，小規模問題では最適解が得られることを示した．JSSP は製造業における生産計画など実応用が多く，量子アニーリングの適用先として有望である．

しかし，Carugno らの手法には課題が残されている．Carugno らの論文 [5] では $n = 26$（n ジョブ $\times n$ 機械の正方インスタンス，各ジョブは n 個の操作を持つ）の問題で BQM（Binary Quadratic Model）生成に 8 時間以上を要することが報告されており，大規模問題での評価が困難な状況にある．また，Carugno らはペナルティモデル処理がボトルネックであると報告しているが，dwavebinarycsp 内部のどの処理が支配的である

かまでは特定していない．QPU（Quantum Processing Unit）上のアニーリング時間（annealing_time）は μs 単位であり [35]，ジョブ投入から結果受領までのサービス時間はキュー待ち等により秒〜分となり得る [36]．一方，BQM 生成には分〜時間単位を要しており，これがボトルネックとなり QPU の性能評価が制限されている可能性がある．

本研究では，この BQM 生成のボトルネックを調査し，原因を特定する．具体的には，Carugno らが使用した dwavebinarycsp ライブラリの内部処理を分析し，ボトルネックの発生箇所と原因を明らかにする．

ボトルネックの原因を特定した上で，その解決手段を探索する．Venturelli ら [4] は JSSP の QUBO 定式化を示しているが，実装に必要な以下の手続きが明示されていない：

(a) 時間窓で削減された変数へのインデックス割り当て
(b) 数式の展開結果を QUBO 係数に変換する手順
(c) 複数の制約項が同一変数ペアに寄与する場合の係数累積
(d) QUBO 係数から BQM オブジェクトを構築する手順

本研究では，これらの手続きを明示化し，dwavebinarycsp を経由せずに BQM を直接構築する手法を整理する．

本手法の性能評価として，従来手法との BQM 生成時間の比較を行う．さらに，ボトルネックが解消されることで可能となる QPU そのものの性能評価を実施し，量子アニーリングが JSSP に対してどの程度の問題サイズまで有効かを明らかにする．

1.1 本論文の構成

本論文は以下の 7 章から構成される．

第 2 章「関連技術」では，本研究に関連する技術的背景を述べる．JSSP の問題設定と定式化，量子アニーリングと QUBO/BQM の概念，D-Wave システムと Ocean SDK について説明する．

第 3 章「関連研究」では，JSSP への量子アニーリング適用に関する先行研究を整理する．Venturelli らの時間インデックス型定式化，Carugno らの実装と評価について述べる．

第 4 章「ボトルネック分析」では，Carugno らの手法を再現実装し，処理時間を詳細に計測することでボトルネックの原因を特定する．dwavebinarycsp ライブラリ内部の処理を分析し，一意性制約における $O(2^T)$ の列挙処理が原因であることを明らかにする．

第 5 章「直接 BQM 構築」では，dwavebinarycsp を経由せずに BQM を直接構築する

手法を整理する．Venturelli らの論文では明示されていなかった 4 つの手続き (a)〜(d) を明示化し，列挙処理を経ずに Venturelli の定式化から直接 QUBO 係数を計算する手法を示す．

第 6 章「実験・評価」では，本手法の有効性を 2 つの実験により検証する．BQM 生成時間の比較実験と，QPU 性能評価実験の結果を報告する．

第 7 章「結言」では，本研究の成果をまとめ，今後の課題を述べる．

1.2 表記方法

本論文で使用する主要な記号を表 1.1 に示す．

表 1.1　主要な記号一覧

記号	意味
n	ジョブ数
m	機械数
J_j	ジョブ j（$j = 1, \ldots, n$）
M_i	機械 i（$i = 1, \ldots, m$）
$O_{j,k}$	ジョブ j の k 番目の操作
$p_{j,k}$	操作 $O_{j,k}$ の処理時間
$\mu_{j,k}$	操作 $O_{j,k}$ を処理する機械
T	タイムスパン（スケジュールの時間幅）
$x_{j,k,t}$	操作 $O_{j,k}$ が時刻 t に開始するとき 1，そうでないとき 0
h_1	順序制約（ジョブ内の操作順序）
h_2	衝突回避制約（機械の排他利用）
h_3	一意性制約（各操作は 1 回だけ開始）

注：本論文では Venturelli ら [4] と同じ制約番号を使用する（h_1：順序制約，h_2：衝突回避制約，h_3：一意性制約）．Carugno ら [5] では異なる番号体系（h_1：一意性制約，h_3：順序制約）が使用されている．先行研究との対応は第 3 章の表 3.1 を参照されたい．

第 2 章

関連技術

2.1 はじめに

本章では，本研究に関連する技術について述べる．まず JSSP の問題設定と定式化について説明し，次に量子アニーリングと QUBO/BQM の概念を述べる．続いて，D-Wave システムと Ocean SDK について説明し，最後に関連研究を整理する．

2.2 ジョブショップスケジューリング問題

2.2.1 問題設定

Job Shop Scheduling Problem（JSSP）は，n 個のジョブと m 台の機械が存在する環境において，各ジョブを構成する操作（タスク）を機械に割り当てるスケジューリング問題である [20]．JSSP は NP 困難問題として知られており [19]，問題サイズの増加に伴い計算量が指数的に増加する．問題を構成する変数の関係を以下に定義する．

集合の定義：

$$J = \{J_1, J_2, \ldots, J_n\} \qquad \text{（ジョブの集合）} \tag{2.1}$$

$$M = \{M_1, M_2, \ldots, M_m\} \qquad \text{（機械の集合）} \tag{2.2}$$

ジョブと操作の関係：各ジョブ J_j は m_j 個の操作の順序列として定義される：

$$J_j = (O_{j,1}, O_{j,2}, \ldots, O_{j,m_j}) \tag{2.3}$$

操作の属性：各操作 $O_{j,k}$ は，処理機械 $\mu_{j,k}$ と処理時間 $p_{j,k}$ の組として特徴づけられる：

$$O_{j,k} = (\mu_{j,k}, p_{j,k}), \quad \mu_{j,k} \in M, \quad p_{j,k} \in \mathbb{Z}^+ \tag{2.4}$$

具体例：2 ジョブ ×2 機械の例を考える．機械 $M = \{M_1, M_2\}$ と 2 つのジョブ J_1, J_2 があるとする．

J_1 は 2 つの操作からなり，$J_1 = (O_{1,1}, O_{1,2})$ と表される：

- $O_{1,1}$（ジョブ 1 の 1 番目の操作）：$\mu_{1,1} = M_1$（機械 M_1 で処理），$p_{1,1} = 3$（処理時間 3）
- $O_{1,2}$（ジョブ 1 の 2 番目の操作）：$\mu_{1,2} = M_2$（機械 M_2 で処理），$p_{1,2} = 2$（処理時間 2）

J_2 も同様に 2 つの操作からなり，$J_2 = (O_{2,1}, O_{2,2})$ と表される：

- $O_{2,1}$（ジョブ 2 の 1 番目の操作）：$\mu_{2,1} = M_2$，$p_{2,1} = 2$
- $O_{2,2}$（ジョブ 2 の 2 番目の操作）：$\mu_{2,2} = M_1$，$p_{2,2} = 4$

この例では，$O_{1,1}$ と $O_{2,2}$ が同じ機械 M_1 を使用し，$O_{1,2}$ と $O_{2,1}$ が同じ機械 M_2 を使用する．同一機械上では複数の操作を同時に実行できないため，実行順序を決定する必要がある．JSSP の目的は，この機械競合制約を満たしつつ，すべてのジョブが完了するまでの時間を最小化するスケジュールを決定することである．

2.2.2 目的関数

前節では，各操作 $O_{j,k}$ の属性として処理機械 $\mu_{j,k}$ と処理時間 $p_{j,k}$ を定義した．これらは問題の入力として与えられる値である．一方，各操作の処理開始時刻は決定すべき変数である．操作 $O_{j,k}$ が機械 $\mu_{j,k}$ 上で処理を開始する時刻を $S_{j,k}$ と表す．

操作 $O_{j,k}$ は時刻 $S_{j,k}$ に処理を開始し，処理時間 $p_{j,k}$ の経過後に処理が終了するため，処理終了時刻は $S_{j,k} + p_{j,k}$ となる．ジョブ J_j の完了時刻 C_j は，最後の操作 O_{j,m_j} の終了時刻として定義される：

$$C_j = S_{j,m_j} + p_{j,m_j} \tag{2.5}$$

すべてのジョブの完了時刻の最大値をメイクスパン $C_{\max}$ と呼び，以下のように定義する：

$$C_{\max} = \max(C_1, C_2, \ldots, C_n) \tag{2.6}$$

JSSP の目的は，このメイクスパン $C_{\max}$ を最小化することである．

具体例：前節の 2 ジョブ ×2 機械の例で考える．仮に以下のスケジュールが得られたとする：

- J_1：$S_{1,1} = 0$（時刻 0 に $O_{1,1}$ 開始）→ 時刻 $0 + p_{1,1} = 3$ に終了 → $S_{1,2} = 3$ → 時刻 $3 + p_{1,2} = 5$ に終了
- J_2：$S_{2,1} = 0$（時刻 0 に $O_{2,1}$ 開始）→ 時刻 $0 + p_{2,1} = 2$ に終了 → $S_{2,2} = 3$ → 時刻 $3 + p_{2,2} = 7$ に終了

このとき，$C_1 = 5$，$C_2 = 7$ となり，メイクスパンは $C_{\max} = \max(5, 7) = 7$ である．

本研究では，メイクスパン最小化の代わりに，与えられた timespan T 以内にすべてのジョブを完了できるかどうかを判定する意思決定版の問題を扱う．

2.2.3 基本制約

JSSP には以下の基本制約が課される．

2.2.3.1 順序制約

同一ジョブ内の操作は，指定された順序で実行されなければならない．

$$S_{j,k} + p_{j,k} \leq S_{j,k+1} \quad \forall j, k < m_j \tag{2.7}$$

2.2.3.2 資源競合制約

同一機械上で複数の操作が同時に実行されてはならない．同じ機械を使用する 2 つの操作 $O_{j,k}$ と $O_{j',k'}$（すなわち $\mu_{j,k} = \mu_{j',k'}$）について：

$$S_{j,k} + p_{j,k} \leq S_{j',k'} \quad \text{または} \quad S_{j',k'} + p_{j',k'} \leq S_{j,k} \tag{2.8}$$

つまり，どちらかの操作が終了してから他方が開始しなければならない．

具体例は付録 A.1.1 を参照．

2.3 時間インデックスモデル

前節では，JSSP において各操作の処理開始時刻 $S_{j,k}$ を決定する必要があることを述べた．しかし，D-Wave などの量子アニーリングマシンは QUBO（0-1 変数の 2 次最適化問題）を解くため，$S_{j,k} = 3$ のような整数値をそのまま扱うことができない．

時間インデックスモデルは，この問題を解決するための表現方法である．時間を離散的な時刻（$t = 0, 1, 2, \ldots, T-1$，t は非負整数，合計 T 個）に分割し，「操作 $O_{j,k}$ が時刻 t に開始するか否か」を 0-1 変数で表現する．この時刻の総数 T を timespan（タイムスパン）と呼ぶ．

2.3.1 時間インデックス変数の定義

決定変数 $x_{j,k,t} \in \{0,1\}$ を以下のように定義する：

$$x_{j,k,t} = \begin{cases} 1 & \text{操作 } O_{j,k} \text{ が時刻 } t \text{ に開始する場合} \\ 0 & \text{それ以外} \end{cases} \tag{2.9}$$

前節で定義した開始時刻 $S_{j,k}$ との関係は以下のようになる：

$$S_{j,k} = t \quad \Leftrightarrow \quad x_{j,k,t} = 1 \tag{2.10}$$

具体例：前節の 2 ジョブ ×2 機械の例で，$O_{1,1}$ が時刻 3 に開始する場合（$S_{1,1} = 3$）を考える．timespan $T = 6$ とすると，$O_{1,1}$ に対応する変数は $t = 0, 1, \ldots, 5$ の 6 個であり：

$$x_{1,1,0} = 0, \quad x_{1,1,1} = 0, \quad x_{1,1,2} = 0, \quad x_{1,1,3} = 1, \quad x_{1,1,4} = 0, \quad x_{1,1,5} = 0$$

となる．時刻 3 のみが 1 で，他はすべて 0 である．

2.3.2 一意性制約

各操作は 1 回だけ実行される（複数回実行されない）ことを保証する，時間インデックスモデル特有の制約である．

2.2.3 節の $S_{j,k}$ モデルでは，$S_{1,1} = 3$ のように 1 つの変数で開始時刻を表現するため，「複数回実行」という状態はそもそも表現できない．一方，時間インデックスモデルでは各時刻に独立した変数があるため，制約がないと以下のような「2 回実行」の状態が許容されてしまう：

$$x_{1,1,0} = 1, \quad x_{1,1,3} = 1 \quad \text{（時刻 0 と時刻 3 で開始 = 2 回実行）}$$

これを防ぐため，以下の制約を課す：

$$\sum_{t=0}^{T-1} x_{j,k,t} = 1 \quad \forall j, k \tag{2.11}$$

2.3.3 順序制約

同一ジョブ内の操作順序を保証する制約である．

2.2.3 節の $S_{j,k}$ モデルでは，$S_{j,k} + p_{j,k} \leq S_{j,k+1}$ という 1 つの式で表現できた．例えば，$O_{1,1}$（処理時間 $p_{1,1} = 3$）の後に $O_{1,2}$ を実行する場合，$S_{1,1} + 3 \leq S_{1,2}$ となる．

一方，時間インデックスモデルでは，$O_{1,1}$ が時刻 t に開始するなら，$O_{1,2}$ は時刻 $t+3$ 以降でなければならない．これを 0-1 変数で表現すると，以下のようにすべての時刻の組について制約が必要となる：

$$x_{1,1,0} + x_{1,2,0} \leq 1, \quad x_{1,1,0} + x_{1,2,1} \leq 1, \quad x_{1,1,0} + x_{1,2,2} \leq 1, \quad \ldots$$

一般化すると：

$$x_{j,k,t} + x_{j,k+1,t'} \leq 1 \quad \forall j, k < m_j, \forall t, t' \text{ where } t' < t + p_{j,k} \tag{2.12}$$

2.3.4 衝突回避制約

同一機械上で複数の操作が同時に実行されないことを保証する制約であり，2.2.3 節の資源競合制約に対応する．

2.2.3 節の $S_{j,k}$ モデルでは，「開始時刻の大小関係」で制約を表現した．同一機械を使用する $O_{1,1}$ と $O_{2,2}$ について，「$S_{1,1} + 3 \leq S_{2,2}$ または $S_{2,2} + 4 \leq S_{1,1}$」となる．

一方，時間インデックスモデルでは，「同時に 1 になってはいけない変数のペア」で制約を表現する．例えば，$O_{1,1}$ が時刻 0 に開始し（$x_{1,1,0} = 1$），処理時間が 3 の場合，$O_{2,2}$ は時刻 0, 1, 2 には開始できない：

$$x_{1,1,0} + x_{2,2,0} \leq 1, \quad x_{1,1,0} + x_{2,2,1} \leq 1, \quad x_{1,1,0} + x_{2,2,2} \leq 1$$

一般化すると，同一機械を使う 2 つの操作 $O_{j,k}$ と $O_{j',k'}$ について，時間的に重なる（オーバーラップする）条件は以下の双方向条件で表される：

$$(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t) \tag{2.13}$$

この条件を満たす時刻の組み合わせ (t, t') に対して，以下の制約を課す：

$$x_{j,k,t} + x_{j',k',t'} \leq 1 \tag{2.14}$$

なお，操作ペアを無順序（$(j,k) < (j',k')$）で列挙すれば，同一ペアの二重カウントを防げる．

2.3.5 timespan の設定指針

時間インデックスモデルでは，変数 $x_{j,k,t}$ を $t = 0, 1, \ldots, T-1$ の範囲（T 個）で生成するため，事前に timespan T を設定する必要がある．しかし，最適なメイクスパン $C^*_{\max}$

は問題を解くまで分からないため，T は「最適解がこの時間内に収まるだろう」という見積もりとして設定する．

この T の設定は，変数数と解の存在可能性のトレードオフに影響する：

- T が小さすぎる：実行可能解が存在しない可能性（$T < C^*_{\max}$ の場合）
- T が大きすぎる：変数数が増大し，計算コストが増加

Carugno ら [5] は，処理時間を 1 に固定した $n \times n$ の正方インスタンスにおいて，$T = n+1$ から $T = n+6$ の範囲で評価を行い，$T = n+1$ で最良の結果を得ている．本研究でもこの設定に従い，$T = n+1$ を基本設定とする．

ただし，この設定は処理時間が均一で最適メイクスパンが事前に推定可能な場合に有効である．実際の問題では最適解が未知であるため，timespan の適切な設定は依然として課題である [5].

2.3.6 時間窓による変数削減

前節で示したように，timespan T を設定すると $t = 0, 1, \ldots, T-1$ の各時刻に変数を生成する．しかし，すべての時刻が実際に開始可能とは限らない．2.2.3 節で述べた順序制約により，「この操作は絶対にこの時刻には開始できない」という時刻が存在する．

時間窓とは，timespan で与えられた時刻の集合から開始不可能な時刻を除外し，各操作が開始可能な時刻の範囲に絞り込んだものである．量子アニーリングはこの時間窓の中で最適な開始時刻の組み合わせを探索する．

各操作 $O_{j,k}$ について，処理を開始できる時刻の範囲を**時間窓**（time window）と呼び，$[ES_{j,k}, LS_{j,k}]$ で表す．時間窓内のどの時刻でも処理を開始でき，開始後は処理時間 $p_{j,k}$ で終了する．timespan が全体の時刻の集合 $\{0, 1, \ldots, T-1\}$（T 個）を定めるのに対し，時間窓はその中で各操作が開始可能な時刻の部分集合を定める．

具体例：$T = 9$（時刻 $t = 0, 1, \ldots, 8$），ジョブ $J_1 = (O_{1,1}, O_{1,2})$（処理時間それぞれ 3, 2）の場合を考える．

$O_{1,1}$ について：

- 先行操作がないため，最早で時刻 0 から開始可能（$ES_{1,1} = 0$）
- $O_{1,1}$ と $O_{1,2}$ の処理時間の合計は $3 + 2 = 5$ なので，時刻 4 より後に開始すると $T = 9$ に収まらない（$LS_{1,1} = 9 - 5 = 4$）
- よって時間窓は $[0, 4]$

$O_{1,2}$ について：

- $O_{1,1}$ の処理時間が 3 なので，最早で時刻 3 から開始可能（$ES_{1,2} = 3$）
- 処理時間が 2 なので，時刻 7 より後に開始すると $T = 9$ に収まらない（$LS_{1,2} = 9 - 2 = 7$）
- よって時間窓は $[3, 7]$

表中の ∘ はその時刻に開始可能であることを示す（∘ の数は処理時間ではなく，開始可能な時刻の選択肢の数である）．例えば $O_{1,1}$ は時刻 0, 1, 2, 3, 4 のいずれかで開始でき，どの時刻に開始しても処理時間 3 単位で終了する．∘ がついている時刻のどれを選ぶかは量子アニーリングが探索する．∘ の数が多いほど選択肢（探索の余地）が大きい：

時刻 t	0	1	2	3	4	5	6	7	8	
$O_{1,1}$ の時間窓 $[0, 4]$	∘	∘	∘	∘	∘					変数 5 個
$O_{1,2}$ の時間窓 $[3, 7]$				∘	∘	∘	∘	∘		変数 5 個

timespan だけで変数を作ると，各操作に対して $t = 0, 1, \ldots, 8$ の 9 個の変数が必要となり，2 つの操作で $2 \times 9 = 18$ 個の変数となる．一方，時間窓を考慮すると，$O_{1,1}$ は 5 個，$O_{1,2}$ は 5 個で，合計 10 個に削減できる．

このように，時間窓は各操作に対して 1 つずつ存在する．n 個のジョブがあり各ジョブに n 個の操作がある場合，時間窓は操作の総数である n^2 個となる．

時間窓は以下のように計算される：

最早開始時刻（Earliest Start）：先行する操作がすべて完了する時刻

$$ES_{j,k} = \sum_{k' < k} p_{j,k'} \tag{2.15}$$

最遅開始時刻（Latest Start）：timespan 内に処理を完了できる最後の時刻

$$LS_{j,k} = T - \sum_{k' \geq k} p_{j,k'} \tag{2.16}$$

理論上は，時間窓外（$t < ES_{j,k}$ または $t > LS_{j,k}$）の変数を生成しないことで変数数を削減できる．ただし，Carugno らの dwavebinarycsp 実装では全変数を生成した後に時間窓外の変数を 0 に固定する方式（後段 pruning）であり，制約生成コストの観点では変数削減の恩恵を受けられない（第 5 章参照）．

2.4 量子アニーリング

2.4.1 概要

量子アニーリング（Quantum Annealing, QA）は，量子力学的な効果を利用して組合せ最適化問題の解を探索する計算手法である [16]．組合せ最適化問題をエネルギー最小化問題として定式化し，量子トンネル効果を利用して最適解を探索する．量子トンネル効果とは，古典力学では乗り越えられないエネルギー障壁を，量子力学的な確率振幅により透過できる現象である．この効果により，局所最適解に捕捉されることなく大域的な最適解を探索できる可能性がある [17]．量子アニーラに問題を投入するには，問題を QUBO 形式または Ising 形式で記述する必要がある（詳細は 2.5 節で述べる）．

2.4.2 Ising モデルとアニーリング過程

Ising モデルは，物質中の磁性を記述するために提案された物理モデルである．モデル中の各要素は**スピン変数** $s_i \in \{-1, +1\}$ で表され，これは磁石の向き（上向き $+1$ または下向き -1）に対応する．

スピン同士は相互作用を持ち，系全体のエネルギーは以下で表される：

$$E = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j \tag{2.17}$$

ここで h_i は各スピンにかかる外部磁場，J_{ij} はスピン間の結合強度である．

最適化問題を解く際には，この物理モデルを利用する．決定変数の値をスピンの向き s_i に対応させ，目的関数をエネルギー E として表現する．スピンの集合がエネルギー最小の状態に落ち着くことで，最適化問題の解が得られる．

なお，QUBO 形式（$x_i \in \{0, 1\}$）とは変数変換 $s_i = 2x_i - 1$ により相互変換が可能である．量子アニーラは，この Ising モデルのエネルギーを物理的に実装し，基底状態（最低エネルギー状態）を探索する．

量子力学では，スピンの集合全体のエネルギーを**ハミルトニアン** H という関数（演算子）で表す．スピンの集合はエネルギーが最小となる状態（基底状態）に落ち着く性質があるため，ハミルトニアンを最小化することが最適化問題を解くことに対応する．

D-Wave 装置で想定される時間依存ハミルトニアンは，以下で与えられる [11]：

$$H(s) = -\frac{A(s)}{2} \sum_i \sigma_i^x + \frac{B(s)}{2} \Big(\sum_i h_i \sigma_i^z + \sum_{i<j} J_{ij} \sigma_i^z \sigma_j^z \Big) \tag{2.18}$$

右辺第 2 項の h_i と J_{ij} が解くべき問題を記述する部分であり，これが QUBO/BQM 形

式で与えられる（詳細は 2.5 節で述べる）．量子アニーリングはこのハミルトニアンを最小化し，基底状態に到達したスピン変数の値を観測することで最適化問題の解を得る．

ここで，$s = t/t_a \in [0, 1]$ はアニーリングの正規化時間（t_a はアニーリング時間），σ_i^x, σ_i^z はパウリ演算子（量子ビットの状態を操作する基本演算子）である．$A(s)$ はドライバスケール（量子揺らぎの強さを制御），$B(s)$ は問題スケール（問題ハミルトニアンの強さを制御）であり，アニーリング開始時は $A(s) \gg B(s)$，終了時は $A(s) \ll B(s)$ となるよう時間変化させる．

2.4.3 サンプリングと確率的解探索

量子アニーリングは確率的な解探索手法であり，1 回のアニーリングで最適解が得られるとは限らない [11].

サンプリングとは，1 回のアニーリング実行後にスピン変数の状態を観測し，1 つの解（サンプル）を得ることである．量子アニーリングでは，同じ問題に対して複数回のアニーリングを実行し，複数のサンプルを収集する．D-Wave では，この実行回数を `num_reads` パラメータで指定する．1 回のサンプリングはマイクロ秒単位で完了するため，回数を増やすコストは比較的低い．典型的には 100〜1000 回程度を指定し，得られたサンプルの中から最もエネルギーの低い解（最良解）を選択する．

2.5 QUBO と BQM

本節では，量子アニーラに問題を投入するために必要な QUBO と BQM について説明する．量子アニーラは，特定の数学的形式（QUBO または Ising 形式）で記述された問題のみを受け付けるため，JSSP のような制約付き最適化問題を量子アニーラで解くには，まず問題をこれらの形式に変換する必要がある．QUBO の一般的な定式化手法については Glover ら [21] によるチュートリアルが詳しく，Lucas[1] は多くの NP 問題を Ising 形式（QUBO と等価）で定式化する方法を体系的に整理している．なお，JSSP に特化した QUBO 定式化については第 3 章で述べる Venturelli らの研究を参照されたい．

2.5.1 QUBO 定義

Quadratic Unconstrained Binary Optimization（QUBO：制約なし二次二値最適化）は，二値変数 $x_i \in \{0, 1\}$ に対する二次形式の最適化問題である．

「Unconstrained」の意味：最適化問題は形式により分類される．

形式	構成	例
Constrained（制約付き）	目的関数 + 制約式（別々に記述）	線形計画法
Unconstrained（制約なし）	目的関数のみ	QUBO

すなわち，「Unconstrained」は形式の分類名であり，制約付き問題を扱えないという意味ではない．制約付き問題を QUBO 形式で扱う方法については次節で説明する．本節では，QUBO の定義と係数の意味について説明する．

二次式と行列表現：N 個の二値変数 $x_1, x_2, \ldots, x_N$ に対する二次式は，以下のように表される：

$$H = Q_{1,1}x_1 + Q_{2,2}x_2 + \cdots + Q_{N,N}x_N + Q_{1,2}x_1x_2 + Q_{1,3}x_1x_3 + \cdots \tag{2.19}$$

ここで，$Q_{i,i}$ は変数 x_i 単独の係数（一次項），$Q_{i,j}$（$i \neq j$）は変数 x_i と x_j の積の係数（二次項）である．

これらの係数をまとめて行列 Q として表現すると：

$$H_{\mathrm{QUBO}} = \sum_i Q_{i,i}x_i + \sum_{i<j} Q_{i,j}x_ix_j = \boldsymbol{x}^T Q \boldsymbol{x} \tag{2.20}$$

係数の意味：QUBO は H_{QUBO} を最小化する問題である．係数の符号と値は以下の意味を持つ：

- 一次項 $Q_{i,i} < 0$：$x_i = 1$ にするとエネルギーが下がる（$x_i = 1$ が好ましい）
- 一次項 $Q_{i,i} > 0$：$x_i = 1$ にするとエネルギーが上がる（$x_i = 0$ が好ましい）
- 二次項 $Q_{i,j} < 0$：$x_i = x_j = 1$ にするとエネルギーが下がる（両方 1 が好ましい）
- 二次項 $Q_{i,j} > 0$：$x_i = x_j = 1$ にするとエネルギーが上がる（両方 1 は避けたい）

2.5.2 QUBO の構成手順

本節では，具体例を通じて QUBO の構成手順を説明する．

QUBO を解くとは：QUBO を解くとは，すべての変数の組み合わせ（2^n 通り）の中から，ハミルトニアン H が最小となる組み合わせを見つけることである．2 変数の場合，(x_1, x_2) の組み合わせは $(0,0), (0,1), (1,0), (1,1)$ の 4 通りあり，それぞれについて H を計算し，最小値を与える組み合わせを探索する．

係数調整による制約の埋め込み（ペナルティ化）：QUBO は「Unconstrained（制約なし）」と名が付くように，制約式を別途記述する場所がない．しかし，係数を適切に設定

することで,「制約を満たす解のエネルギーが低く，違反する解のエネルギーが高くなる」ように設計できる．この手法を**ペナルティ化**と呼ぶ.

目的関数とペナルティ項：実際の最適化問題（JSSP など）では，ハミルトニアンを以下のように構成する：

$$H = H_{\text{目的関数}} + \lambda \cdot H_{\text{ペナルティ}} \tag{2.21}$$

- $H_{\text{目的関数}}$：最適化したい量．良い解ほどエネルギーが低い
- $H_{\text{ペナルティ}}$：満たすべき制約．違反するとエネルギーが増加

$H_{\text{ペナルティ}}$ は，制約を満たす解では 0，違反する解では正の値を取るように設計する．λ（ペナルティ係数）を十分大きく設定することで，制約違反の解が選ばれにくくなる．

具体例（2 変数）：2 変数 x_1, x_2 で，$(x_1, x_2) = (1, 0)$ をエネルギー最小としたい場合を考える．これは，以下の制約を満たす QUBO を構成することで実現できる：

- $x_1 = 1$ が好ましい（係数を負に設定）
- $x_2 = 0$ が好ましい（係数を正に設定）
- $x_1 = x_2 = 1$ は避けたい（係数を正に設定）

係数の決定：上記の状況を QUBO で表現するには，各係数を以下のように設定する：

- $x_1 = 1$ にしたい $\rightarrow Q_{1,1} < 0$ とする $\rightarrow$ 例えば $Q_{1,1} = -1$
- $x_2 = 1$ は避けたい $\rightarrow Q_{2,2} > 0$ とする $\rightarrow$ 例えば $Q_{2,2} = 2$
- 両方 1 は避けたい $\rightarrow Q_{1,2} > 0$ とする $\rightarrow$ 例えば $Q_{1,2} = 3$

これらの係数を用いると，ハミルトニアンは以下のようになる：

$$H = -1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_1 x_2 \tag{2.22}$$

これを QUBO 行列で表すと：

$$Q = \begin{pmatrix} -1 & 3 \\ 0 & 2 \end{pmatrix} \tag{2.23}$$

対角要素 $Q_{1,1} = -1$ と $Q_{2,2} = 2$ が一次項の係数，非対角要素 $Q_{1,2} = 3$ が二次項の係数である．Q は上三角行列として表現する（$Q_{2,1} = 0$）.

検算：すべての組み合わせについてエネルギーを計算する：

- $(x_1, x_2) = (0, 0)$：$H = -0 + 2 \cdot 0 + 3 \cdot 0 \cdot 0 = 0$
- $(x_1, x_2) = (0, 1)$：$H = -0 + 2 \cdot 1 + 3 \cdot 0 \cdot 1 = 2$
- $(x_1, x_2) = (1, 0)$：$H = -1 + 2 \cdot 0 + 3 \cdot 1 \cdot 0 = -1 \leftarrow$ 最小

- $(x_1, x_2) = (1, 1)$：$H = -1 + 2 \cdot 1 + 3 \cdot 1 \cdot 1 = 4$

確かに $(x_1, x_2) = (1, 0)$ がエネルギー最小となり，意図した制約を満たす QUBO が構成できた．

3 変数への拡張：変数が増えても同様の規則で行列を構成できる．3 変数 x_1, x_2, x_3 の場合，$(x_1, x_2, x_3) = (1, 0, 1)$ をエネルギー最小としたい場合を考える．係数を以下のように設定する：

- $Q_{1,1} = -1$（$x_1 = 1$ が好ましい）
- $Q_{2,2} = 2$（$x_2 = 0$ が好ましい）
- $Q_{3,3} = -3$（$x_3 = 1$ が好ましい）
- $Q_{1,3} = -2$（$x_1 = x_3 = 1$ が好ましい）
- $Q_{1,2} = 4$（$x_1 = x_2 = 1$ は避けたい）
- $Q_{2,3} = 5$（$x_2 = x_3 = 1$ は避けたい）

QUBO 行列は：

$$Q = \begin{pmatrix} -1 & 4 & -2 \\ 0 & 2 & 5 \\ 0 & 0 & -3 \end{pmatrix} \tag{2.24}$$

すべての組み合わせ（$2^3 = 8$ 通り）についてエネルギーを計算すると，$(x_1, x_2, x_3) = (1, 0, 1)$ で $H = -6$ となり，最小値を取る．一般に，N 変数の場合は $N \times N$ の上三角行列となる．

2.5.3 BQM との関係

前節では，QUBO 行列 Q を用いて最適化問題を表現した．D-Wave QPU にこの問題を投入するには，QUBO 行列を BQM（Binary Quadratic Model：二値二次モデル）というデータ構造に変換する必要がある．

BQM とは：BQM は，D-Wave 社が Ocean SDK（dimod ライブラリ）で規定したデータ構造である [12]．QUBO 行列が理論・数学的な表現であるのに対し，BQM はプログラム上で扱うための実装表現である．

BQM は以下のエネルギー関数を表現する：

$$E(\boldsymbol{v}) = \sum_i a_i v_i + \sum_{i<j} b_{i,j} v_i v_j + c \tag{2.25}$$

ここで，a_i は線形バイアス，$b_{i,j}$ は二次バイアス，c はオフセット（定数項）である．

BQM は以下の 4 つの要素から構成される：

- 線形項（linear）：各変数の係数を格納する辞書．形式は $\{v : b, \ldots\}$
- 二次項（quadratic）：変数ペアの係数を格納する辞書．形式は $\{(u, v) : b, \ldots\}$
- オフセット（offset）：定数項（スカラー値）
- 変数タイプ（vartype）：BINARY（0/1）または SPIN（-1/+1）

変換の手続き：QUBO 行列から BQM への変換は以下のように行う：

- QUBO 行列の対角要素 $Q_{i,i}$ → linear（線形項の辞書）
- QUBO 行列の非対角要素 $Q_{i,j}$ → quadratic（二次項の辞書）
- オフセット $= 0$
- vartype = BINARY

具体例（2 変数）：前節の 2 変数 QUBO（$(x_1, x_2) = (1, 0)$ をエネルギー最小とする例）を BQM に変換する．

変換前（QUBO 行列）：

$$Q = \begin{pmatrix} -1 & 3 \\ 0 & 2 \end{pmatrix} \tag{2.26}$$

変換後（BQM）：

$$\text{linear} = \{x_1 : -1,\ x_2 : 2\} \tag{2.27}$$
$$\text{quadratic} = \{(x_1, x_2) : 3\} \tag{2.28}$$
$$\text{offset} = 0 \tag{2.29}$$
$$\text{vartype} = \text{BINARY} \tag{2.30}$$

具体例（3 変数）：前節の 3 変数 QUBO（$(x_1, x_2, x_3) = (1, 0, 1)$ をエネルギー最小とする例）を BQM に変換する．

変換前（QUBO 行列）：

$$Q = \begin{pmatrix} -1 & 4 & -2 \\ 0 & 2 & 5 \\ 0 & 0 & -3 \end{pmatrix} \tag{2.31}$$

変換後（BQM）：

$$\text{linear} = \{x_1 : -1,\ x_2 : 2,\ x_3 : -3\} \tag{2.32}$$
$$\text{quadratic} = \{(x_1, x_2) : 4,\ (x_1, x_3) : -2,\ (x_2, x_3) : 5\} \tag{2.33}$$
$$\text{offset} = 0 \tag{2.34}$$
$$\text{vartype} = \text{BINARY} \tag{2.35}$$

QUBO 行列と BQM は，表現形式は異なるが，同じハミルトニアンを表す．

本研究では，JSSP の制約を QUBO 行列として構成し，それを BQM に変換して D-Wave QPU に投入する．

2.6 D-Wave システム

本節では，量子アニーリングを実行するハードウェアである D-Wave 量子アニーリングマシンについて説明する．D-Wave 社は 2011 年に初の商用量子アニーリングマシンを発表し [18]，以降継続的に改良を重ねてきた．その性能評価については McGeoch ら [22] や Rønnow ら [27] による研究がある．

2.6.1 D-Wave システムの世代とトポロジ

D-Wave 社は，量子アニーリングマシンを継続的に改良してきた [13, 14]．表 2.1 に各世代の主要なスペックを示す．

表 2.1 D-Wave 量子アニーリングマシンの世代比較

システム	トポロジ	次数	量子ビット数	カプラー数
D-Wave 2000Q	Chimera C_{16}	6	>2,000	>6,000
Advantage	Pegasus P_{16}	15	>5,000	>35,000
Advantage2*	Zephyr Z_{12}	20	4,589	41,729

*Advantage2_system1.10（2025 年 5 月一般提供開始 [31]）の提供中 working graph 値 [30]．Zephyr Z12 ベースで約 4,400+ qubits / 40,000+ couplers 級 [37]．将来フルスケール（Zephyr Z15）では理論上 7,440 qubits 級を想定 [38].

表中の用語を以下に説明する．

トポロジ：量子ビット間の接続構造を定義するグラフである．D-Wave のシステムでは，すべての量子ビットが互いに接続されているわけではなく，特定のパターンに従って隣接する量子ビット同士のみが接続されている．

次数：1 つの量子ビットに直接接続されている他の量子ビットの数を意味する．世代が進むにつれて次数が増加しており，より複雑な問題を効率的に表現できるようになっている．

カプラー：2 つの量子ビット間を物理的に接続する素子である．カプラーは量子ビット間の相互作用（QUBO の二次項 Q_{ij}）を実現する．

図 2.1 に，これらの概念を模式的に示す．

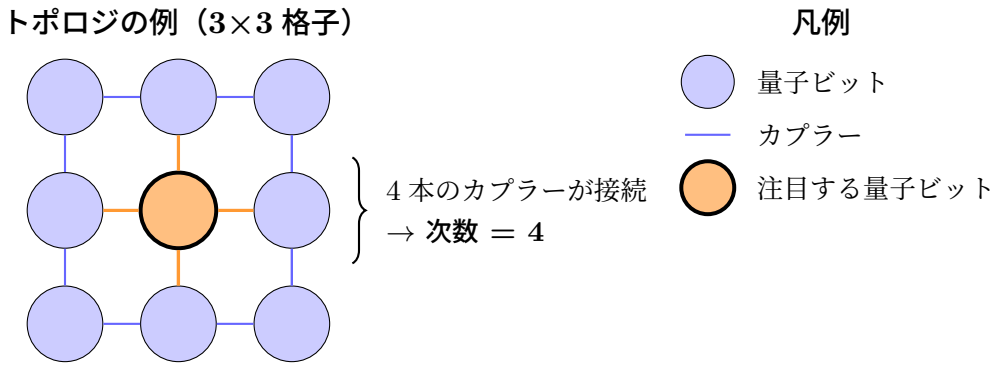


図 2.1　トポロジ・次数・カプラーの概念図

2.6.2　ハードウェアグラフと埋め込み

マイナー埋め込み：QUBO の論理変数を D-Wave の物理量子ビットに写像する処理である．論理問題グラフ（QUBO の変数と相互作用）をハードウェアグラフ（量子ビットとカプラー）に埋め込む必要がある．

ハードウェアの接続性の制約から，1 つの論理変数を複数の物理量子ビットで表現する場合がある．これら複数の量子ビットの集合を**チェーン**と呼ぶ．チェーン内の量子ビットは同じ値を取る必要があり，**チェーン強度**パラメータで結合の強さを制御する．

図 2.2 にマイナー埋め込みの例を示す．

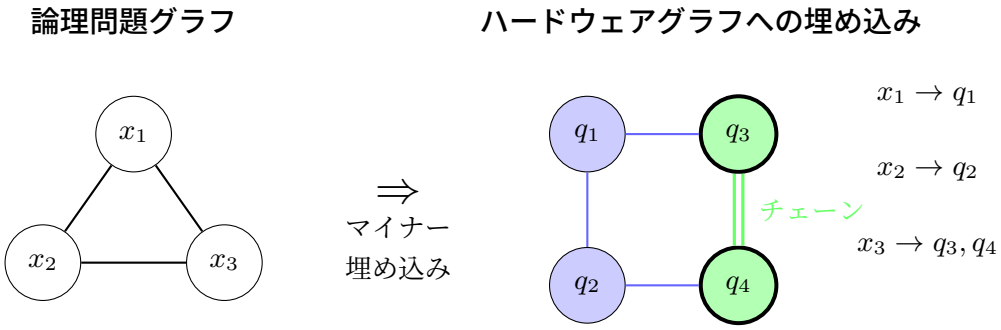


図 2.2　マイナー埋め込みの例：論理変数 x_3 を 2 つの量子ビット（チェーン）で表現

トポロジの次数が高いほど，同じサイズの問題に対してより短いチェーンで埋め込みが可能となる．短いチェーンはチェーンブレイク（チェーン内の量子ビットが異なる値を取る現象）のリスクを低減し，解品質の向上に寄与する．

2.6.3 アニーリング実行サイクル

D-Wave QPU での 1 回のアニーリング実行は，以下のフェーズから構成される：

1. **初期化**：量子ビットを均一な重ね合わせ状態に初期化
2. **アニーリング**：ハミルトニアンを時間発展させ，問題ハミルトニアンの基底状態を探索
3. **読み出し**：量子ビットの状態を古典ビットとして測定

1 回のアニーリングはマイクロ秒単位で完了するが，確率的な手法であるため，複数回（`num_reads` パラメータで指定）実行して最良解を選択する．

2.7 D-Wave Ocean SDK

2.7.1 Leap と Ocean SDK の実行フロー

Leap[11] は，D-Wave 社が提供するクラウドサービスであり，インターネット経由で D-Wave の QPU にアクセスする手段を提供する．ユーザーは API トークンを用いて認証を行い，クラウド上の QPU にジョブを送信してサンプリング結果を取得する．

Ocean SDK[29] は，D-Wave 社が提供する Python ベースのオープンソース SDK である．主要なコンポーネントを以下に示す：

dwavebinarycsp　CSP（制約充足問題）形式で記述された制約を BQM に変換するライブラリである．

penaltymodel　dwavebinarycsp が内部で使用するライブラリである．各制約に対して，制約充足時にエネルギーが低く，違反時に高くなる QUBO 係数を自動生成する．

dimod　BQM や QUBO モデルの表現と操作を行うライブラリである．

minorminer　論理グラフからハードウェアグラフへのマイナー埋め込み [28] を探索するライブラリである [24].

DWaveSampler　Leap 経由で QPU と通信し，サンプリングを実行する．

図 2.3 に Ocean SDK の実行フローを示す．

2.7.2 dwavebinarycsp

dwavebinarycsp は，制約充足問題（CSP）形式で制約を記述し，自動的に BQM へ変換するライブラリである．

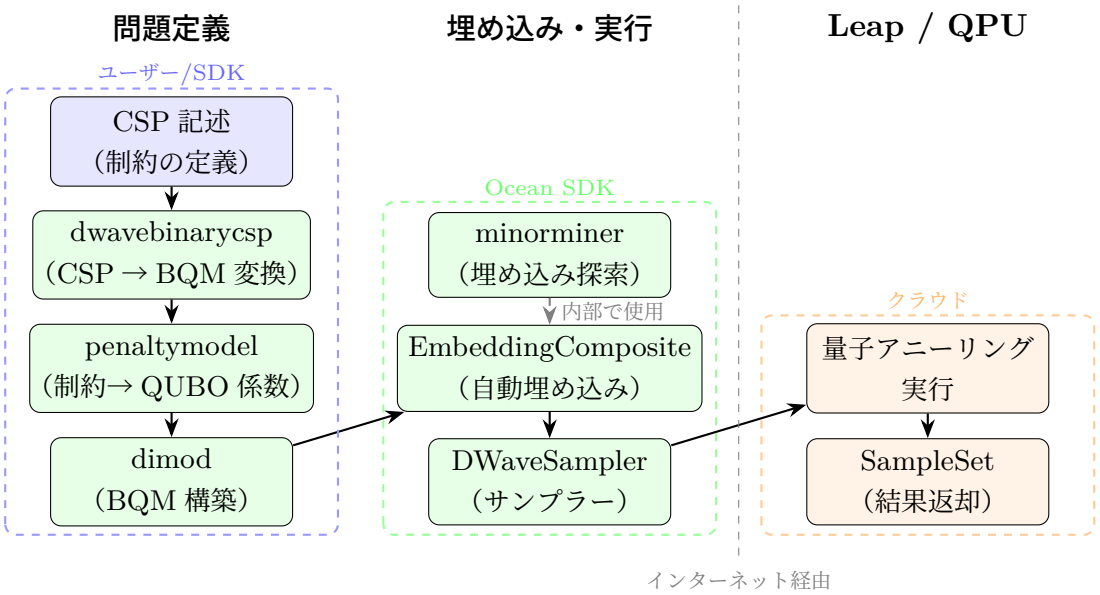


図 2.3　Ocean SDK の主要コンポーネントと実行フロー

CSP モデルの構築：ユーザーは `ConstraintSatisfactionProblem` オブジェクトを作成し，`add_constraint()` メソッドで制約を追加する．制約は「変数のリスト」と「充足条件を判定する関数」のペアで指定する．

BQM への変換：`stitch()` 関数を呼び出すことで，CSP を BQM に変換する．内部では penaltymodel ライブラリを用いて，各制約を満たす構成にはエネルギー 0，違反する構成には正のエネルギーを与える QUBO 係数を自動生成する．

2.7.3　dimod.BinaryQuadraticModel

dimod.BinaryQuadraticModel（BQM）は，D-Wave Ocean SDK における二値二次モデルの標準的なデータ構造である．dimod は BQM の「入れ物」（データ構造）を提供するライブラリであり，BQM の中身である QUBO 係数（linear, quadratic, offset）は別途計算する必要がある．

BQM の生成方法：

- QUBO 辞書からの生成：`dimod.BinaryQuadraticModel.from_qubo(Q)`
- 線形項・二次項の直接指定：`dimod.BinaryQuadraticModel(linear, quadratic, offset, vartype)`
- dwavebinarycsp 経由の生成：`dwavebinarycsp.stitch(csp)`

上記 1, 2 の方法では，QUBO 係数を事前に計算して dimod に渡す．3 の方法では，CSP 形式で記述した制約から dwavebinarycsp が内部で QUBO 係数を計算する．いずれの方法でも，最終的に dimod の BQM オブジェクトが生成される点は共通である．

本研究では，dwavebinarycsp を経由せずに QUBO 係数を直接計算し，2 の方法で BQM を生成する手法を整理する．これにより，dwavebinarycsp 内部で発生する $O(2^T)$ の列挙処理を回避できる．

2.8 おわりに

本章では，本研究の本手法および実験を理解するために必要な技術的基盤について述べた．表 2.2 に本章で説明した技術と，本研究における対応箇所を示す．

表 2.2　本章で説明した技術と本研究における対応

節	説明内容	本研究における対応
2.2	JSSP（問題設定・制約）	対象問題の定義
2.3	時間インデックスモデル	QUBO 定式化の基礎
2.4	量子アニーリング	最適化手法
2.5	QUBO/BQM	入力形式・制約表現
2.6	D-Wave システム	QPU 実行環境
2.7	Ocean SDK	従来手法（dwavebinarycsp）

次章では，JSSP への量子アニーリング適用に関する先行研究を概観し，本研究の位置づけを明確にする．

第 3 章

関連研究

3.1 はじめに

本章では，JSSP への量子アニーリング適用に関する先行研究を概観し，本研究の位置づけを明確にする．

JSSP への量子アニーリング適用は近年活発に研究されている分野である．Kurowski ら [9] はハイブリッド量子アニーリングヒューリスティックを提案し，Denkena ら [7] はフレキシブル JSSP への適用を検討した．また，Amaro ら [8] は変分量子アルゴリズム（VQA）による JSSP 解法を提案し，Zhang ら [10] は QUBO 専用ハードウェアでの性能評価を行った．

これらの研究の中で，QUBO 定式化の先駆的研究は Venturelli ら（2016）によるものである．彼らは JSSP の制約を QUBO 形式で表現する数式を提示したが，その数式から BQM オブジェクトを構築するための具体的な手続きは記載されていなかった．

その後，Carugno ら（2022）は Venturelli らの定式化を踏まえ，D-Wave Ocean SDK の dwavebinarycsp ライブラリを用いて BQM への変換を行った．すなわち，QUBO 定式化のロジックは Venturelli らの方法に従いつつ，実際の変換手続きは D-Wave のライブラリに委ねた形である．彼らは大規模インスタンスで BQM 生成に 8 時間以上を要すると報告したが，なぜ遅いのかについての解析は行っていなかった．

一方，Robinbux（2020）は dwavebinarycsp を使用せず，NumPy で QUBO 行列を直接構築する実装を GitHub で公開した．しかし，Venturelli らが提示した時間窓計算は実装されておらず，すべての時刻に対して変数を生成するため，大規模問題では変数数が増大するという問題がある．

本章では，Venturelli らおよび Carugno らの研究を詳細に分析し，本研究の位置づけ

を明確にする．これらの研究を選択した理由は，Venturelli らが JSSP の QUBO 定式化の基礎を提供し，Carugno らが D-Wave 実機での性能評価を最も詳細に行っているためである．

3.2 Venturelli らの研究

3.2.1 研究の概要

Venturelli ら [4] は，JSSP への量子アニーリング適用における先駆的研究である．彼らは NASA Ames Research Center の Quantum Artificial Intelligence Laboratory（QuAIL）と 1QB Information Technologies の共同研究として，JSSP の時間インデックス定式化に基づく QUBO 定式化を提案した．

彼らの定式化では，JSSP の制約を以下の 3 つのペナルティ項として表現する：

- h_1（順序制約）：ジョブ内の操作順序を強制する
- h_2（衝突回避制約）：同一機械上での操作の重複を禁止する
- h_3（一意性制約）：各操作が正確に 1 つの時刻に開始することを強制する

これらを組み合わせた QUBO ハミルトニアンは以下のように表される：

$$H_T(\bar{x}) = h_1(\bar{x}) + h_2(\bar{x}) + h_3(\bar{x}) \tag{3.1}$$

ここで，H_T はメイクスパン T に対するハミルトニアン（目的関数），$\bar{x}$ は全二値変数のベクトル，h_1, h_2, h_3 は各制約の違反を表すペナルティ関数である．各 h_i にはペナルティ係数（h_1 に α，h_2 に β，h_3 に γ）が含まれており，実行可能解では $H_T = 0$ となる[*1]．

また，彼らは変数削減手法として「Window Shaving」を提案した．これは各操作の有効な開始時刻範囲（時間窓）を事前計算し，その範囲外の変数を削除する手法である．

実験では，D-Wave Two（搭載 512 量子ビット，稼働 509 量子ビット）を用いて 3×3 から 6×6 のインスタンスを求解し，古典的ソルバーとの比較を行った．

以下では，各制約のペナルティ化と数式展開を説明する．

[*1] Venturelli らの原典 [4] では $H_T = \eta h_1 + \alpha h_2 + \beta h_3$ と外側係数を用いて表記しているが，本論文では係数を h_i の定義に含める形式に統一した．

3.2.2　一意性制約のペナルティ化（h_3）

第 2.3 節で定義した時間インデックスモデルの 3 つの制約（一意性制約，順序制約，衝突回避制約）を，ペナルティ化により QUBO ハミルトニアンに変換する．本節では一意性制約を扱う．

3.2.2.1　制約の定義

第 2.3 節で述べたように，時間インデックスモデルでは各操作 $O_{j,k}$ に対して複数の変数 $\{x_{j,k,0}, x_{j,k,1}, \ldots, x_{j,k,T-1}\}$ が存在する．このとき，$x_{j,k,t} = 1$ は「ジョブ j の操作 k が時刻 t に開始する」ことを意味し，$x_{j,k,t} = 0$ は「開始しない」ことを意味する．

各操作は正確に 1 つの時刻にのみ開始するため，以下の制約が必要である：

$$\sum_{t} x_{j,k,t} = 1 \quad \forall j, k \tag{3.2}$$

この制約は「T 個の時刻変数のうち正確に 1 つだけが 1 となる」という形式であり，**one-hot 制約**（または **exactly-one 制約**）と呼ばれる．

図 3.1 に，ジョブ，操作，時間インデックス変数の関係を示す．

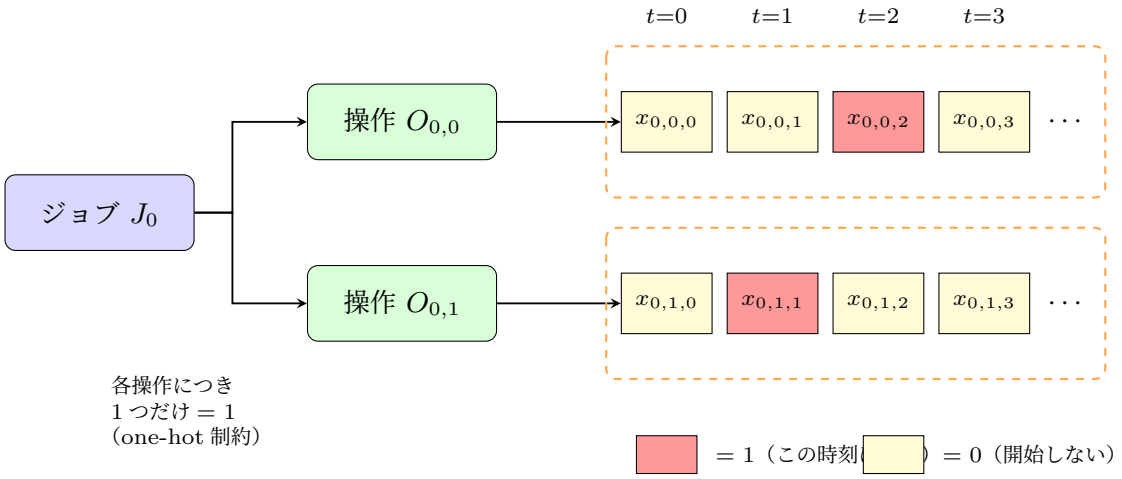


図 3.1　ジョブ，操作，時間インデックス変数の関係．各ジョブは複数の操作を持ち，各操作に対して時刻ごとの二値変数が存在する．各操作について，正確に 1 つの変数のみが 1 となる（one-hot 制約）．

3.2.2.2 ペナルティ化

この等式制約をペナルティ化により目的関数に組み込む．等式制約 $\sum_t x_{j,k,t} = 1$ を変形し，制約充足時のみゼロ，違反時は正のペナルティとしたいので，二乗することで：

$$\left(1 - \sum_t x_{j,k,t}\right)^2$$

制約を満たすとき（$\sum_t x_{j,k,t} = 1$）はペナルティ 0，違反時は正のペナルティとなる．

全操作について合計し，ペナルティ係数 γ を掛けると：

$$h_3 = \gamma \sum_{j,k} \left(1 - \sum_t x_{j,k,t}\right)^2 \tag{3.3}$$

3.2.2.3 QUBO 行列の構造

ペナルティ項 h_3 を展開すると，QUBO 行列の係数が得られる．式 (3.3) は全操作 (j,k) についての合計であるが，各操作は独立に同じ形式の制約を持つため，1 つの操作に注目して展開する．

操作 (j,k) に対する変数 $\{x_{j,k,0}, x_{j,k,1}, \ldots, x_{j,k,T-1}\}$ を簡単のため $\{x_0, x_1, \ldots, x_{T-1}\}$ と略記すると，その操作に対するペナルティ項は：

$$\begin{aligned}\gamma \left(1 - \sum_t x_t\right)^2 &= \gamma \left(1 - 2\sum_t x_t + \left(\sum_t x_t\right)^2\right) \\ &= \gamma - 2\gamma \sum_t x_t + \gamma \sum_t x_t^2 + 2\gamma \sum_{t<t'} x_t x_{t'}\end{aligned} \tag{3.4}$$

二値変数では $x_t^2 = x_t$ が成り立つため：

$$= \gamma - \gamma \sum_t x_t + 2\gamma \sum_{t<t'} x_t x_{t'} \tag{3.5}$$

定数項 γ を除くと，QUBO 行列の係数は以下となる：

- 対角成分（線形項）：$Q_{tt} = -\gamma$
- 非対角成分（二次項）：$Q_{tt'} = 2\gamma$（$t \neq t'$）

図 3.2 に，1 つの操作に対する一意性制約の QUBO 部分行列の構造を示す．

3.2.3 順序制約のペナルティ化（h_1）

本節では順序制約をペナルティ化により QUBO ハミルトニアンに変換する．

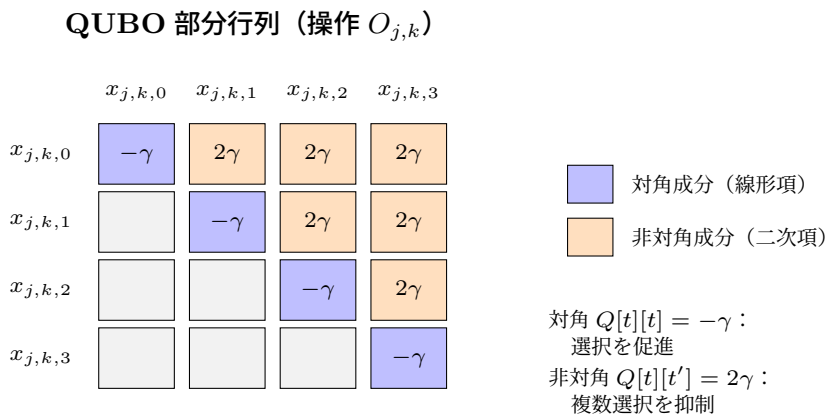


図 3.2　一意性制約における QUBO 行列の構造．1 つの操作 $O_{j,k}$ に対して，行・列ともに同じ操作の時刻変数 $\{x_{j,k,0}, x_{j,k,1}, \ldots\}$ を表す．対角成分（青）は「その時刻に開始する」ことへの促進，非対角成分（橙）は「複数の時刻で同時に開始する」ことへの抑制を表す．

3.2.3.1　制約の定義

第 2.2 節で定義した順序制約は，同一ジョブ内の連続する操作 $O_{j,k}$ と $O_{j,k+1}$ について，$O_{j,k}$ が完了してから $O_{j,k+1}$ が開始することを要求する：

$$S_{j,k} + p_{j,k} \leq S_{j,k+1} \tag{3.6}$$

ここで，$S_{j,k}$ は操作 $O_{j,k}$ の開始時刻，$p_{j,k}$ は処理時間である．

時間インデックスモデルでは，$x_{j,k,t} = 1$ のとき $S_{j,k} = t$（操作 $O_{j,k}$ が時刻 t に開始），$x_{j,k+1,t'} = 1$ のとき $S_{j,k+1} = t'$（操作 $O_{j,k+1}$ が時刻 t' に開始）となる．したがって，順序制約は $t + p_{j,k} \leq t'$ と表される．

図 3.3 に，順序制約の概念を示す．

3.2.3.2　ペナルティ化

順序制約の違反条件は $t + p_{j,k} > t'$ である．この違反が発生するのは，$x_{j,k,t} = 1$ かつ $x_{j,k+1,t'} = 1$（両方の操作がその時刻に開始する）の場合のみである．

違反時のみペナルティを加算し，非違反時はペナルティをゼロとしたいので，積 $x_{j,k,t} \cdot x_{j,k+1,t'}$ を用いる：

- 両方が 1 のとき（違反）：$1 \cdot 1 = 1$（ペナルティ加算）
- どちらかが 0 のとき（非違反）：0（ペナルティなし）

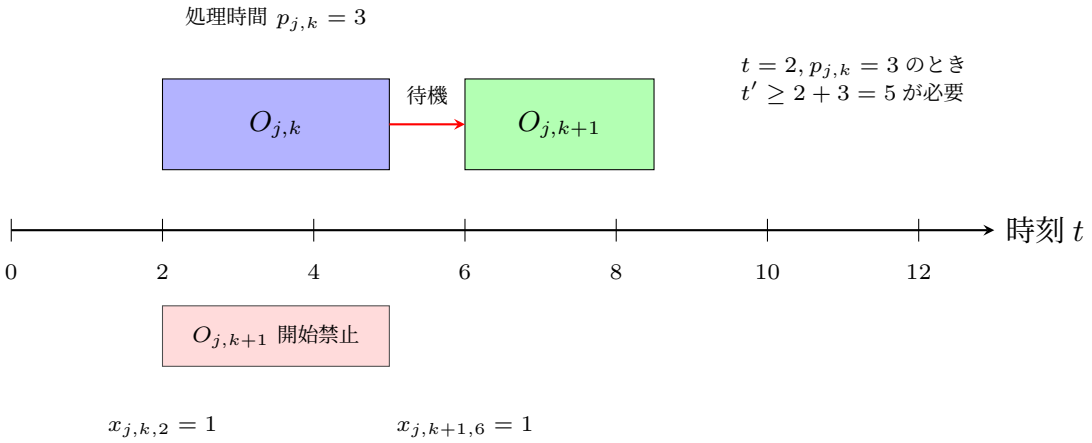


図 3.3　順序制約の概念．操作 $O_{j,k}$ が時刻 t に開始し処理時間 $p_{j,k}$ の場合，次の操作 $O_{j,k+1}$ は時刻 $t + p_{j,k}$ 以降にしか開始できない．

違反条件 $t' < t + p_{j,k}$ を満たすすべての (t, t') の組み合わせについてペナルティ項を合計し，全ジョブ・全操作ペアについて合計する．ペナルティ係数 α を掛けると：

$$h_1 = \alpha \sum_{j=0}^{n-1} \sum_{k=0}^{m-2} \sum_{t=0}^{T-1} \sum_{t'=0}^{\min(T-1,\, t+p_{j,k}-1)} x_{j,k,t} \cdot x_{j,k+1,t'} \tag{3.7}$$

ここで，k の範囲が $m-2$ までなのは，$O_{j,k}$ と $O_{j,k+1}$ のペアを考えるためである．t' の上限に min を用いるのは，$t + p_{j,k} - 1$ が $T - 1$ を超える場合に，存在しない変数を参照することを防ぐためである．

3.2.3.3　QUBO 行列の構造

式 (3.7) の各項 $\alpha \cdot x_{j,k,t} \cdot x_{j,k+1,t'}$ は，2 つの異なる変数の積である．したがって，QUBO 行列では非対角成分として表現される：

$$Q[x_{j,k,t}][x_{j,k+1,t'}] = \alpha \quad \text{（違反条件 } t' < t + p_{j,k} \text{ を満たす } (t, t') \text{ について）} \tag{3.8}$$

一意性制約と異なり，順序制約では対角成分（線形項）は発生せず，非対角成分（二次項）のみが設定される．

端的に言えば，順序制約は「前の操作 $O_{j,k}$ が終わる前に，次の操作 $O_{j,k+1}$ を開始してしまう時刻の組み合わせ」にペナルティを付与する．

図 3.4 に，処理時間 $p_{j,k} = 2$ の場合の順序制約における QUBO 部分行列の構造を示す．行は操作 $O_{j,k}$ の変数，列は操作 $O_{j,k+1}$ の変数に対応する．

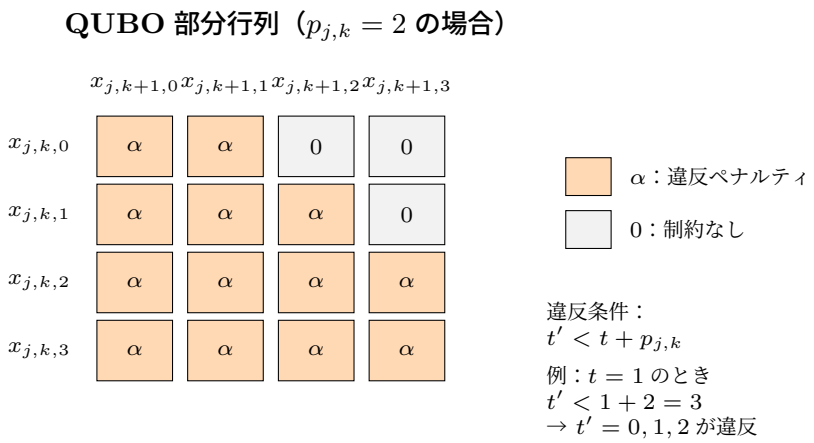


図 3.4　順序制約における QUBO 部分行列の構造（$p_{j,k} = 2$ の場合）．行は操作 $O_{j,k}$ の時刻変数，列は操作 $O_{j,k+1}$ の時刻変数を表す．違反条件 $t' < t + p_{j,k}$ を満たす (t, t') ペアに対して係数 α が設定される．

3.2.4　衝突回避制約のペナルティ化（h_2）

3.2.4.1　制約の定義

第 2.2 節で定義した衝突回避制約は，同一機械を使用する異なる操作が時間的に重複しないことを要求する．すなわち，機械 M_i を使う 2 つの操作 $O_{j,k}$ と $O_{j',k'}$（ただし $(j,k) \neq (j',k')$，$\mu_{j,k} = \mu_{j',k'} = M_i$）について，一方の処理中に他方が開始してはならない．

時間インデックスモデルでは，$x_{j,k,t} = 1$ のとき操作 $O_{j,k}$ は時刻 t に開始し，時刻 $t, t+1, \ldots, t+p_{j,k}-1$ の間（合計 $p_{j,k}$ 時刻）機械を占有する．このとき，操作 $O_{j',k'}$ が時刻 t' に開始する（$x_{j',k',t'} = 1$）場合，衝突が発生する条件は：

$$t \leq t' < t + p_{j,k} \tag{3.9}$$

これは「操作 $O_{j,k}$ の処理中に $O_{j',k'}$ が開始する」という片方向の衝突を表す．完全な衝突回避には逆方向（$O_{j',k'}$ の処理中に $O_{j,k}$ が開始）も禁止する必要があるが，すべての操作ペアについて両方向を検査するため，対称性により網羅される．

図 3.5 に，衝突回避制約の概念を示す．

3.2.4.2　ペナルティ化

2 つの操作 $O_{j,k}$ と $O_{j',k'}$ が同一機械上で時間的に重複する違反条件は，$(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t)$ である（双方向条件）．この違反が発生するのは，$x_{j,k,t} = 1$ かつ $x_{j',k',t'} = 1$（両方の操作がそれぞれの時刻に開始する）の場合のみである．

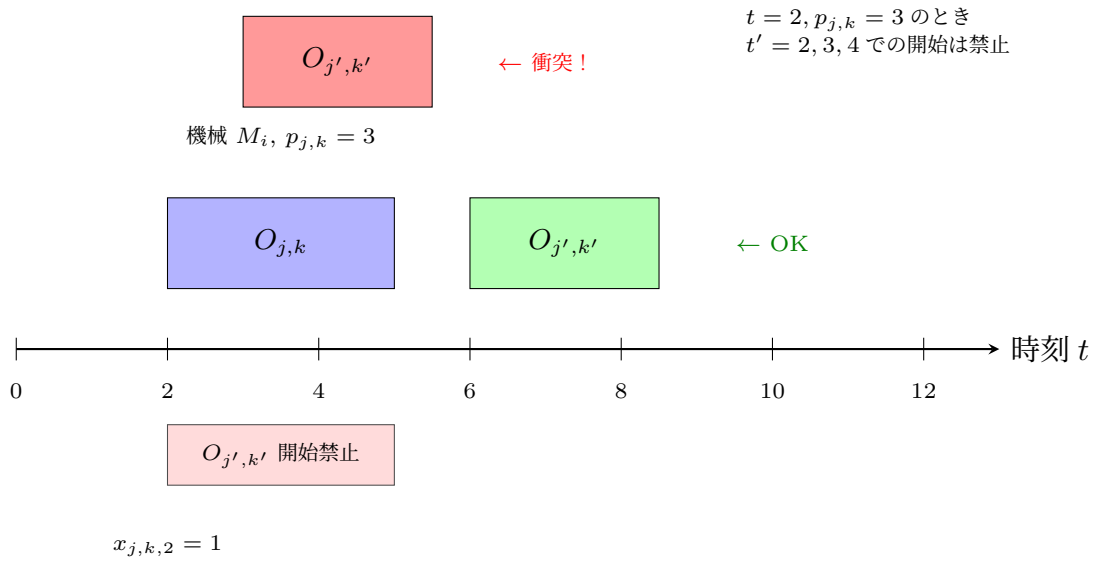


図 3.5　衝突回避制約の概念．操作 $O_{j,k}$ が時刻 t に開始し処理時間 $p_{j,k}$ の場合，同一機械を使う他の操作 $O_{j',k'}$ は時刻 t から $t + p_{j,k} - 1$ の間は開始できない．

違反時のみペナルティを加算し，非違反時はペナルティをゼロとしたいので，積 $x_{j,k,t} \cdot x_{j',k',t'}$ を用いる：

- 両方が 1 のとき（違反）：$1 \cdot 1 = 1$（ペナルティ加算）
- どちらかが 0 のとき（非違反）：0（ペナルティなし）

$\mathcal{O}_i = \{(j,k) : \mu_{j,k} = M_i\}$ を機械 M_i を使う操作の集合とする．双方向の違反条件 $(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t)$ を満たすすべての (t, t') の組み合わせについてペナルティ項を合計し，全機械について無順序の操作ペアを合計する．ペナルティ係数 β を掛けると：

$$h_2 = \beta \sum_{i=0}^{m-1} \sum_{\substack{(j,k),(j',k') \in \mathcal{O}_i \\ (j,k) < (j',k')}} \sum_{t,t'} \mathbf{1}[(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t)] \cdot x_{j,k,t} \cdot x_{j',k',t'} \quad (3.10)$$

ここで，外側の総和 $\sum_i$ は全機械について，内側の総和は同一機械を使う無順序の操作ペア（二重カウント防止のため $(j,k) < (j',k')$ に制限）についての合計である．$\mathbf{1}[\cdot]$ は条件を満たすとき 1，そうでないとき 0 を返す指示関数である．

3.2.4.3　QUBO 行列の構造

式 (3.10) の各項 $\beta \cdot x_{j,k,t} \cdot x_{j',k',t'}$ は，2 つの異なる変数の積である．したがって，QUBO 行列では非対角成分として表現される：

$$Q[x_{j,k,t}][x_{j',k',t'}] \mathrel{+}= \beta \quad \text{（双方向違反条件を満たす } (t, t') \text{ について）} \tag{3.11}$$

「+=」は，複数の操作ペアが同じ変数の組み合わせに寄与する場合に係数を累積することを表す．

順序制約と同様に，衝突回避制約では対角成分（線形項）は発生せず，非対角成分（二次項）のみが設定される．

端的に言えば，衝突回避制約は「同一機械を使う 2 つの操作が同時に（または重複して）処理される時刻の組み合わせ」にペナルティを付与する．

図 3.6 に，処理時間 $p_{j,k} = 2$ の場合の衝突回避制約における QUBO 部分行列の構造を示す．行は操作 $O_{j,k}$ の変数，列は同一機械の別の操作 $O_{j',k'}$ の変数に対応する．

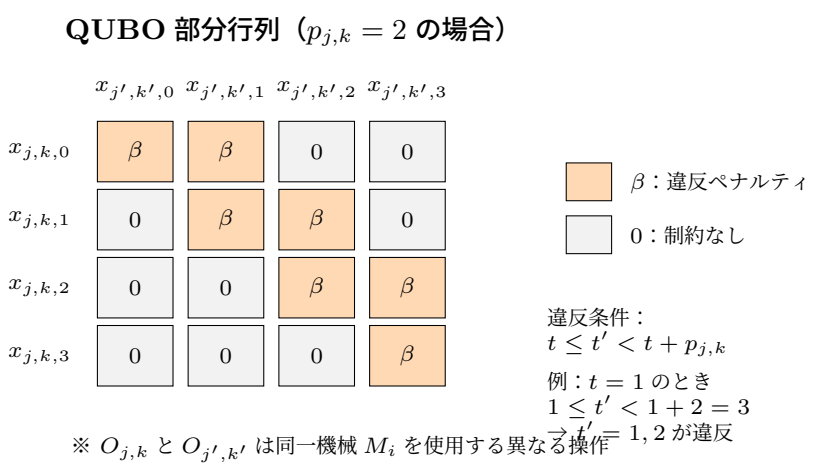


図 3.6　衝突回避制約における QUBO 部分行列の構造（$p_{j,k} = 2$ の場合）．行は操作 $O_{j,k}$ の時刻変数，列は同一機械を使う別の操作 $O_{j',k'}$ の時刻変数を表す．違反条件 $t \le t' < t + p_{j,k}$ を満たす (t, t') ペアに対して係数 β が設定される．

3.2.5　全体のハミルトニアン

3 つの制約項を統合した最終的なハミルトニアンは，式 (3.1) で定義した H_T と同一である：

$$H_T = h_1 + h_2 + h_3 \tag{3.12}$$

ここで，h_1（順序制約，係数 α），h_2（衝突回避制約，係数 β），h_3（一意性制約，係数 γ）は各制約のペナルティ関数である．

このハミルトニアンが 0 となる変数割り当ては，すべての制約を満たす実行可能なスケジュールに対応する．

3.2.6 時間窓計算

第 2.3 節で述べたように，時間窓は各操作が開始可能な時刻の範囲 $[ES_{j,k}, LS_{j,k}]$ を表す．Venturelli らは，この時間窓を計算する Shaving アルゴリズムを提案した．具体的には，Carlier ら [26] のアルゴリズムを応用した ICP（Iterated Carlier and Pinson）手続きにより，計算量 $O(n^2m^2T\log n)$ で時間窓を計算できることを示した（n: ジョブ数，m: 機械数）．

ただし，Venturelli らの論文では時間窓計算のアルゴリズムは示されているものの，計算された時間窓を使って変数を生成する手続き，および生成した変数にインデックスを割り当てて QUBO 係数に反映する具体的手続きは示されていない．この点については第 5 章で詳述する．

3.2.6.1 最早開始時刻の計算

最早開始時刻（Earliest Start, ES）は，先行する操作がすべて完了する時刻である．ジョブ J_j の操作 $O_{j,k}$ について，先行する操作 $O_{j,0}, O_{j,1}, \ldots, O_{j,k-1}$ がすべて完了するまで開始できないため：

$$ES_{j,k} = \sum_{k'<k} p_{j,k'} \tag{3.13}$$

すなわち，$ES_{j,k}$ は操作 $O_{j,0}$ から $O_{j,k-1}$ までの処理時間の合計である．特に，最初の操作 $O_{j,0}$ は $ES_{j,0} = 0$（時刻 0 から開始可能）となる．

図 3.7 に，最早開始時刻の計算例を示す．

3.2.6.2 最遅開始時刻の計算

最遅開始時刻（Latest Start, LS）は，timespan T 内にジョブのすべての後続操作を完了させるために，操作 $O_{j,k}$ を開始できる最後の時刻である．操作 $O_{j,k}$ を開始した後，自身の処理（$p_{j,k}$）および後続の操作 $O_{j,k+1}, \ldots, O_{j,m-1}$ の処理がすべて T 以内に完了する必要があるため：

$$LS_{j,k} = T - \sum_{k'\geq k} p_{j,k'} \tag{3.14}$$

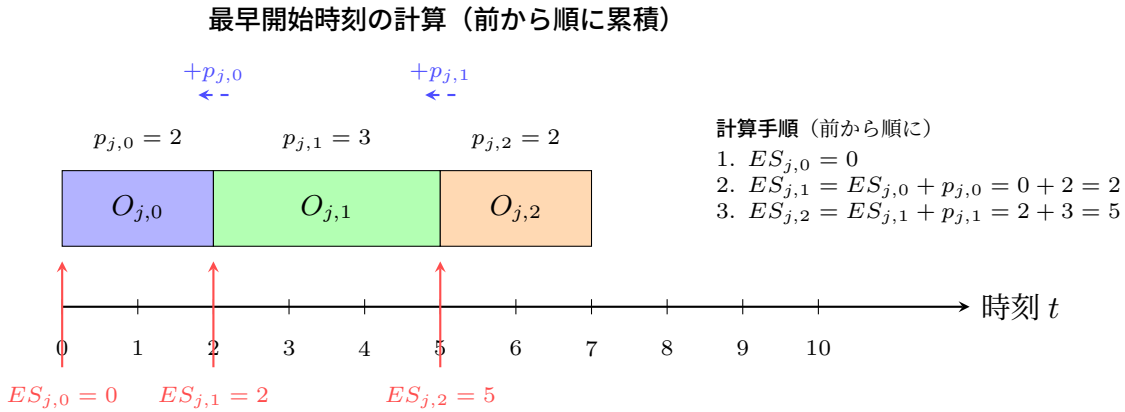


図 3.7　最早開始時刻（ES）の計算．各操作の最早開始時刻は，先行操作の処理時間を前から順に累積して求める．$ES_{j,k}=\sum_{k'<k} p_{j,k'}$

すなわち，$LS_{j,k}$ は timespan から操作 $O_{j,k}$ 以降のすべての処理時間を引いた値である．特に，最後の操作 $O_{j,m-1}$ は $LS_{j,m-1}=T-p_{j,m-1}$ となる．

図 3.8 に，最遅開始時刻の計算例を示す．最早開始時刻（図 3.7）が前から順に累積するのに対し，最遅開始時刻は後ろから逆算して求める．

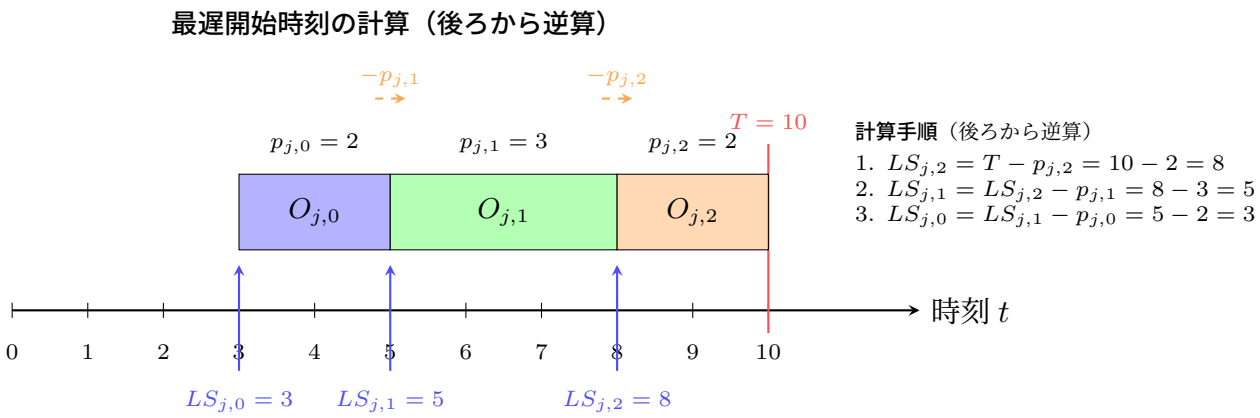


図 3.8　最遅開始時刻（LS）の計算．各操作の最遅開始時刻は，timespan から後続操作の処理時間を後ろから逆算して求める．$LS_{j,k}=T-\sum_{k'\geq k} p_{j,k'}$

3.2.6.3　変数削減効果

時間窓外（$t<ES_{j,k}$ または $t>LS_{j,k}$）の変数を生成しないことで，変数数を大幅に削減できる．

例えば，$n \times n$ インスタンスで $T = n + 1$ の場合：

- 時間窓なし：各操作に $T = n + 1$ 個の変数
- 時間窓あり：各操作に平均 2 個程度の変数

この削減により，QUBO 行列のサイズと制約数が大幅に減少する．

3.2.7 論文に記載されていない部分

Venturelli らの論文 [4] は，QUBO 数式（h_1, h_2, h_3）と時間窓計算を提示した．しかし，QUBO 数式を実際の QUBO 行列に変換する手続きは論文に記載されていない．

3.2.7.1 QUBO 行列と変数インデックスの関係

QUBO 行列 Q は，整数インデックス $i, j \in \{0, 1, 2, \ldots, N-1\}$ で参照される $N \times N$ の行列である（N は変数の総数）．一方，JSSP の変数 $x_{j,k,t}$ は 3 つのインデックス（ジョブ j，操作 k，時刻 t）を持つ．QUBO 行列を構築するには，この 3 次元インデックス (j, k, t) を 1 次元の整数インデックス i に変換する必要がある．

時間窓を考慮しない場合，この変換は単純な算術式で表現できる：

$$i = j \cdot m \cdot T + k \cdot T + t \tag{3.15}$$

ここで，m は機械数（＝各ジョブの操作数），T は timespan である．

しかし，時間窓計算により一部の変数が除外されると，この単純な変換式は使えなくなる．時間窓 $[ES_{j,k}, LS_{j,k}]$ の範囲外の時刻 t に対応する変数は生成されないため，変数インデックスが不連続になり，(j, k, t) から整数インデックスへの写像が複雑になる．

3.2.7.2 記述が十分でない手続き

以上を踏まえ，Venturelli らの論文で記述が十分でない手続きを整理すると：

(a) **変数インデックス割り当て**　時間窓で削減された変数 $x_{j,k,t}$ に対して，連続した整数インデックス $\{0, 1, 2, \ldots, N-1\}$ を割り当てる手順．どの (j, k, t) がどの整数インデックスに対応するかを管理する必要がある．

(b) **QUBO 係数への変換**　数式（h_1, h_2, h_3）の展開結果を，具体的な QUBO 行列 $Q[i][j]$ の係数に変換する手順．3 次元インデックス (j, k, t) を整数インデックスに変換しながら係数を設定する．

(c) **係数の累積**　複数の制約項が同一変数ペアに寄与する場合の係数累積処理．例えば，h_3 の展開で同じ (i, j) ペアに複数回係数が加算される場合がある．

(d) **BQM 構築**　QUBO 行列 Q から D-Wave Ocean SDK の BQM オブジェクトを構築する手順．

3.2.7.3　自明でない理由

これらの手続きが自明でない理由は，時間窓計算による変数削減にある．

時間窓を考慮しない場合，変数 $x_{j,k,t}$ から整数インデックスへの写像は単純な算術式で表現できる．しかし，時間窓により一部の変数が除外されると：

- 変数インデックスが不連続になる
- 制約式中の変数参照が複雑になる
- 係数累積の対象変数の特定が困難になる

本研究では，第 5 章でこれらの手続き (a)〜(d) を明示化する．

すなわち，Venturelli らは QUBO 定式化の理論を提示したが，QUBO 行列への具体的な展開手順は示していない．この点が，後続研究において追試の障壁となった．

3.3　Carugno らの研究

3.3.1　研究の概要

Carugno ら [5] は，Politecnico di Milano（ミラノ工科大学）による実証研究である．Venturelli らが 2016 年に JSSP の QUBO 定式化を提示してから約 6 年後の 2022 年，彼らはこの定式化の実用可能性を検証する研究を Scientific Reports 誌に発表した．

Venturelli らの研究当時，D-Wave Two は 512 量子ビットしか持たず，実験可能な JSSP インスタンスは極めて小規模に限られていた．Carugno らは，D-Wave Advantage（5640 量子ビット）および D-Wave 2000Q（2048 量子ビット）を用いて，より大規模なインスタンスでの性能評価を行った．

実験では，既知最適解インスタンス（$n = m$，処理時間すべて 1）と未知最適解インスタンス（4 ジョブ ×4 機械 ×4 操作，処理時間 1 または 2）の 2 種類を使用した．

以下では，Carugno らの理論的貢献と手続き，報告された計算コスト，および未解明の課題を整理する．

3.3.2 理論：QUBO 定式化

Carugno らは，Venturelli らの定式化（3.2 節）を基本的に踏襲している．3 つの制約（一意性・順序・衝突回避）とそのペナルティ化の方法は同一である．以下に軽微な差分を示す．

3.3.2.1 変数記法の違い

Venturelli らは 3 次元インデックス $x_{j,k,t}$（ジョブ j，操作 k，時刻 t）を使用するのに対し，Carugno らは 2 次元インデックス $x_{i,t}$（操作のグローバルインデックス i，時刻 t）を使用する．これは記法の簡略化であり，本質的な違いはない．

3.3.2.2 制約番号の対応

両者で制約の番号付けが異なる．表 3.1 に対応関係を示す．

表 3.1　Venturelli らと Carugno らの制約番号の対応

制約の内容	Venturelli	Carugno
一意性制約（one-hot）	h_3	h_1
衝突回避制約（share machine）	h_2	h_2
順序制約（precedence）	h_1	h_3

3.3.2.3 makespan 最小化項 $h_4(x)$ の追加

Carugno らは，Venturelli らの定式化に加えて，makespan 最小化のための線形ペナルティ項 $h_4(x)$ を導入している．この項は，各ジョブの最終操作の開始時刻が遅いほどペナルティを与える：

> “a linear penalty $h_4(x)$ is added to the variable corresponding to the last starting time of the last operation of each job” [5]

これにより，決定問題でありながら，より makespan の小さい解を優先的に探索できる．

3.3.2.4 Venturelli らの課題との関係

上記の差分は，いずれも**定式化の本質的な改善ではない**．変数記法の違いはインデックス管理の簡略化に過ぎず，$h_4(x)$ の追加も解の優先順位付けに過ぎない．

特に重要な点として，3.2.7.2 項で指摘した以下の手続きは，Carugno らの研究でも**未解決のままである**：

- (a) 変数インデックス割り当て
- (b) QUBO 係数への変換
- (c) 係数の累積
- (d) BQM 構築

Carugno らはこれらの手続きを自ら実装したのではなく，D-Wave Ocean SDK のライブラリ（dwavebinarycsp, penaltymodel）に委託している．この「ライブラリへの委託」が，後述するボトルネックの原因となっている．

3.3.3 手続き：ライブラリによる BQM 生成

Carugno らは，D-Wave Ocean SDK に含まれる `dwavebinarycsp` ライブラリを用いて BQM を生成した．`dwavebinarycsp` は，制約充足問題（CSP）を BQM に自動変換する機能を提供する．変換時には内部で `penaltymodel` ライブラリが呼び出され，ペナルティ係数が自動決定される．以下では，`dwavebinarycsp` を用いた手続きの流れを説明する．

3.3.3.1 全体フロー

Carugno らの手法は，以下の 3 ステップで構成される：

1. **Step 1：CSP 制約の定義**
   JSSP の制約（一意性・順序・衝突回避）を dwavebinarycsp ライブラリで CSP 形式に記述する．
2. **Step 2：BQM への変換**
   dwavebinarycsp の `stitch()` 関数により，CSP を BQM（Binary Quadratic Model）に変換する．この変換では，内部で penaltymodel ライブラリが呼び出され，ペナルティ係数が自動決定される．
3. **Step 3：量子アニーリング**
   生成された BQM を D-Wave QPU に送信し，サンプリングを行う．QPU への埋め込みには，minor-miner ライブラリを使用する．

図 3.9 に，Carugno らの手法の全体フローを示す．

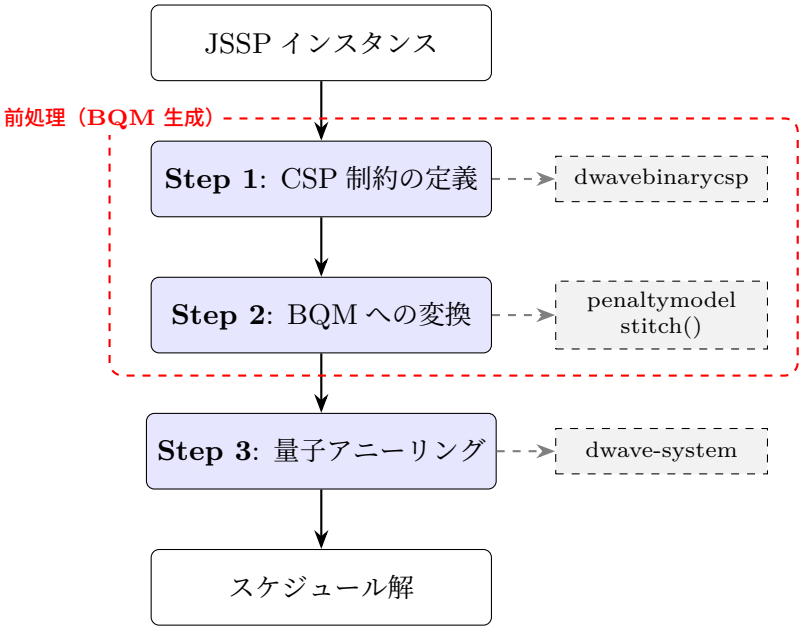


図 3.9　Carugno らの手法の全体フロー．JSSP インスタンスを CSP 形式で記述し，D-Wave のライブラリを用いて BQM に変換後，量子アニーリングで求解する．赤枠部分が前処理（BQM 生成）であり，ボトルネックとなる．

3.3.3.2　CSP から BQM への変換

dwavebinarycsp は，D-Wave Ocean SDK に含まれるライブラリであり，制約充足問題（CSP）を BQM に変換する機能を提供する．Carugno らはこのライブラリを使用して，JSSP の制約を BQM に変換した．

CSP から BQM への変換では，以下の処理が行われる：

1. `add_constraint()` メソッドで制約を登録
2. `stitch()` 関数でペナルティ係数を決定し，BQM を生成

3.3.3.3　制約の登録方法

`add_constraint()` メソッドには，以下の形式で制約を渡すことができる：

- **Callable（判定関数）**：制約を満たすかどうかを判定する関数
- **Iterable（有効構成）**：制約を満たす変数値の組み合わせのリスト

Carugno らの実装では，一意性制約を判定関数として定義している：

Listing 3.1　Carugno らの実装における一意性制約の定義

```
def sum_to_one(*args):
    return sum(args) == 1

for task_times in all_task_times:
    csp.add_constraint(sum_to_one, task_times)
```

ここで，`sum_to_one` は判定関数であり，引数の合計が 1 のときのみ `True` を返す．`all_task_times` は各操作に対応する時間変数のリストであり，`task_times` は 1 つの操作に対する全時刻の変数集合（例：$\{x_{0,0,0}, x_{0,0,1}, \ldots\}$）を表す．`csp.add_constraint()` により，「これらの変数のうち正確に 1 つだけが 1」という一意性制約が CSP に登録される．

このように，**判定関数**を渡す方法を採用している点が重要である．この選択が計算コストに影響することを，第 4 章で分析する．

3.3.4 報告された計算コスト

3.3.4.1 BQM 生成時間

Carugno らは，QUBO 生成の計算コストが高いことを報告している．表 3.2 に，インスタンスサイズと BQM 生成時間の関係を示す．

表 3.2 Carugno らが報告した BQM 生成の計算コスト

インスタンスサイズ	変数数の目安	BQM 生成時間
15×15 以下	～数百	数秒～数分
20×20 程度	～数千	数十分～数時間
26×26	～1 万	8 時間以上

Carugno らは，「penalty model プロセスが最も計算集約的」であり，特に**一意性制約（Carugno らの記法では h_1，本論文の記法では h_3）のペナルティ選択**に時間がかかると報告している．

3.3.4.2 penaltymodel ライブラリの問題

Carugno らは，penaltymodel ライブラリが大規模インスタンスで失敗する傾向があることも指摘している．具体的には：

- 処理時間のばらつきが大きいインスタンスで失敗しやすい

- timespan が大きいインスタンスで失敗しやすい
- 失敗時の原因特定が困難

3.3.5 未解明の課題

Carugno らの研究では，以下の点が未解明のまま残された：

1. **ボトルネックの原因**：「penalty model プロセス」がボトルネックであり，特に一意性制約のペナルティ選定に時間がかかると報告したが，dwavebinarycsp 内部の具体的な処理（from_func の列挙処理など）の計算量までは分析していない．
2. **計算量の分析**：BQM 生成時間がインスタンスサイズに対してどのような計算量で増加するか（多項式時間か指数時間か）を分析していない．
3. **改善策**：計算コストを削減するための具体的な改善策を提案していない．

これらの未解明点を明らかにすることが，本研究の動機の一つである．第 4 章では，dwavebinarycsp の内部動作を詳細に分析し，ボトルネックの原因を特定する．

3.3.6 Venturelli らの未解決課題との関係

3.2.7.2 項で指摘した手続き (a)〜(d) に対する Carugno らの対応を表 3.3 に示す．

表 3.3 Venturelli らの未解決課題に対する Carugno らの対応

手続き	内容	Carugno らの対応
(a)	変数インデックス割り当て	dwavebinarycsp に委託
(b)	QUBO 係数への変換	penaltymodel に委託
(c)	係数の累積	penaltymodel に委託
(d)	BQM 構築	stitch() 関数に委託

3.3.2 項で述べたように，Carugno らはこれらの手続きを自ら実装せず，ライブラリに委託している．この「ライブラリへの委託」がボトルネックの原因であり，詳細は第 4 章で分析する．

3.3.7 未解決課題の整理

表 3.4 に，Venturelli らと Carugno らの研究における未解決課題を整理する．

表 3.4　先行研究における未解決課題の整理（Venturelli と Carugno）

分類	課題	Venturelli	Carugno
手続き	(a) 変数インデックス割り当て	未記載	ライブラリ委託
	(b) QUBO 係数への変換	未記載	ライブラリ委託
	(c) 係数の累積	未記載	ライブラリ委託
	(d) BQM 構築	未記載	ライブラリ委託
性能分析	ボトルネックの特定	言及なし	報告のみ
	計算量の分析	なし	なし
	改善策の提案	なし	なし
時間窓	手続きとの連携	理論のみ	後段 pruning

この表から，以下の点が明らかである：

1. **手続きの不在**：Venturelli らは理論（QUBO 定式化）を提示したが，実装に必要な手続き (a)〜(d) を記載していない．Carugno らはこれらをライブラリに委託しており，手続きの詳細は依然として不明である．
2. **性能問題の未解決**：Carugno らは「penalty model プロセスがボトルネック」と報告したが，その原因の特定，計算量の分析，改善策の提案は行っていない．
3. **時間窓計算の未活用**：Venturelli らは時間窓計算の理論を提示したが，これを手続き (a)〜(d) とどう連携させるかは示していない．Carugno らは時間窓計算を QUBO 生成後の後段 pruning として実施しているが，変数生成段階での削減（前段生成）は行っていない．

本研究では，第 5 章でこれらの手続き (a)〜(d) を明示的に定義し，時間窓計算と連携した直接 BQM 構築手法を整理する．

3.4　Robinbux らの実装

3.4.1　研究の概要

Robinbux[6] は，GitHub で JSSP の量子アニーリング実装を公開したオープンソースプロジェクトである．この実装は Venturelli らの定式化に基づいており，D-Wave の量子アニーラおよびシミュレーテッドアニーラの両方に対応している．

学術論文としては発表されておらず，GitHub リポジトリのみで公開されている点で，Venturelli らや Carugno らの研究とは性質が異なる．しかし，本研究にとって重要な示唆を含むため，ここで取り上げる．

以下では，Robinbux の理論的立場と手続き，および課題を整理する．

3.4.2 理論：QUBO 定式化

Robinbux は，Venturelli らの定式化（3.2 節）をそのまま採用している．3 つの制約（一意性・順序・衝突回避）とペナルティ化の方法に差分はない．

3.4.3 手続き：直接 BQM 構築

Robinbux の実装の特徴は，Carugno らとは異なり，`dwavebinarycsp` を使用せず，NumPy で QUBO 行列を直接構築している点である．

具体的には，Venturelli らが提示した制約（一意性制約，順序制約，衝突回避制約）を，CSP 形式を経由せずに QUBO 係数として直接計算している．この方法は，Carugno らが使用した `dwavebinarycsp` による変換よりも効率的である可能性がある．

すなわち，Robinbux は 3.2.7.2 項で指摘した手続き (a)〜(d) を，ライブラリに委託せず自ら実装している．この点で，Carugno らとは異なるアプローチを採用している．

3.4.4 課題と限界

Robinbux の実装には以下の制限がある：

- Venturelli らが提案した時間窓計算（Window Shaving）が実装されていない
- すべての時刻に対して変数を生成するため，大規模問題では変数数が増大する
- 手法の詳細な説明や性能評価が行われていない

すなわち，Robinbux は直接 BQM 構築という方向性を示したが，時間窓計算による変数削減は行っておらず，手続き (a) における変数削減が欠如している．また，性能評価も行っていないため，Carugno らの手法と比較してどの程度効率的かは不明である．

3.4.5 未解決課題の整理（Robinbux を含む）

表 3.4 に Robinbux を加えた状況を表 3.5 に示す．

表 3.5　先行研究における未解決課題の整理（Robinbux を含む）

分類	課題	Venturelli	Carugno	Robinbux
手続き	(a) 変数インデックス割り当て	未記載	ライブラリ委託	実装（時間窓なし）
	(b) QUBO 係数への変換	未記載	ライブラリ委託	実装
	(c) 係数の累積	未記載	ライブラリ委託	実装
	(d) BQM 構築	未記載	ライブラリ委託	実装
性能分析	ボトルネックの特定	言及なし	報告のみ	なし
	計算量の分析	なし	なし	なし
	改善策の提案	なし	なし	なし
時間窓	手続きとの連携	理論のみ	後段 pruning	未実装

Robinbux は手続き (a)〜(d) を自ら実装しており，Carugno らのライブラリ委託とは異なるアプローチを採用している．しかし，時間窓計算との連携および性能分析は依然として未解決である．

3.5　本研究の位置づけ

本節では，3.2〜3.4 節で述べた先行研究の貢献と限界を集約し，本研究の位置づけを明確にする．

Venturelli らの貢献と限界：Venturelli らは，JSSP の QUBO 定式化（h_1, h_2, h_3）と時間窓計算の理論を提示した．しかし，QUBO 数式から BQM（QUBO 行列）を構築する具体的手続き，すなわち変数 $x_{j,k,t}$ を QUBO 行列のどの行・列に対応させ，どの成分に係数を格納するかは記載されていない．

Carugno らの貢献と限界：Carugno らは，Venturelli らの定式化を実装し，D-Wave 量子アニーラでの性能評価を行った．しかし，BQM 構築の手続き (a)〜(d) は dwavebinarycsp ライブラリに委託しており，手続きの詳細は不明のままである．また，BQM 生成に長時間を要する問題を報告したが，その原因の特定や改善策の提案は行っていない．

Robinbux の貢献と限界：Robinbux は，ライブラリに依存せず手続き (a)〜(d) を自ら実装し，直接 BQM 構築という方向性を示した．しかし，時間窓計算は実装されておらず，性能評価も行われていない．

以上より，先行研究では以下の点が未解明のままである：

1. **手続きの明示化**：Venturelli らの数式から BQM を構築する具体的手続き
2. **ボトルネックの原因**：Carugno らの手法においてなぜ BQM 生成に長時間を要するか
3. **時間窓との連携**：時間窓計算を含む完全な直接 BQM 構築手法

本研究では，これらの未解明点を解決する．具体的には，Carugno らの手法においてなぜ実行時間がかかるのかを調査してボトルネックを特定し（第 4 章），そのボトルネックを回避する直接 BQM 構築手法を整理する（第 5 章）．さらに，本手法の性能改善を実験により検証する（第 6 章）．

表 3.6 に，先行研究と本研究の比較を示す．

表 3.6　本研究の位置づけ

項目	Venturelli	Carugno	Robinbux	本研究
QUBO 数式	○	○	○	○
QUBO 行列構築手順	−	−	△	○
時間窓計算（ES/LS）	○	△ *	−	○
直接 BQM 構築	−	−	○	○
ボトルネック分析	−	−	−	○
高速化評価	−	−	−	○
研究の目的	理論の提示	解の質の評価	実装の公開	生成時間の改善

表中の△は，実装コードとして提供されているが論文等の文書として手順が明示的に説明されていないことを表す．*Carugno は Venturelli の時間窓計算（ES/LS）を使用せず，後段 pruning（不要変数の固定）を行う．

3.6　おわりに

本章では，JSSP への量子アニーリング適用に関する先行研究（Venturelli，Carugno，Robinbux）を概観し，未解決課題を表 3.5 に整理した．

表 3.7 に，本研究の構成を示す．

表より，先行研究では以下の項目が未解決である：

表 3.7　本研究の構成

項目	Venturelli	Carugno	Robinbux	本研究
QUBO 数式	○	○	○	–
QUBO 行列構築手順	–	–	△	第 5 章
時間窓計算（ES/LS）	○	△ *	–	–
直接 BQM 構築	–	–	○	第 5 章
ボトルネック分析	–	–	–	第 4 章
高速化評価	–	–	–	第 6 章

* 後段 pruning（不要変数の固定）のみ

- **ボトルネック分析**（第 4 章）：Carugno らの BQM 生成がなぜ長時間を要するか
- **QUBO 行列構築手順**（第 5 章）：QUBO 数式から QUBO 行列への具体的な展開手順
- **直接 BQM 構築**（第 5 章）：時間窓計算を含む完全な実装
- **高速化評価**（第 6 章）：本手法と従来手法の BQM 生成時間の比較

以降の章では，これらの未解決項目に順に取り組む.

第 4 章

ボトルネック分析

4.1 はじめに

本章の構成は以下の通りである．まず第 4.2 節で測定箇所を示し，Carugno らの手法（第 3.3 節）の全体フローと計測対象を明確にする．次に第 4.3 節で測定方法を述べ，計測環境（ハードウェア・ソフトウェア）と計測手順を示す．第 4.4 節で測定結果を示し，各処理ステップの時間内訳を明らかにする．第 4.5 節でボトルネック部分を詳細に調査し，dwavebinarycsp ライブラリの内部動作を分析する．最後に第 4.6 節で考察を行い，第 4.7 節でまとめる．

本章では，Carugno らの手法における BQM 生成のボトルネックを分析する．第 3.3 節で述べたように，Carugno らは 26×26 インスタンスで BQM 生成に 8 時間以上を要することを報告しているが，その原因は詳細に分析されていない．本章の目的は，Carugno らの手法を再現実装し，各処理ステップの処理時間を計測することで，ボトルネックの原因を特定することである．

本章で測定対象とするのは，Carugno らの手法による BQM 生成処理である．Carugno らの手法は，制約のペナルティ化（QUBO 定式化）として Venturelli らの定式化（第 3.2 節）に基づいており，BQM 生成には dwavebinarycsp ライブラリを使用している（第 3.3 節）．また，QUBO の定義，構成手順，および BQM への変換については第 2.5 節に整理した．

4.2　測定箇所

本節では，Carugno らの手法の全体フローを示し，処理時間の測定箇所を明らかにする．図 4.1 に，Carugno らの手法の全体フローを示す．処理は以下の 3 つのステップで構成される：

1. **Step 1（CSP 制約の定義）**：JSSP の制約（一意性・順序・衝突回避）を dwavebinarycsp ライブラリの `add_constraint` メソッドで CSP に登録する．
2. **Step 2（BQM への変換）**：`stitch` 関数により，CSP を BQM（Binary Quadratic Model）に変換する．この変換では，内部で penaltymodel ライブラリが呼び出される．
3. **Step 3（量子アニーリング）**：生成された BQM を D-Wave QPU に送信し，量子アニーリングで求解する．

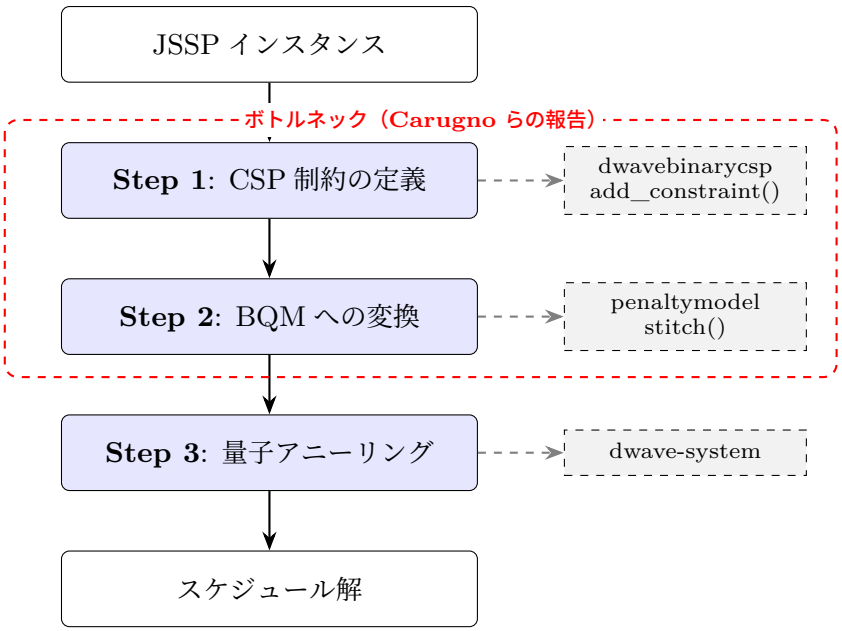


図 4.1　Carugno らの手法の全体フロー．JSSP インスタンスを CSP 形式で記述し，BQM に変換後，量子アニーリングで求解する．赤枠部分が Carugno らによりボトルネックと報告されている．

Carugno らは，Step 1 および Step 2 の処理（図 4.1 の赤枠部分）がボトルネックであると報告している．Step 3 は D-Wave QPU で実行されるため，CPU 側の処理時間には含まれない．

Step 1 を詳しく見ると，3 種類の制約が CSP オブジェクトに登録される．図 4.2 に，

Step 1 における制約登録の詳細フローを示す．各制約は以下のように登録される（詳細な登録手順は付録 A.2 を参照）：

- **(1) 一意性制約**：各操作に対する T 個の時刻変数を**関数渡し**で登録
- **(2) 順序制約**：連続する 2 操作の時刻ペアを**集合渡し**（2 変数）で登録
- **(3) 衝突回避制約**：同一機械を使用する 2 操作の時刻ペアを**集合渡し**（2 変数）で登録

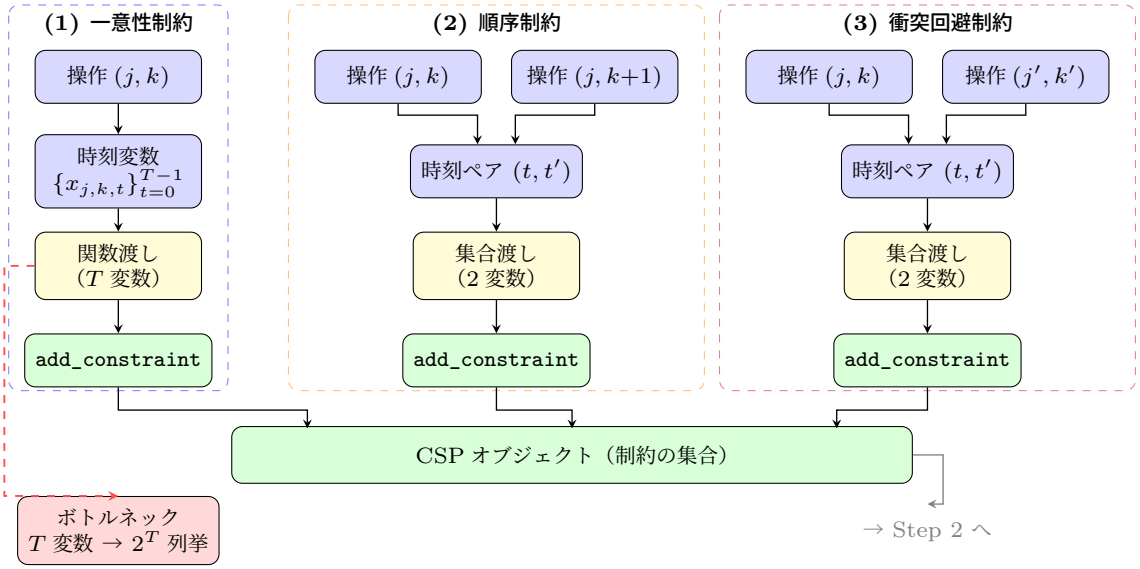


図 4.2　Step 1 の詳細フロー．(1) 一意性制約は T 個の変数を関数渡しで登録するため，後段で 2^T 列挙が発生しボトルネックとなる．(2)(3) は 2 変数の集合渡しで登録される．

図 4.2 に，Step 1 における制約登録の詳細フローを示す．ここで，一意性制約は関数渡しで登録されるのに対し，順序制約と衝突回避制約は集合渡しで登録される．この登録方法の違いがボトルネックに関係しており，詳細は付録 A.2 で説明する．

Carugno らは「一意性制約のペナルティ選択がボトルネック」と報告しているが，実際にどの処理がどの程度の時間を要するかは分析されていない．そこで本研究では，図 4.3 に示す 5 箇所の処理時間を測定する．

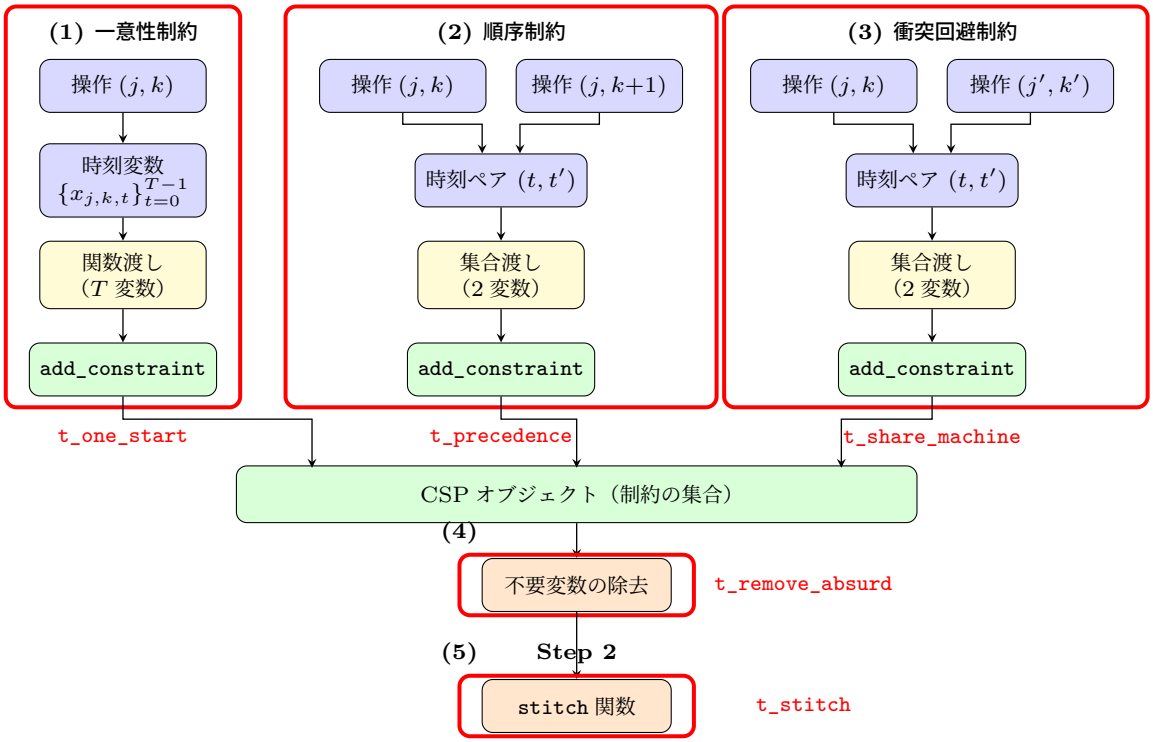


図 4.3　処理時間の測定箇所（図 4.2 を再掲）．赤枠で囲んだ 5 箇所の処理時間を測定する．

測定項目は以下の 5 つである：

1. `t_one_start`：一意性制約の CSP 登録時間（Step 1）
2. `t_precedence`：順序制約の CSP 登録時間（Step 1）
3. `t_share_machine`：衝突回避制約の CSP 登録時間（Step 1）
4. `t_remove_absurd`：不要変数の除去時間（Step 1，詳細は付録 A.2.5）
5. `t_stitch`：`stitch` 関数の実行時間（Step 2）

次節では，これらの項目の具体的な測定方法を述べる．

4.3　測定方法

4.3.1　計測環境

計測に使用した環境を表 4.1 および表 4.2 に示す．

表 4.1　ハードウェア環境

項目	仕様
CPU	Intel Core i7-10700 @ 2.90GHz
メモリ	32GB
OS	Windows 11

表 4.2　ソフトウェア環境

パッケージ	バージョン
Python	3.10.11
dwavebinarycsp*	0.3.1
penaltymodel*	1.3.0
dimod	0.12.21
dwave-system	1.34.0
dwave-ocean-sdk	9.1.0
numpy	2.2.6
scipy	1.15.3

*dwavebinarycsp および penaltymodel は Ocean SDK 9.x のメタパッケージに含まれないため別途インストール [32].

4.3.2　計測手順

4.3.2.1　計測コードの挿入位置

各処理ステップの前後で Python の `time.time()` を呼び出し，経過時間を記録した．リスト 4.1 に計測コードの挿入位置を示す．このコードは Carugno らの実装における `get_bqm()` メソッド内に挿入されている．

Listing 4.1　処理時間の計測コード（挿入位置）

```
import time

# --- 測定(1): t_one_start ---
start = time.time()
self._add_one_start_constraint()  # 一意性制約をCSPに登録
t_one_start = time.time() - start
```

```
7
8  # --- 測定(2): t_precedence ---
9  start = time.time()
10 self._add_precedence_constraint()  # 順序制約をCSPに登録
11 t_precedence = time.time() - start
12
13 # --- 測定(3): t_share_machine ---
14 start = time.time()
15 self._add_share_machine_constraint()  # 衝突回避制約をCSPに登録
16 t_share_machine = time.time() - start
17
18 # --- 測定(4): t_remove_absurd ---
19 start = time.time()
20 self._remove_absurd_times()  # 不要変数を0に固定
21 t_remove_absurd = time.time() - start
22
23 # --- 測定(5): t_stitcher ---
24 start = time.time()
25 bqm = dwavebinarycsp.stitch(self.csp)  # CSPをBQMに変換
26 t_stitcher = time.time() - start
```

表 4.3 に，各測定項目の詳細を示す．各測定箇所は図 4.3 の赤枠 (1)〜(5) に対応し，図 4.1 の Step 1 または Step 2 に属する．

表 4.3　測定項目の詳細

測定変数	測定対象関数	処理内容	図 4.1	図 4.3
t_one_start	_add_one_start_constraint	一意性制約の CSP 登録	Step 1	(1)
t_precedence	_add_precedence_constraint	順序制約の CSP 登録	Step 1	(2)
t_share_machine	_add_share_machine_constraint	衝突回避制約の CSP 登録	Step 1	(3)
t_remove_absurd	_remove_absurd_times	不要変数の 0 固定	Step 1	(4)
t_stitcher	dwavebinarycsp.stitch	CSP から BQM への変換	Step 2	(5)

4.3.2.2　問題インスタンス

計測に使用した問題インスタンスは，Carugno らの実装に含まれる $n \times n$ の正方インスタンスである．これは n 個のジョブ，各ジョブが n 個の操作を持ち，n 台の機械を使用する構成である．各操作の処理時間はすべて 1 とした．時間軸の上限 T は $T = n + 1$ に

設定した.

4.3.2.3 計測範囲

問題サイズ $n = 2, 3, 4, \ldots, 20$ の各サイズについて計測を行った．次節では，代表的なサイズ（$n = 5, 10, 15, 20$）の結果を示す.

4.4 測定結果

表 4.4 に，各問題サイズにおける処理時間の計測結果を示す．また，図 4.4 に問題サイズと処理時間の関係をグラフで示す.

表 4.4 処理時間の計測結果（単位：秒）

n	一意性制約	順序制約	衝突回避	不要変数除去	stitch	合計
2	0.00	0.00	0.00	0.00	0.02	0.02
3	0.00	0.00	0.00	0.00	0.05	0.05
4	0.00	0.01	0.01	0.00	0.09	0.11
5	0.00	0.02	0.03	0.01	0.15	0.21
6	0.01	0.04	0.06	0.01	0.24	0.36
7	0.01	0.08	0.12	0.02	0.34	0.57
8	0.02	0.12	0.20	0.04	0.47	0.85
9	0.04	0.21	0.37	0.06	0.65	1.33
10	0.08	0.28	0.52	0.10	0.74	1.72
11	0.20	0.41	0.79	0.20	1.04	2.64
12	0.48	0.58	1.11	0.21	1.18	3.56
13	1.21	0.83	1.54	0.28	1.55	5.41
14	2.57	1.08	2.12	0.44	1.82	8.03
15	6.17	1.44	2.66	0.62	2.11	13.00
16	14.26	2.27	3.58	1.16	3.48	24.75
17	33.33	2.58	4.36	1.04	3.05	44.36
18	79.20	2.98	5.58	1.29	3.47	92.52
19	183.12	3.67	7.05	1.63	4.50	199.97
20	432.69	4.80	9.21	2.17	4.61	453.48

また，表 4.5 に，QPU で得られた解の品質を示す．本実験では，Carugno らの実装で生成した BQM を D-Wave Advantage（Advantage_system2.1）で実行し，得られた解の makespan を評価した.

表 4.5　解の品質評価（QPU 実行結果）*

問題サイズ n	理論最適 makespan**	得られた makespan	最適解	best_energy
2	2	2	達成	0.0
3	3	3	達成	0.0
4	4	4	達成	0.0
5	5	5	達成	0.0
6	6	6	達成	0.0
7	7	7	達成	0.0
8	8	8	達成	0.0
9	9	9	達成	0.0
10	10	10	達成	0.0
11	11	11	達成	0.0
12	12	12	達成	0.0
13	13	13	達成	0.0
14	14	14	達成	0.0
15	15	15	達成	0.0
16	16	16	達成	0.0
17	17	17	達成	0.0
18	18	18	達成	0.0
19	19	19	達成	0.0
20	20	20	達成	0.0

* 正方形 JSSP（n ジョブ $\times n$ 機械，処理時間すべて 1）を使用.

** 本インスタンスでは理論最適 makespan $= n$．一般の JSSP では成り立たない.

正方形 JSSP（n ジョブ $\times n$ 機械，処理時間すべて 1）の理論最適 makespan は n である．すべての問題サイズにおいて，best_energy $= 0.0$（制約違反なし）かつ makespan $= n$（理論最適値）が得られており，Carugno らの実装が正しく動作していることが確認できた.

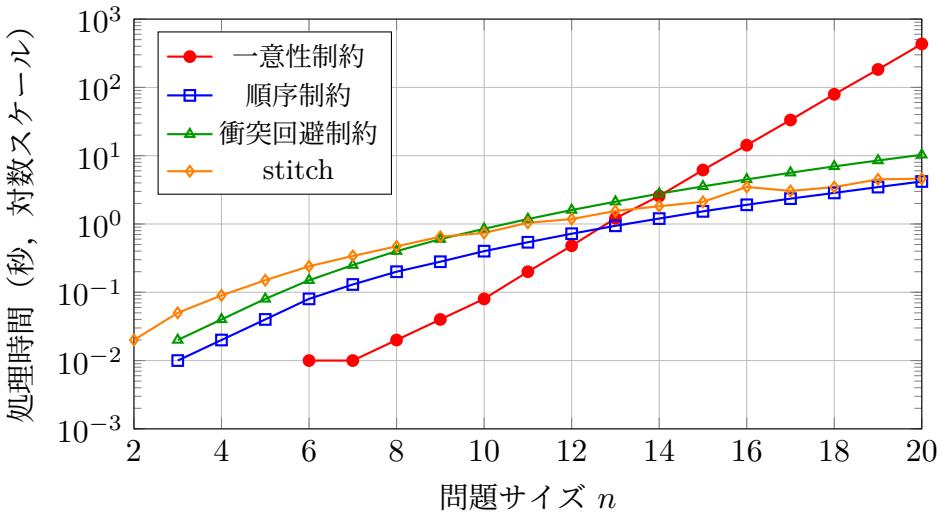


図 4.4　問題サイズと処理時間の関係（対数スケール）．一意性制約のみが指数的増加を示し，他の処理は緩やかな増加にとどまる．

図 4.5 に，縦軸を線形スケールにしたグラフを示す．線形スケールでは，一意性制約の処理時間が他を圧倒していることが視覚的に明確である．

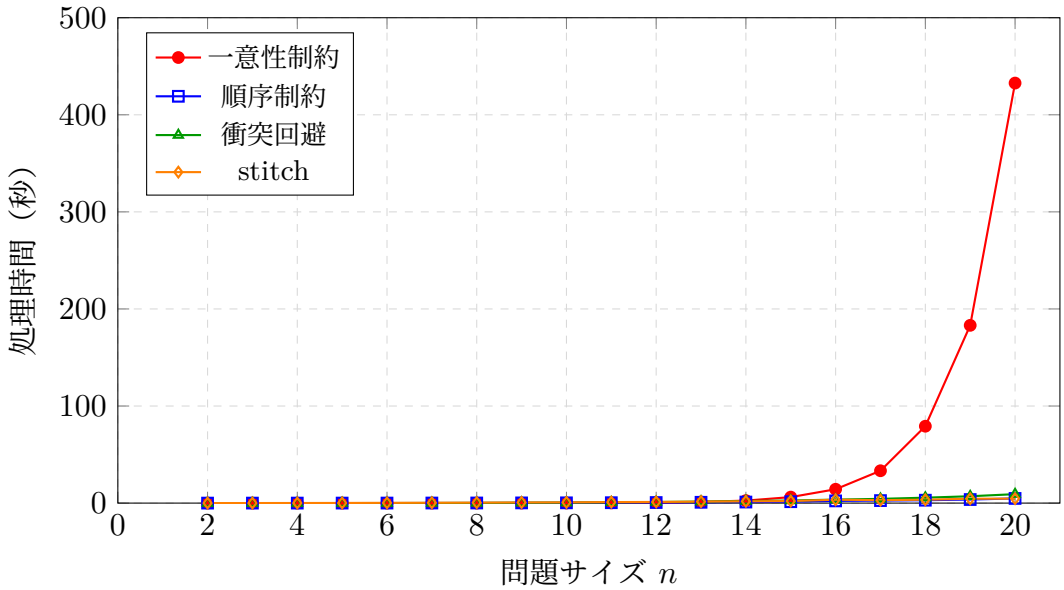


図 4.5　問題サイズと処理時間の関係（線形スケール）．一意性制約の処理時間が指数的に増加し，他の処理を圧倒している．

表 4.4 および図 4.4，図 4.5 から，以下の事実が読み取れる．

- 一意性制約（赤線）の処理時間は，問題サイズ n の増加に伴い指数的に増加している．$n = 10$ では 0.08 秒であったものが，$n = 20$ では 432.69 秒と，約 5,400 倍に増加している．
- 一方，順序制約（青線），衝突回避（緑線），stitch（橙線）の処理時間は，$n = 20$ でもそれぞれ 4.80 秒，9.21 秒，4.61 秒と，いずれも 10 秒未満である．
- $n = 15$ 以降，一意性制約の処理時間が他のすべての処理の合計を大きく上回り，グラフ上でも視覚的に明確な差が確認できる．

4.4.1 結果の考察

Carugno らの時点では，「一意性制約のペナルティ選択が最も計算集約的である」ことが報告されていたが，各処理ステップの所要時間は定量的に示されていなかった．本実験により，以下の新たな知見が得られた：

- 一意性制約の CSP 登録処理（Step 1）が全体の約 95% を占め，$n = 20$ で約 433 秒を要する
- Carugno らが「ボトルネック」と報告した Step 2（stitch 関数）は約 4.6 秒であり，全体の約 1% に過ぎない
- 一意性制約の処理時間は問題サイズに対して指数的に増加しており（$n = 15$ から $n = 20$ で約 70 倍），$O(2^k)$（k は制約内変数数，一意性制約では $k = T$）の計算量が示唆される

図 4.6 に $n = 20$ における処理時間の内訳を示す．

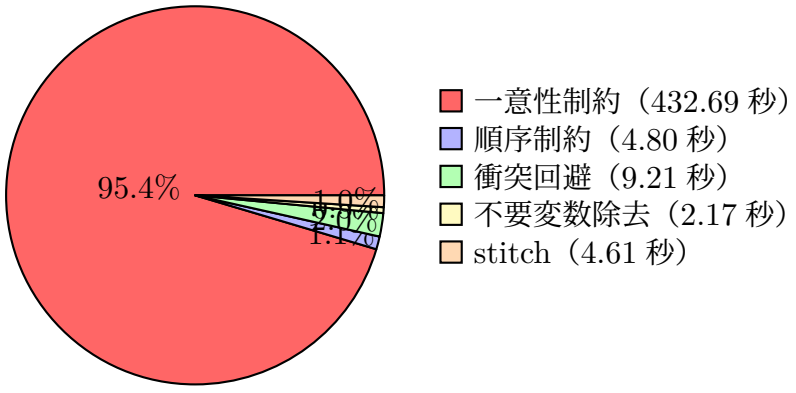


図 4.6　$n = 20$ における処理時間の内訳．一意性制約が 95.4% を占める．

以上の結果から，BQM 生成を高速化するためには，Step 2 ではなく **Step 1 の一意性**

制約登録処理を改善する必要があることが明らかになった．

なお，各制約の登録処理の詳細については付録 A.2.2〜A.2.5 を参照されたい．次節では，なぜ一意性制約の登録処理が指数的に増加するのかを，dwavebinarycsp の内部動作を分析することで明らかにする．

4.5 ボトルネック部分の調査

前節の計測結果から，一意性制約の CSP 登録処理が全体の処理時間の約 95% を占めていることが判明した．本節では，なぜ一意性制約だけがこれほど時間を要するのかを調査する．

端的に言えば, 一意性制約の登録処理では, dwavebinarycsp ライブラリの内部で $O(2^T)$ **の全列挙処理**が行われているためである．ここで T は時刻変数の数（timespan）であり，本実験の正方インスタンス（$n \times n$）では $T = n + 1$ に設定しているため，T と n はほぼ同値である．以下，この問題を詳細に説明する．

4.5.1 dwavebinarycsp の制約登録メカニズム

図 4.7 に，dwavebinarycsp による BQM 生成の全体フローを示す．

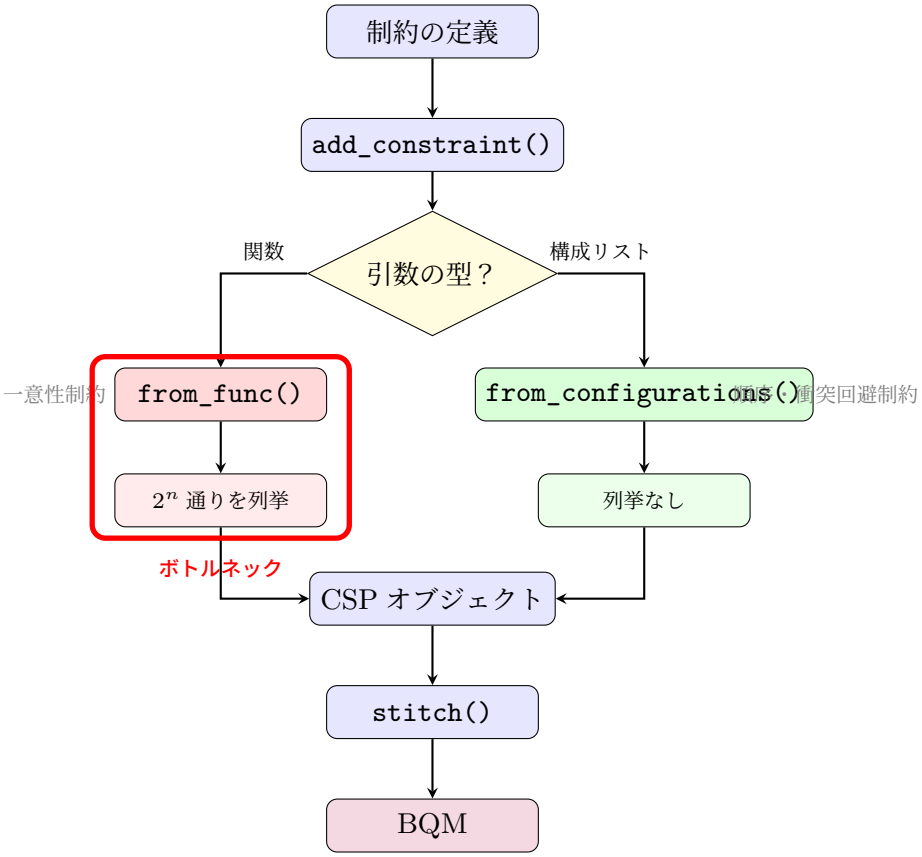


図 4.7　dwavebinarycsp による BQM 生成フロー．引数の型により内部処理が分岐する．`from_func`（赤）は 2^n 列挙を伴い，`from_configurations`（緑）は列挙なしで処理される．n は制約に関係する変数の数であり，一意性制約では $n = T$（時刻変数の数）となる．実装箇所：`from_func` は `dwavebinarycsp/core/constraint.py` の 130–183 行目（特に 179–181 行目で `itertools.product` による全列挙），`from_configurations` は同ファイル 186–232 行目，`stitch` は `dwavebinarycsp/compilers/stitcher.py` の 30–181 行目．

dwavebinarycsp では，制約を CSP に登録する際に `add_constraint` メソッドを使用する．このメソッドは，渡された引数の型に応じて異なる内部処理を実行する：

- **関数を渡した場合**：`Constraint.from_func()` を呼び出す
- **有効構成のリストを渡した場合**：`Constraint.from_configurations()` を呼び出す

Carugno らの実装では，一意性制約に対しては判定関数（`sum_to_one`）を渡し，順序制約・衝突回避制約に対しては有効構成のリストを渡している．この違いが処理時間の差を生む原因である．

4.5.2　from_func メソッドの問題点

`from_func` メソッドの実装を確認すると，図 4.8 に示す処理が行われている．

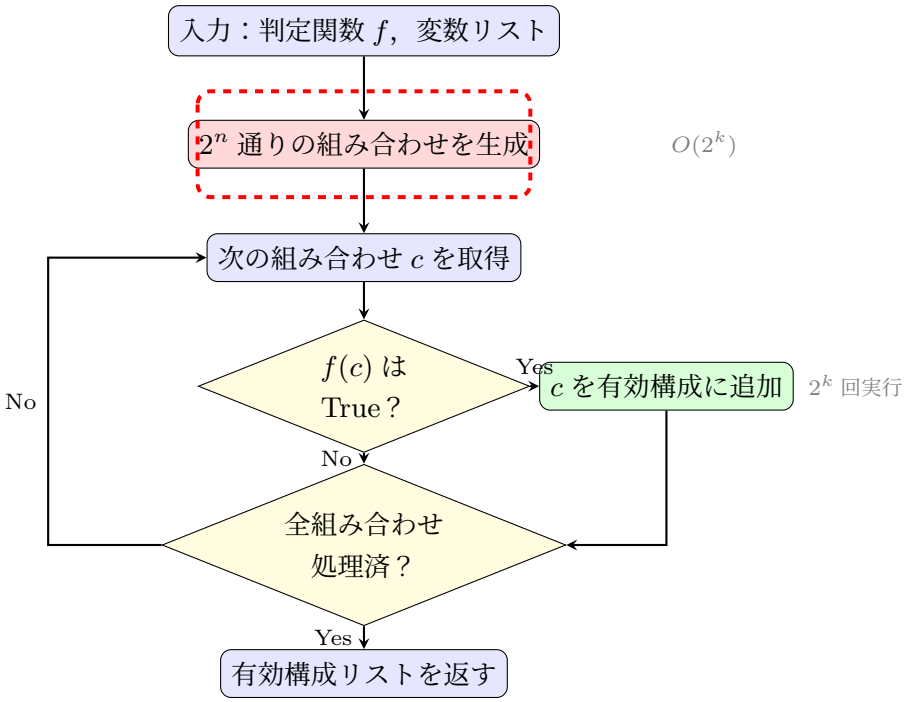


図 4.8　`from_func` メソッドの内部処理フロー．赤破線部分で 2^k 通りの全列挙が発生し（k は制約内変数数），各組み合わせに対して判定関数が呼び出される．

処理の手順は以下の通りである：

1. n 個の二値変数に対して 2^n 通りの全組み合わせを列挙
2. 各組み合わせについて判定関数を呼び出す
3. 判定が True となる組み合わせを有効構成として記録

リスト 4.9 に，`from_func` メソッドの実装コアを示す．

```
1  configurations = frozenset(
2      config
3      for config in itertools.product(      # <-- ここ！
4          vartype.value,                     #      2^n通りの
5          repeat=len(variables)              #      全列挙
6      )
7      if func(*config)
8  )
```

図 4.9　`from_func` メソッドの実装（`dwavebinarycsp/core/constraint.py` の 179–181 行目）．`itertools.product` が 2^k 通りの全組み合わせを生成する（k は制約内変数数）．

3 行目の `itertools.product(vartype.value, repeat=len(variables))` が 2^k 通りの全列挙を行う部分である．`vartype.value` は`{0, 1}`（二値），`len(variables)` は制約内の変数の数 k であるため，この呼び出しは $\{0,1\}^k$ のすべての要素を列挙する．

例えば $k = 3$ の場合，`itertools.product({0,1}, repeat=3)` は以下の $2^3 = 8$ 通りを生成する：

```
(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0),
(1,1,1)
```

問題点：`from_func` は $O(2^k)$ の計算量を持つ（k は制約内変数数）．一意性制約では $k = T$（時刻変数の数）であるため，$T = 20$ の場合，$2^{20} \approx 100$ 万通りの列挙が発生する．これが一意性制約の処理時間が支配的である原因である可能性が高い．

4.6　考察

本節では，ボトルネック分析の結果を踏まえた考察と，dwavebinarycsp の現在の状況について述べる．

4.6.1　Ocean SDK における非推奨化

dwavebinarycsp は，D-Wave Ocean SDK 8.1（2024 年 12 月リリース）において非推奨（deprecated）となり，Ocean SDK 9.0（2025 年 9 月リリース）において SDK メタパッケージから削除された [32, 39, 34]．なお，「削除」とは dwave-ocean-sdk メタパッケージの依存関係から外れたことを意味し，dwavebinarycsp 自体は別途インストール可

能であるが，保守されない非推奨状態である．同様に，dwavebinarycsp が内部で使用する penaltymodel も非推奨となり，Ocean 10 で削除される予定である [34].

したがって，**Ocean SDK 9.x 系（9.0 以降 [33]）では dwavebinarycsp は標準構成から削除されており，先行研究の実装をそのまま実行するには追加のインストールと依存関係の固定が必要である**．本研究では，Carugno らの手法を再現するために Ocean SDK 9.1.0 環境において dwavebinarycsp を別途インストールし，依存関係を固定して計測を行った（第 4 章，表 4.2 参照）.

この非推奨化の背景には，本章で分析したような性能上の問題があると考えられる．`Constraint.from_func()` による $O(2^k)$ 列挙（k は制約内変数数）は，変数数の増加に対してスケールしない.

4.6.2 代替アプローチ

dwavebinarycsp の非推奨化に伴い，以下のアプローチが考えられる：

1. **CQM（Constrained Quadratic Model）**：制約をペナルティ関数に変換せず，ネイティブに扱えるモデル形式．`LeapHybridCQMSampler` を通じてハイブリッドソルバーで実行する．D-Wave 公式ドキュメントでは制約付き問題に対してこの方法を推奨している [32].
2. **dimod による直接的な BQM 構築**：dwavebinarycsp を経由せず，BQM オブジェクト（`dimod.BinaryQuadraticModel`）を直接構築する方法.

ただし，CQM はハイブリッドソルバーでのみ実行可能であり，QPU で直接実行する場合は BQM を構築する必要がある．したがって，QPU を用いて JSSP を解くためには，第 3.2 節で述べた Venturelli らの QUBO 定式化を BQM 化する手続きが必要となる.

本研究の本手法（第 5 章）は，上記 2 の「dimod による直接的な BQM 構築」に相当する.

4.6.3 本研究の位置づけ

当初の状況：Carugno らの研究（2022 年）は，当時標準的であった dwavebinarycsp を使用して BQM を生成していた．dwavebinarycsp は制約を記述するだけで BQM を自動生成できる便利なライブラリであった.

本章の実験で判明したこと：dwavebinarycsp の内部で使用される `from_func` メソッドが $O(2^k)$（k は制約内変数数）の全列挙処理を行っており，これが BQM 生成時間の支

配的要因である可能性が高いことが判明した．

改善の可否：この問題は dwavebinarycsp の設計に起因するものであり，ユーザー側での改善は困難である．`from_func` は内部で全列挙を行う設計となっており，これを回避するにはライブラリ自体の修正が必要となる．実際，D-Wave は dwavebinarycsp を非推奨とし，最終的に削除する判断を下した．

代替手段：dwavebinarycsp を使用せず，BQM を直接構築する方法がある．具体的には，Venturelli らの QUBO 定式化に基づき，QUBO 係数を直接計算して BQM オブジェクトを構築する．この方法では $O(2^k)$ の列挙処理が発生しない．

現状の課題：QPU を利用するためには BQM が必要であるが, JSSP において Venturelli らの QUBO 定式化を直接 BQM 化した実装事例は限定的であり，具体的な手続きが明示された文献は少ない．

本研究の方針：そこで次章では，Venturelli らの QUBO 定式化に基づく直接 BQM 構築手法を整理し，その具体的な手続きを示す．

4.7 まとめ

Carugno らは「BQM 生成に長時間を要する」と報告していたが，その原因について詳細な検証はなされていなかった．本章では，Carugno らの実装を用いて処理時間を計測し，ボトルネックの原因を調査した．

問題サイズ $n = 2$ から $n = 20$ まで計測した結果，一意性制約の CSP 登録処理が全体の処理時間の約 95% を占めることが判明した．Carugno らは「一意性制約のペナルティ選択に時間がかかる」と報告していたが，具体的にどの処理がどの程度の時間を要するかは示されていなかった．本章の計測により，ボトルネックが Step 1 の一意性制約登録処理であることを具体的に特定した．

さらに，dwavebinarycsp の内部動作を調査した結果，`from_func` メソッドが $O(2^k)$（k は制約内変数数）の全列挙処理を行っており，これがボトルネックの原因である可能性が高いことを示した．しかし，この問題は dwavebinarycsp の内部設計に起因するものであり，ユーザー側での改善は困難である．実際，D-Wave は dwavebinarycsp を非推奨とし，Ocean SDK 9.0（2025 年 9 月）で削除した．

現在は dwavebinarycsp が標準構成から削除されているため，QPU で問題を解くには CQM（ハイブリッドソルバー経由）または BQM（QPU 直接実行）の 2 つの方法がある．ただし，QPU 単体で実行するには BQM を構築する必要がある．しかし，第 3.2 節で述べた Venturelli らの QUBO 定式化から直接 BQM を構築した実装事例は限定的であ

り，具体的な手続きが明示された文献は少ない．

したがって，dwavebinarycsp を経由せずに BQM を直接構築する手続きが必要である．次章では，Venturelli らの QUBO 定式化に基づく直接 BQM 構築手法を整理する．

第 5 章

直接 BQM 構築

5.1 はじめに

5.1.1 Carugno らの手法の課題

本研究では，JSSP への量子アニーリング適用において Carugno らの手法 [5] をベースに検討を進めてきた．Carugno らは，Venturelli らの QUBO 定式化 [4] を D-Wave Ocean SDK の dwavebinarycsp ライブラリを用いて実装し，実機での性能評価を行った．

しかし，第 4 章での Carugno らの手法の再現実験および時間測定から，BQM 生成時間が問題サイズに対して指数関数的に増加することが判明した．20×20 インスタンスでは約 7.5 分を要し，この指数関数的増加の傾向が続けば，より大規模な問題では BQM 生成に過大な時間を要するようになり，大規模問題での性能検証が困難になる．

時間測定により特定したボトルネック部分の調査から，この原因は dwavebinarycsp ライブラリ内部の `Constraint.from_func()` メソッドが $O(2^k)$（k は制約内変数数）の列挙処理を行うことにあると明らかになった．これはライブラリ内部の問題であるため改修が難しく，dwavebinarycsp は現在 D-Wave によって非推奨（deprecated）とされ，リポジトリもアーカイブされている [15].

現在 D-Wave が示す代替方法としては，Constrained Quadratic Model（CQM）をハイブリッドソルバーで解く方法がある．CQM では制約を直接記述できるため，dwavebinarycsp のような変換処理が不要である．しかし，ハイブリッドソルバーは CPU と QPU を組み合わせた処理であり，QPU のみを使用することができない．そのため，QPU 単体の性能評価には適さない．

したがって，QPU の性能を正確に評価するためには，dwavebinarycsp を経由せずに BQM を直接構築する必要がある．

5.1.2 BQM 直接構築に必要な手続き

dwavebinarycsp を使用せずに BQM を直接構築するには，Venturelli らの QUBO 定式化（理論）から BQM（データ構造）への変換を自ら実装する必要がある．この変換には，表 5.1 に示す 4 つの手続きが必要である．

表 5.1　BQM 構築に必要な手続き

記号	手続き名	内容
(a)	変数インデックス割り当て	問題変数 $x_{j,k,t}$ を QUBO 行列の行・列インデックス $i \in \{0, 1, \ldots, N-1\}$ に対応させる
(b)	QUBO 係数への変換	ペナルティ関数 h_1, h_2, h_3 の各項を，QUBO 行列の成分 Q_{ij} として表現する
(c)	係数の累積	複数の制約から同一の (i, j) 成分に寄与する係数を加算する
(d)	BQM オブジェクト構築	QUBO 行列 Q から D-Wave SDK で使用可能な BQM オブジェクトを生成する

5.1.3 Venturelli らの論文における手続きに関する記述

これらの手続きの前例として，まず Venturelli ら [4] の論文を確認する．Venturelli らは，JSSP の QUBO 定式化（h_1, h_2, h_3）と時間窓計算（Window Shaving）の理論を提示した重要な先行研究である．

しかし，Venturelli らの論文は理論レベルでの定式化に留まっており，上記の手続き (a)〜(d) は記載されていない：

- (a) 変数 $x_{j,k,t}$ を QUBO 行列のインデックスにどう対応させるか — 記載なし
- (b) ペナルティ関数から QUBO 係数を導出する具体的手順 — 記載なし
- (c) 複数制約からの係数をどう累積するか — 記載なし
- (d) BQM オブジェクトをどう構築するか — 記載なし

すなわち，Venturelli らは「何を最小化するか（QUBO 定式化）」は示したが，「それをどう実装するか（手続き）」は示していない．

これらの手続きは自明ではない．例えば，手続き (a) において，時間窓を考慮しない場合は $\mathrm{index}(j, k, t) = j \cdot m \cdot T + k \cdot T + t$ のような単純な算術式でインデックスを計算で

きる．しかし，Venturelli らの時間窓計算を適用すると一部の変数が除外されるため，この単純な対応関係が崩れる．除外された変数をどう扱い，残った変数にどう連番を割り当てるかは，実装者が独自に設計する必要がある．

また，手続き (b) においても，ペナルティ関数の展開では二値変数の性質（$x^2 = x$）を考慮した係数計算が必要であり，線形項・二次項・定数項（offset）を正しく分離する必要がある．

5.1.4 Carugno らの論文における手続きに関する記述

Carugno ら [5] は，Venturelli らの QUBO 定式化を引用し，D-Wave Ocean SDK の dwavebinarycsp ライブラリを用いて BQM を生成している．

Carugno らの論文では，手続き (a)〜(d) について以下の状況である：

- (a) 変数インデックス割り当て — dwavebinarycsp に委託（論文に記載なし）
- (b) QUBO 係数への変換 — dwavebinarycsp に委託（論文に記載なし）
- (c) 係数の累積 — dwavebinarycsp に委託（論文に記載なし）
- (d) BQM オブジェクト構築 — dwavebinarycsp に委託（論文に記載なし）

すなわち，Carugno らは dwavebinarycsp ライブラリを使用することで手続き (a)〜(d) を実装する必要がなく，これらの手順は論文に記載されていない．なお，時間窓計算については，Carugno らの実装では QUBO 生成後に不適切な変数を 0 固定する後段 pruning（`remove_absurd`）として反映されているが，変数生成段階で探索空間を縮小する形では反映されていない．そのため，`Constraint.from_func()` の列挙コスト抑制には寄与しない．

第 4 章で述べたように，dwavebinarycsp は内部で $O(2^k)$（k は制約内変数数）の列挙処理を行うため，問題サイズの増加に対して BQM 生成時間が指数関数的に増加する．したがって，Carugno らの手法はスケーラビリティに課題がある．

5.1.5 Robinbux の実装について

Venturelli らが示していない手続き (a)〜(d) について，Robinbux[6] が dwavebinarycsp を使用せずに NumPy で QUBO 行列を直接構築する実装を公開している．

しかし，Robinbux の実装コードを精査したところ，以下の問題が確認された．

■（時間窓計算の未反映）Venturelli らが提案した時間窓計算が実装されていない．時間上限 T がハードコードされており（$T = 9$ 固定），すべての時刻 $t \in [0, T-1]$ に

対して変数を生成している．そのため，時間窓による変数削減が行われていない．

■（衝突回避制約 h_2 の判定条件）同一機械を使用する 2 つの操作のオーバーラップ条件について，Venturelli らの定式化では双方向の判定が必要である：

$$(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t) \tag{5.1}$$

しかし，Robinbux の実装では一方向のみをチェックしている：

$$0 \leq t' - t < p_{j,k} \tag{5.2}$$

この条件は $t' \geq t$ の場合しか検出できず，$t' < t$ の場合のオーバーラップを見逃す．

■（一意性制約 h_3 の係数）一意性制約の展開において，線形項の係数が誤っている．正しい展開では線形項の係数は $-\gamma$ であるべきところ，Robinbux の実装では異なる係数が使用されており，offset 項も欠落している．

■（ペナルティ係数の設定方法）ペナルティ係数の設定が試行錯誤的である．制約違反が検出されると係数を固定値（+1）で増加させ，成功回数が閾値に達するまで繰り返す．この手法は理論的根拠に欠け，再現性も低い．

5.1.6 先行研究の状況整理

表 5.2 に，先行研究における手続きの状況を整理する．ここでは，各手続きについて「理論」と「アルゴリズム」の 2 つの観点から評価する．

- 理論：数式レベルでの定式化の有無
- アルゴリズム：疑似コード等の実行可能な手順の有無

表 5.2　先行研究における手続きの状況（理論／アルゴリズム）

#	手続き	Venturelli	Carugno	Robinbux	本研究
1	JSSP インスタンスの読み込み	○/×	○/×	—/○	×/○
2	時間窓計算（ES, LS）	○/○	×/×	×/×	×/○
3	時間窓を使った変数生成	△[1]/×	×/×	×/×	×/○
4	h_3 制約（一意性）の係数	○/×	○[2]/×	×[3]/×	×/○
5	h_1 制約（順序）の係数	○/×	○[2]/×	—/○	×/○
6	h_2 制約（衝突回避）双方向	○/×	○[2]/×	×[4]/×	×/○
7	ペナルティ係数の設定	△[5]/×	○/×	×[6]/×	×/○
8	BQM 直接構築	△[7]/×	×[8]/×	—/○	×/○

各セルは「理論/アルゴリズム」の形式．○：明示的に記載，△：概念的言及，×：未記載，—：対象外
[1]「時間窓外の変数は除外可能」と概念的に言及
[2]Venturelli を引用
[3] 線形項の係数誤り，offset 項の欠落
[4] 一方向チェックのみ（双方向判定が必要）
[5]「A, B は十分大きく」と概念的に言及
[6] 試行錯誤による調整（$+1$ ずつ増加）
[7]「Q 行列を構築」と概念的に言及
[8]penaltymodel ライブラリに委託

表 5.2 から，先行研究では理論と実装の間にギャップがあることがわかる．Venturelli らは理論を提示したがアルゴリズムは時間窓計算のみ，Carugno らはライブラリに委託，Robinbux は理論的根拠なく一部に誤りがある．本研究は，このギャップを埋め，全手続きについて実装可能なアルゴリズムを提示する．

本章では，BQM 直接構築の全手続きを 6 つのアルゴリズムとして整理する：

- **Algorithm 1**：BQM 直接構築の全体フロー（本節）
- **Algorithm 3**：時間窓に基づく変数生成（第 5.3 節）
- **Algorithm 5**：h_2 制約の双方向判定による係数計算（第 5.5 節）
- **Algorithm 4**：h_3 制約の正確な係数導出（第 5.4 節）
- **Algorithm 2**：ペナルティ係数の理論的設定（第 5.2 節）
- **Algorithm 6**：上記を統合した実装手順（第 5.6 節）

Algorithm 1 が概念的な全体フローを示し，Algorithm 3〜2 が各ステップの詳細を，Algorithm 6 が実装に即した統合手順を示す．

5.1.7　本手法の全体像

本節では, 本研究で整理した BQM 直接構築アルゴリズムの全体像を示す. Algorithm 1 に全体の疑似コードを示す．赤枠で囲んだ部分が，先行研究では理論のみ，または誤った実装しか存在しなかった箇所であり，本研究の貢献部分である.

BQM（Binary Quadratic Model）は，バイナリ変数に対する二次関数を最小化する問題形式である．この最小化の対象となる関数を**エネルギー関数** $E(\mathbf{x})$ と呼ぶ．量子アニーリングでは，エネルギー関数が最小となる変数の組み合わせ $\mathbf{x}$ を探索する.

BQM のエネルギー関数は以下の形式で定義される：

$$E(\mathbf{x}) = \sum_i h_i x_i + \sum_{i<j} J_{ij} x_i x_j + c \tag{5.3}$$

ここで，h_i は線形係数（変数 x_i 単独の重み），J_{ij} は二次係数（変数ペア x_i, x_j の相互作用の重み），c は定数項（エネルギーのオフセット）である．$\mathbf{x} = (x_1, x_2, \ldots, x_N)$ はバイナリ変数のベクトルであり，各 $x_i \in \{0, 1\}$ である．JSSP では，変数 $x_{j,k,t}$ は「ジョブ j の k 番目の操作が時刻 t に開始するか否か」を表す：

$$x_{j,k,t} = \begin{cases} 1 & \text{操作}(j,k)\text{が時刻}t\text{に開始する} \\ 0 & \text{それ以外} \end{cases} \tag{5.4}$$

BQM では，各 $x_{j,k,t}$ に通し番号 $i = 1, 2, \ldots, N$ を付けて x_i と表記する．変数の総数 N は，全操作の時間窓サイズの総和 $N = \sum_{j,k}(LS_{j,k} - ES_{j,k} + 1)$ となる.

以下，この BQM 係数 $(\mathbf{h}, \mathbf{J}, c)$ を構築する手続きを説明する．図 5.1 に処理フロー全体を示す.

図 5.1　BQM 直接構築の処理フローと各 Algorithm の対応．青枠（太線）は本研究でアルゴリズムを整理・改善した部分，灰色枠（丸角）は既存理論をそのまま使用する部分を示す．右側に対応する Algorithm 番号と理論の出典を記載した．

以下，図 5.1 に示した各ステップを順に説明する．

ステップ 1：JSSP インスタンス読み込み（図 5.1 の最上段）

JSSP インスタンスは，ジョブ集合 $\mathcal{J}$ と各操作の情報（使用機械 $M_{j,k}$，処理時間 $p_{j,k}$）から構成される．また，スケジュールの時間範囲としてメイクスパン上限 T を設定する．

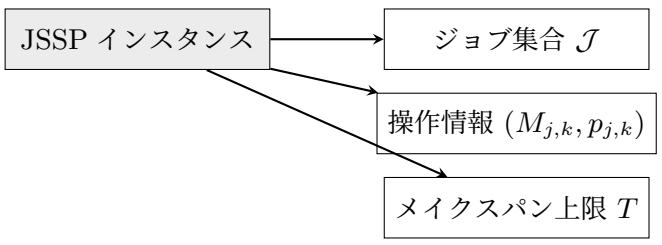


図 5.2　JSSP インスタンスの構成

ステップ 2：時間窓計算・変数生成（図 5.1 の 2 段目）

各操作 $O_{j,k}$ について，開始可能な時刻範囲（時間窓）$[ES_{j,k}, LS_{j,k}]$ を計算する．$ES_{j,k}$ は最早開始時刻，$LS_{j,k}$ は最遅開始時刻である．時間窓内の各時刻 t に対してのみ変数 $x_{j,k,t}$ を生成し，通し番号（インデックス）を割り当てる．

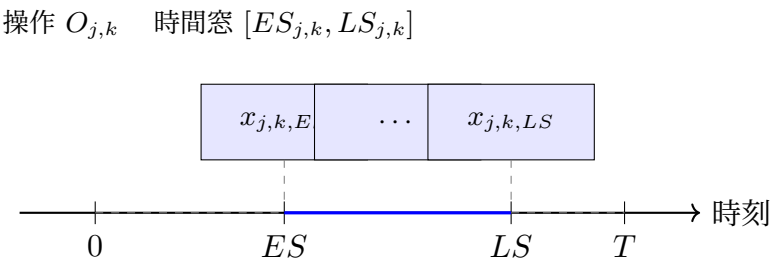


図 5.3　時間窓に基づく変数生成（時間窓内のみに変数を生成）

ペナルティ係数 α，β，γ について

以降のステップでは，制約違反に対するペナルティ係数として正の実数 $\alpha > 0$，$\beta > 0$，$\gamma > 0$ を用いる．これらは Venturelli らの定式化に準拠した記号である：

- α：h_1 制約（順序）用のペナルティ係数
- β：h_2 制約（衝突回避）用のペナルティ係数
- γ：h_3 制約（一意性）用のペナルティ係数

α，β，γ の具体的な設定方法はステップ 6 で述べる．

ステップ 3：h_3 制約（一意性）の係数計算（図 5.1 の 3 段目）

各操作はちょうど 1 つの時刻に開始しなければならない．この一意性制約 $\left(\sum_t x_{j,k,t} - 1\right)^2$ をペナルティ関数として展開すると，線形項，二次項，定数項が得られる．これらの係数を $\mathbf{h}$，$\mathbf{J}$，c に加算する．図 5.4 に，h_3 制約による係数の加算を示す．

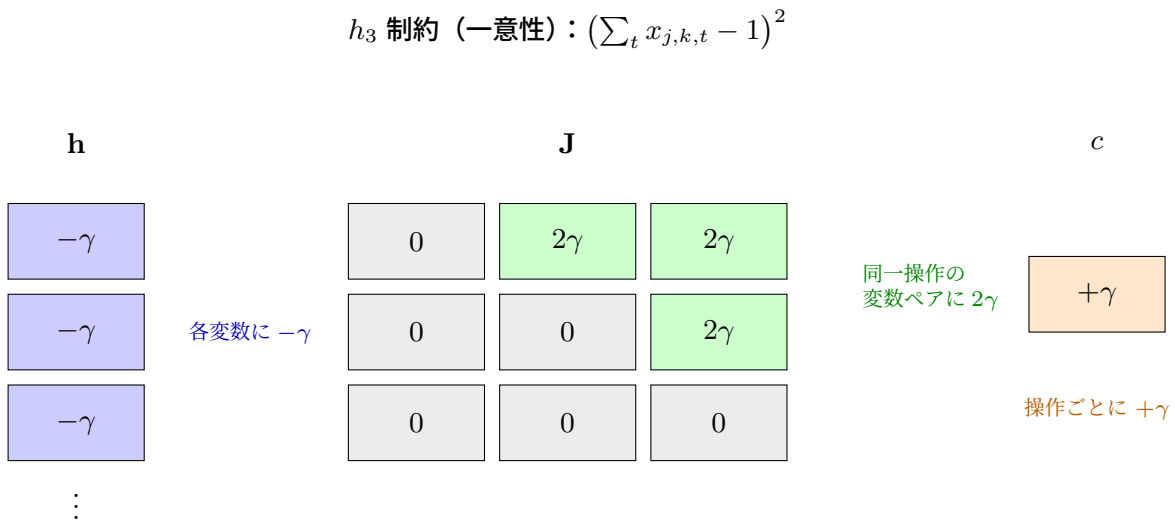


図 5.4　h_3 制約の係数加算．一意性制約の展開により，**h** の各要素に $-\gamma$，**J** の上三角部分に 2γ，c に $+\gamma$ が加算される（γ はペナルティ係数）．

ステップ 4：h_1 制約（順序）の係数計算（図 5.1 の 4 段目）

同一ジョブ内で，先行操作が終了する前に後続操作が開始することを禁止する．この順序制約は変数ペア $(x_{j,k,t}, x_{j,k+1,t'})$ 間の関係であるため，係数は **J** のみに加算される．図 5.5 に，h_1 制約による係数の加算を示す．

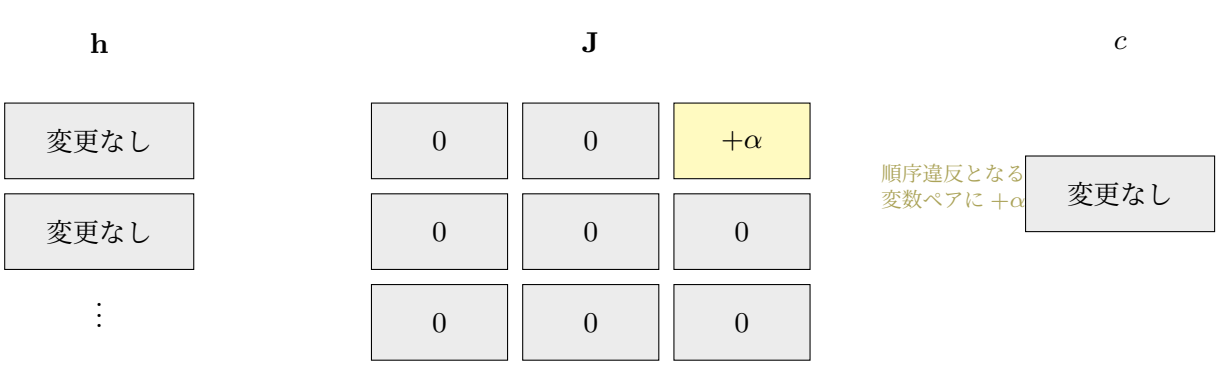


図 5.5　h_1 制約の係数加算．順序違反となる変数ペア (i, j) に対して J_{ij} に $+\alpha$ が加算される（α はペナルティ係数）．**h** と c は変更なし．

ステップ 5：h_2 制約（衝突回避）の係数計算（図 5.1 の 5 段目）

同一機械上で，ある操作の実行中に別の操作が開始することを禁止する．この衝突回避制約は変数ペア $(x_{j,k,t}, x_{j',k',t'})$ 間の関係であるため，係数は $\mathbf{J}$ のみに加算される．図 5.6 に，h_2 制約による係数の加算を示す．

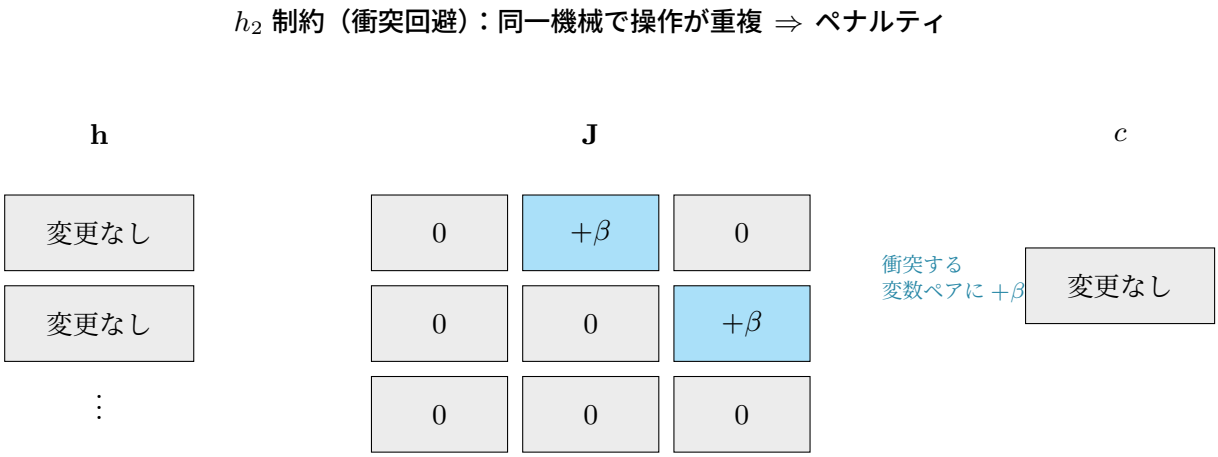
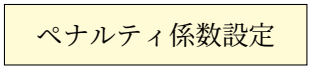


図 5.6　h_2 制約の係数加算．同一機械で時間的に重複する変数ペア (i, j) に対して J_{ij} に $+\beta$ が加算される（β はペナルティ係数）．$\mathbf{h}$ と c は変更なし．

ステップ 6：ペナルティ係数の設定（図 5.1 の 6 段目）

各制約のペナルティ係数 α（h_1 用），β（h_2 用），γ（h_3 用）を設定する．これらの係数は，制約違反時のエネルギー増加量を決定する．



α：h_1 制約（順序）用

β：h_2 制約（衝突回避）用

γ：h_3 制約（一意性）用

図 5.7　ペナルティ係数の設定

ステップ 7：BQM オブジェクト構築（図 5.1 の最下段）

累積された係数 $(\mathbf{h}, \mathbf{J}, c)$ から BQM オブジェクトを生成する．図 5.8 に，各ステップで累積された最終的な係数の構造を示す．

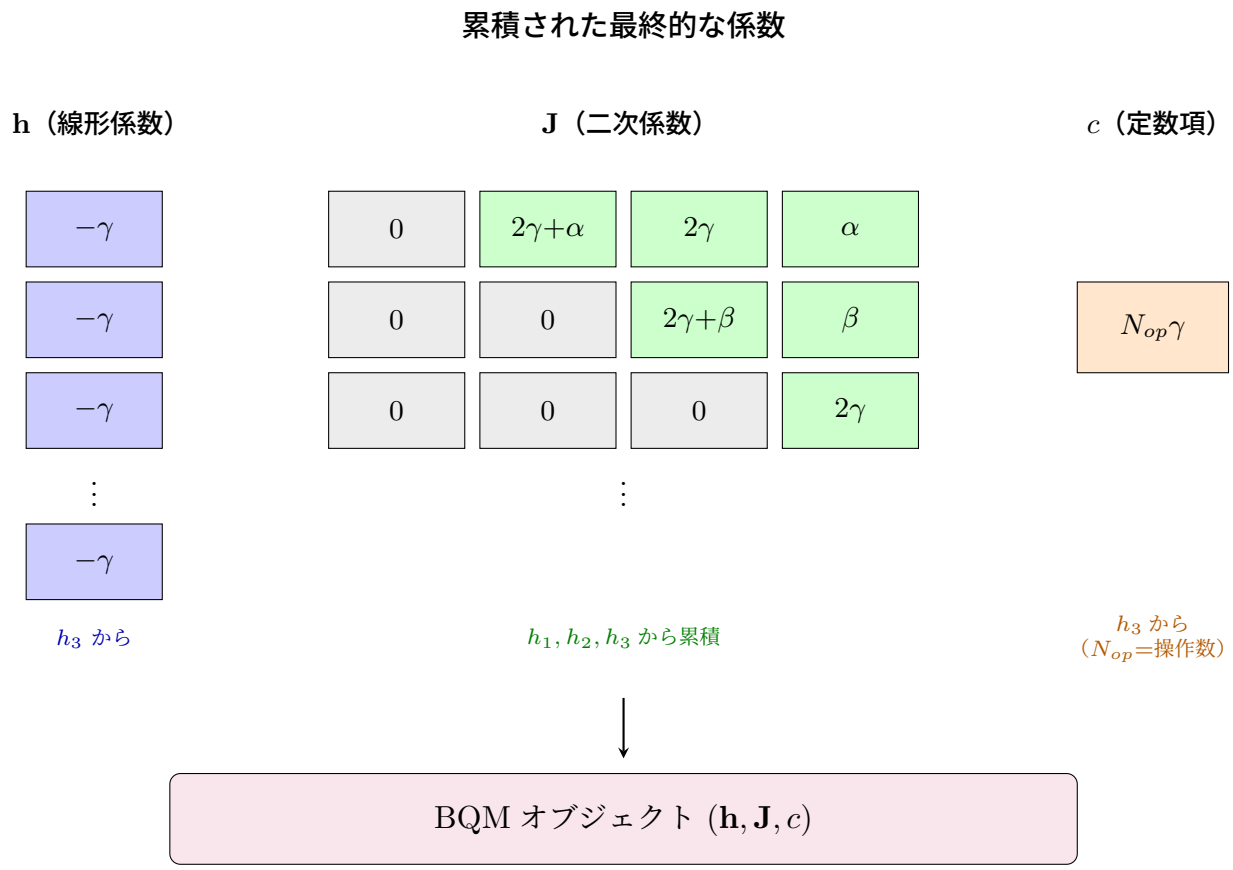


図 5.8　累積された係数の最終的な構造．$\mathbf{h}$ は h_3 制約から，$\mathbf{J}$ は h_1, h_2, h_3 制約から，c は h_3 制約から累積される．

以上の手続きを Algorithm 1 にまとめる．図 5.9 に処理フローを示す．赤枠で囲んだ処理が本研究の貢献部分であり，先行研究では理論のみ，または誤った実装しか存在しなかった箇所である．

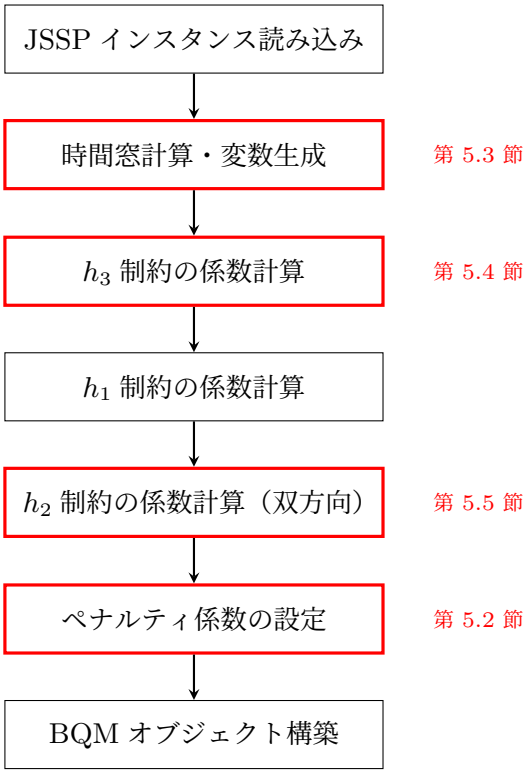


図 5.9　BQM 直接構築の処理フロー（赤枠：本研究の貢献部分）

上記の処理フローをアルゴリズムとして整理すると，Algorithm 1 のようになる．赤枠部分が本研究の貢献部分であり，図 5.9 の赤枠と対応する．なお，各ステップの詳細な実装手順は第 5.6 節の Algorithm 6 で示す．

表 5.3 に，Algorithm 1 で使用する変数の一覧を示す．

表 5.3　Algorithm 1 で使用する変数

分類	変数	説明
入力	$\mathcal{J}$	JSSP インスタンス（全操作の集合）
	$O_{j,k}$	ジョブ j の k 番目の操作
	$p_{j,k}$	操作 $O_{j,k}$ の処理時間
	$\mu_{j,k}$	操作 $O_{j,k}$ が使用する機械
	T	メイクスパン上限（時刻の上限値）
中間変数	α	順序制約 h_1 のペナルティ係数
	β	衝突回避制約 h_2 のペナルティ係数
	γ	一意性制約 h_3 のペナルティ係数
	$ES_{j,k}$	操作 $O_{j,k}$ の最早開始時刻
	$LS_{j,k}$	操作 $O_{j,k}$ の最遅開始時刻
	idx	辞書型．$\mathrm{idx}[j,k,t]=i$ で，変数 $x_{j,k,t}$ の通し番号（整数）を格納
出力	$\mathbf{h}$	線形係数ベクトル（h_i：変数 x_i 単独の重み）
	$\mathbf{J}$	二次係数（J_{ij}：変数ペア x_i, x_j の相互作用）
	c	定数項（エネルギーのオフセット）

Algorithm 1 では，各行で呼び出すサブルーチンと，引数・戻り値の受け渡しを明示する．表 5.4 に呼び出し関係を示す．

表 5.4　Algorithm 1 の呼び出し関係

行	呼び出し先	渡す引数	受け取る戻り値
1	Algorithm 2	なし	α, β, γ
2	Venturelli の時間窓計算（第 3.2 節）	$\mathcal{J}, T$	$\{[ES_{j,k}, LS_{j,k}]\}$
3	Algorithm 3	$\{[ES_{j,k}, LS_{j,k}]\}$	idx
5	Algorithm 4	$\mathcal{J}, \mathrm{idx}, \gamma, \mathbf{h}, \mathbf{J}, c$	$\mathbf{h}, \mathbf{J}, c$（更新後）
6	h_1 制約計算（第 3.2 節）	$\mathcal{J}, \mathrm{idx}, \alpha, \mathbf{J}$	$\mathbf{J}$（更新後）
7	Algorithm 5	$\mathcal{J}, \mathrm{idx}, \beta, \mathbf{J}$	$\mathbf{J}$（更新後）

Algorithm 1 BQM 直接構築アルゴリズム（全体フロー）

入力: JSSP インスタンス $\mathcal{J} = \{(O_{j,k}, p_{j,k}, \mu_{j,k})\}$，メイクスパン上限 T

出力: BQM オブジェクト

1: $(\alpha, \beta, \gamma) \leftarrow$ **Algorithm 2**() ▷ ペナルティ係数を取得
2: 時間窓 $\{[ES_{j,k}, LS_{j,k}]\} \leftarrow$ Venturelli の手法で計算（第 3.2 節参照，$\mathcal{J}, T$ を使用）
3: インデックス表 idx $\leftarrow$ **Algorithm 3**($\{[ES_{j,k}, LS_{j,k}]\}$)
4: $\mathbf{h} \leftarrow \mathbf{0}$，$\mathbf{J} \leftarrow \{\}$，$c \leftarrow 0$ ▷ 係数を初期化
5: $(\mathbf{h}, \mathbf{J}, c) \leftarrow$ **Algorithm 4**($\mathcal{J}$, idx, γ, $\mathbf{h}$, $\mathbf{J}$, c)
6: $\mathbf{J} \leftarrow h_1$ 制約の係数計算（第 3.2 節参照，$\mathcal{J}$, idx, α, $\mathbf{J}$）
7: $\mathbf{J} \leftarrow$ **Algorithm 5**($\mathcal{J}$, idx, β, $\mathbf{J}$)
8: **return** `BinaryQuadraticModel`($\mathbf{h}, \mathbf{J}, c$, `BINARY`)

以降の節では，図 5.9 で赤枠で示した各処理について詳細を述べる：

- 第 5.2 節：ペナルティ係数の理論的設定
- 第 5.3 節：時間窓計算と変数生成
- 第 5.4 節：h_3 制約の正確な係数導出
- 第 5.5 節：h_2 制約の双方向判定
- 第 5.6 節：手続き全体の統合

5.2 ペナルティ係数の設定

5.2.1 現状と問題

QUBO 定式化では，制約違反に対するペナルティ係数 α（順序），β（衝突回避），γ（一意性）の設定が解の品質に影響する．係数が小さすぎると制約違反が許容され，大きすぎると探索空間が歪む．Carugno らは，各制約違反でエネルギーが 2 増加するよう $\alpha = \beta = \gamma = 2.0$ と設定している [5].

5.2.2 先行研究の問題点

ペナルティ係数の設定について，各先行研究の問題点を整理する：

- Venturelli：「η, α, β は 0 より大きく」と概念的に言及するのみで，具体的な値は示していない

- Carugno：各制約違反でエネルギーが 2 増加するよう設定しているが，設定手続き（アルゴリズム）は示していない
- Robinbux：試行錯誤的に調整（制約違反時に +1 ずつ増加し，成功回数が閾値に達するまで繰り返す）

5.2.3 本研究の改善

本研究では，Carugno らの設定に基づき，ペナルティ係数を固定値 2.0 として設定する．これにより，試行錯誤的な調整ループを排除し，再現可能な手順を実現する．

5.2.4 前後比較

図 5.10 に，Robinbux（左）と本研究（右）のペナルティ係数設定方法を比較する．左図は試行錯誤ループによる調整，右図は Carugno らの設定値による直接設定を示す．

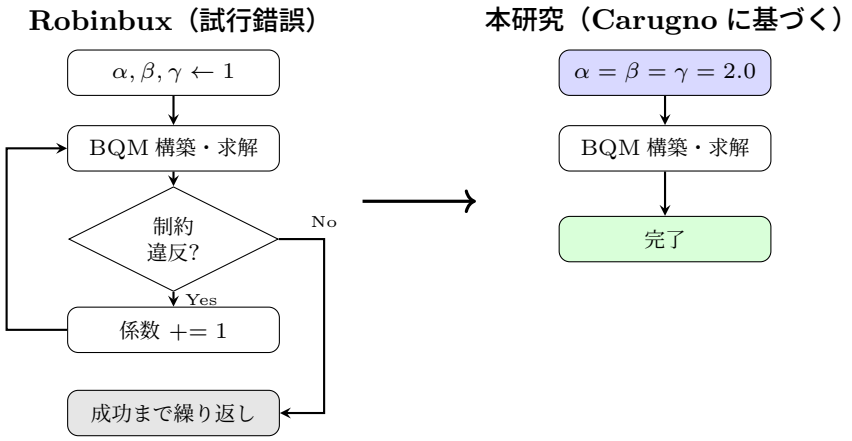


図 5.10　ペナルティ係数設定の比較．左：Robinbux（試行錯誤ループ），右：本研究（Carugno らの設定値を直接使用）．

5.2.5 アルゴリズム

Algorithm 2 に，本研究のペナルティ係数設定手続きを示す．このアルゴリズムは，Algorithm 1 の 1 行目から呼び出され，Carugno らの理論的設定に基づき，ペナルティ係数を固定値として設定する．赤枠部分が先行研究との差分である．

Algorithm 2 ペナルティ係数の設定（理論的設定）

入力: なし（理論値を使用）

出力: ペナルティ係数 (α, β, γ)（Algorithm 1 の 1 行目に返す）

1: $\alpha \leftarrow 2.0$ ▷【変更】順序制約 h_1

2: $\beta \leftarrow 2.0$ ▷【変更】衝突回避制約 h_2

3: $\gamma \leftarrow 2.0$ ▷【変更】一意性制約 h_3

4: **return** (α, β, γ)

各先行研究との差分：

- 2〜4 行目（赤枠）：理論値 2.0 を直接設定
  - vs Venturelli：「十分大きく」と概念的言及のみ
  - vs Carugno：理論値は示したが，設定アルゴリズムは示していない
  - vs Robinbux：1 から開始し，制約違反時に +1 ずつ増加するループ

5.2.6 アルゴリズムの説明

このアルゴリズムでは，QUBO 定式化のペナルティ係数を理論値に基づいて設定する．以下，各部分について説明する．

まず，入力と出力について説明する．

入力: なし（理論値を使用）
出力: ペナルティ係数 α, β, γ

このアルゴリズムは入力を必要としない．Carugno らが理論的に導出した値を直接使用する．出力として，順序制約用の α，衝突回避制約用の β，一意性制約用の γ を返す．

次に，係数の設定について説明する．

1: $\alpha \leftarrow 2.0$
2: $\beta \leftarrow 2.0$
3: $\gamma \leftarrow 2.0$
4: **return** α, β, γ

全てのペナルティ係数を 2.0 に設定する．この値は Carugno らが採用した設定であり，各制約違反でエネルギーが 2 増加するよう設計されている [5].

例えば，Robinbux の実装では $\alpha = \beta = \gamma = 1$ から開始し，制約違反が検出されるたびに係数を +1 ずつ増加させる試行錯誤ループを用いていた．本アルゴリズムでは理論値

を直接設定することで，この不要な反復処理を排除できる．

5.3 時間窓による変数削減

5.3.1 現状と問題

時間インデックスモデルでは，各操作 $O_{j,k}$ と各時刻 t の組み合わせに対して変数 $x_{j,k,t}$ を生成する．時間窓を考慮しない場合，変数数は $n \times m \times T$ となる．

しかし，Venturelli らの時間窓計算により，各操作の開始可能時刻は $[ES_{j,k}, LS_{j,k}]$ に制限される．時間窓外の時刻（$t < ES_{j,k}$ または $t > LS_{j,k}$）では $x_{j,k,t} = 0$ が確定しているため，これらの変数は不要である．

5.3.2 先行研究の問題点

時間窓を使った変数生成について，各先行研究の問題点を整理する：

- Venturelli：時間窓計算のアルゴリズムは示したが，その結果を使った変数生成手続きは示していない
- Carugno：penaltymodel ライブラリに委託しており，変数生成手続きは示していない
- Robinbux：時間窓計算を行わず，時間上限 $T = 9$ を固定し，すべての時刻に対して変数を生成する

5.3.3 理論

本アルゴリズムは，第 2.3 節および第 3.2 節で述べた時間窓の理論に基づく．

第 2.3 節では，時間窓 $[ES_{j,k}, LS_{j,k}]$ の定義と，最早開始時刻・最遅開始時刻の計算式を示した：

$$ES_{j,k} = \sum_{k' < k} p_{j,k'} \tag{5.5}$$

$$LS_{j,k} = T - \sum_{k' \geq k} p_{j,k'} \tag{5.6}$$

第 3.2 節では，Venturelli らがこの時間窓計算を Window Shaving として提案したことを述べた．

本アルゴリズムは，これらの理論で計算された時間窓を用いて，時間窓内の時刻 $t \in [ES_{j,k}, LS_{j,k}]$ のみに変数 $x_{j,k,t}$ を生成し，各変数に QUBO 行列のインデックスを割り当てる手続きを明示化したものである．

5.3.4 本研究の改善

本研究では，時間窓 $[ES_{j,k}, LS_{j,k}]$ 内の時刻のみに変数を生成する．

5.3.5 前後比較

図 5.11 に，Robinbux（左）と本研究（右）の変数生成を比較する．2 ジョブ 2 機械（$T = 9$）の例で，行は各操作 $O_{j,k}$（4 操作），列は時刻 $t \in \{0, 1, \ldots, 8\}$ を表す．青色セルは変数 $x_{j,k,t}$ を生成することを，灰色セルは変数を除外（時間窓外）することを示す．左図（Robinbux）では時間窓計算を行わないため，全操作・全時刻に対して変数を生成する（36 変数）．右図（本研究）では時間窓 $[ES_{j,k}, LS_{j,k}]$ 内の時刻のみに変数を生成する（16 変数，56% 削減）．

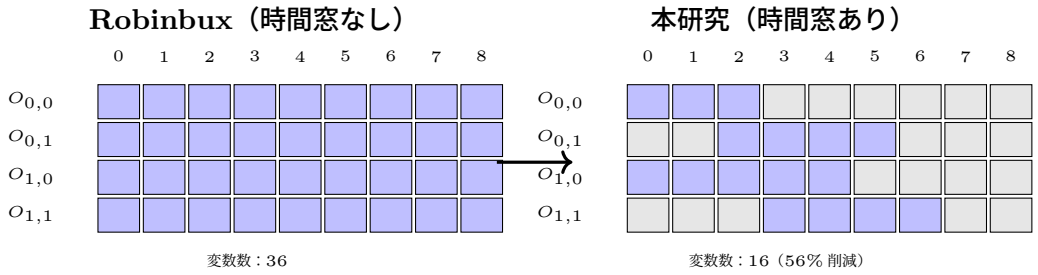


図 5.11　変数生成の比較．左：Robinbux（全時刻に変数生成），右：本研究（Venturelli らの時間窓計算に基づき時間窓内のみ生成）．

5.3.6 アルゴリズム

Algorithm 3 に，本研究の変数生成手続きを示す．このアルゴリズムは，Algorithm 1 の 3 行目から呼び出され，各操作 $O_{j,k}$ について時間窓 $[ES_{j,k}, LS_{j,k}]$ 内の時刻のみに変数 $x_{j,k,t}$ を生成し，各変数に QUBO 行列のインデックスを割り当てる．赤枠部分が先行研究との差分である．

Algorithm 3 変数生成（時間窓あり）

入力: 時間窓の集合 $\{[ES_{j,k}, LS_{j,k}]\}$（Algorithm 1 の 2 行目で計算）

出力: インデックス対応表 idx：$(j,k,t) \mapsto i$（QUBO 行列の行・列番号）

1: idx $\leftarrow \{\}$, $i \leftarrow 0$
2: **for** 各操作 $O_{j,k}$ **do**
3: **for** $t \leftarrow ES_{j,k}$ to $LS_{j,k}$ **do** ▷【変更】時間窓内のみ
4: idx$[j,k,t] \leftarrow i$ ▷ 変数 $x_{j,k,t}$ にインデックス i を割り当て
5: $i \leftarrow i+1$
6: **end for**
7: **end for**
8: **return** idx ▷ Algorithm 1 の 3 行目に返す

各先行研究との差分：

- 1 行目（赤枠）：Venturelli の時間窓計算結果を入力として使用
  - vs Venturelli：時間窓計算アルゴリズムは示したが，変数生成手続きは示していない
  - vs Carugno：penaltymodel ライブラリに委託しており，変数生成手続きは示していない
  - vs Robinbux：時間窓計算を行わないため，この入力自体がない
- 5 行目（赤枠）：時間窓 $[ES_{j,k}, LS_{j,k}]$ 内のみループ
  - vs Venturelli：変数生成手続きを示していない
  - vs Carugno：penaltymodel ライブラリに委託
  - vs Robinbux：$t \leftarrow 0$ to $T-1$ で全時刻をループ

5.3.7 アルゴリズムの説明

このアルゴリズムでは，QUBO 変数の集合とインデックス対応表を作成する．以下，各部分について説明する．

まず，入力と出力について説明する．

入力: 各操作の時間窓 $[ES_{j,k}, LS_{j,k}]$
出力: 変数集合 X，インデックス対応表

入力として，各操作が開始可能な最早時刻 $ES_{j,k}$ と最遅時刻 $LS_{j,k}$ の範囲を受け取る．

これは Algorithm 1 の 2 行目で計算され，本アルゴリズムに渡される．出力として，ジョブ番号・操作番号・時刻の組 (j, k, t) から QUBO 行列のインデックス i を引けるインデックス対応表 idx を返す．この idx は Algorithm 4 や Algorithm 5 で係数を設定する際に使用される．

例えば，操作 $O_{1,1}$ の時間窓が $[0, 3]$，操作 $O_{1,2}$ の時間窓が $[2, 5]$ のように，各操作について開始可能な時刻範囲が与えられる．

次に，初期化部分を説明する．

```
1: idx ← {}, i ← 0
```

インデックス対応表 idx を空の辞書で初期化し，インデックスのカウンタ i を 0 に設定する．

次に，変数生成のループ部分を説明する．

```
1: for 各操作 O_{j,k} do
2:     for t ← ES_{j,k} to LS_{j,k} do
3:         idx[j, k, t] ← i                ▷ 変数 x_{j,k,t} にインデックス i を割り当て
4:         i ← i + 1
5:     end for
6: end for
7: return idx
```

全ての操作について，時間窓内の各時刻に対して変数 $x_{j,k,t}$ を 1 つ生成する．対応表 $\mathrm{idx}[j, k, t]$ にカウンタの値を登録してカウンタを 1 増やす．これにより，ジョブ j の k 番目の操作が時刻 t に開始することを表す変数に，一意のインデックスが割り当てられる．最終的に，インデックス対応表 idx を返す．

例えば，操作 $O_{1,1}$ の時間窓が $[0, 3]$ の場合，変数 $x_{1,1,0}, x_{1,1,1}, x_{1,1,2}, x_{1,1,3}$ が生成され，それぞれにインデックス $0, 1, 2, 3$ が割り当てられる．対応表には $\mathrm{idx}[1, 1, 0] = 0$, $\mathrm{idx}[1, 1, 1] = 1$, $\mathrm{idx}[1, 1, 2] = 2$, $\mathrm{idx}[1, 1, 3] = 3$ と登録される．

ここで作成したインデックス対応表は，後続の制約係数計算（h_1, h_2, h_3）において，変数ペアに対応する QUBO 行列の位置を特定するために使用する．

5.4 h_3 制約の正確な係数

5.4.1 現状と問題

一意性制約 h_3 は，各操作が正確に 1 つの時刻に開始することを要求する．Venturelli の定式化では：

$$h_3 = \gamma \sum_{j,k} \left(1 - \sum_{t} x_{j,k,t}\right)^2 \tag{5.7}$$

この制約を QUBO 係数に変換するには，二値変数の性質（$x^2 = x$）を用いた展開が必要である．展開結果から，線形項は $-\gamma$，二次項は 2γ，定数項（offset）は $\gamma \times n \times m$ となる．

5.4.2 先行研究の問題点

h_3 制約の係数計算について，各先行研究の問題点を整理する：

- Venturelli：理論（数式）は示したが，QUBO 係数への展開手順は示していない
- Carugno：penaltymodel ライブラリに委託しており，係数計算手続きは示していない
- Robinbux：線形項の係数が誤っており，offset 項も欠落している

5.4.3 理論

本アルゴリズムは，第 3.2 節で述べた Venturelli らの一意性制約 h_3 の定式化に基づく．

Venturelli らは，各操作が正確に 1 つの時刻に開始するという制約を，以下のペナルティ関数で表現した：

$$h_3 = \gamma \sum_{j,k} \left(1 - \sum_{t} x_{j,k,t}\right)^2 \tag{5.8}$$

本アルゴリズムは，この数式を二値変数の性質（$x^2 = x$）を用いて展開し，QUBO 係数（線形項 $-\gamma$，二次項 2γ，offset 項 γ）を導出する手続きを明示化したものである．

5.4.4 本研究の改善

本研究では，Venturelli の定式化を正確に展開し，線形項・二次項・offset 項の係数を正しく設定する．

5.4.5　前後比較

図 5.12 に，Robinbux（左）と本研究（右）の h_3 制約係数を比較する．1 つの操作 $O_{j,k}$ について，時間窓内の時刻 $t \in \{0, 1, 2\}$ に対応する 3 変数の例である．行・列は変数インデックス i を表し，セルの値は QUBO 係数 Q_{ij} を示す．

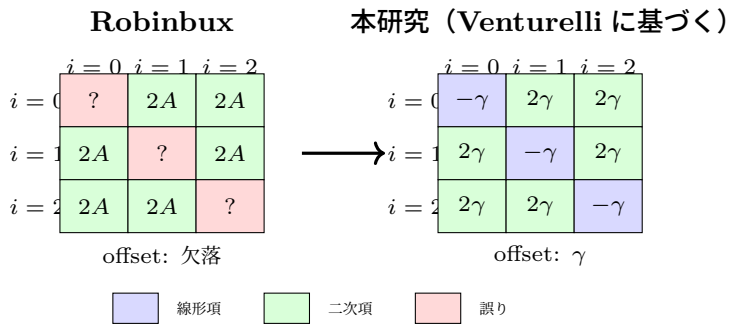


図 5.12　h_3 制約係数の比較（1 操作・3 変数の例）．左：Robinbux（線形項誤り・offset 欠落），右：本研究（Venturelli の定式化に基づく正確な係数）．

5.4.6　アルゴリズム

Algorithm 4 に，本研究の h_3 制約係数計算手続きを示す．このアルゴリズムは，Algorithm 1 の 5 行目から呼び出され，各操作について一意性制約を展開し，線形項・二次項・offset 項の係数を正確に計算する．赤枠部分が先行研究との差分である．

Algorithm 4 h_3 制約の係数計算（正確な展開）

入力: JSSP インスタンス $\mathcal{J}$，インデックス表 idx（Algorithm 3 の出力），
ペナルティ係数 γ，現在の係数 $\mathbf{h}, \mathbf{J}, c$

出力: 更新後の $\mathbf{h}, \mathbf{J}, c$（Algorithm 1 の 5 行目に返す）

for 各操作 $O_{j,k} \in \mathcal{J}$ **do**
  for $t \in [ES_{j,k}, LS_{j,k}]$ **do**
    $\mathbf{h}[\mathrm{idx}[j,k,t]] \mathrel{+}= -\gamma$ ▷【変更】線形項
    for $t' \in [ES_{j,k}, LS_{j,k}]$, $t' > t$ **do**
      $\mathbf{J}[\mathrm{idx}[j,k,t], \mathrm{idx}[j,k,t']] \mathrel{+}= 2\gamma$ ▷ 二次項
    end for
  end for
  $c \mathrel{+}= \gamma$ ▷【追加】offset 項
end for
return $(\mathbf{h}, \mathbf{J}, c)$

各先行研究との差分：

- 4 行目（赤枠）：線形項の係数 $-\gamma$
  - vs Venturelli：理論は示したが，アルゴリズムは示していない
  - vs Carugno：penaltymodel ライブラリに委託
  - vs Robinbux：係数が誤っている
- 8 行目（赤枠）：offset 項 γ の加算
  - vs Venturelli：理論は示したが，アルゴリズムは示していない
  - vs Carugno：penaltymodel ライブラリに委託
  - vs Robinbux：offset 項が欠落している

5.4.7 アルゴリズムの説明

このアルゴリズムでは，一意性制約 h_3 を QUBO 形式に展開し，線形係数・二次係数・定数項（offset）を計算する．以下，各部分について説明する．

まず，入力と出力について説明する．

入力: JSSP インスタンス $\mathcal{J}$，インデックス表 idx，ペナルティ係数 γ，現在の係数 $\mathbf{h}, \mathbf{J}, c$
出力: 更新後の $\mathbf{h}, \mathbf{J}, c$

入力として，JSSP インスタンス $\mathcal{J}$，Algorithm 3 で生成したインデックス表 idx，ペナルティ係数 γ，および現在の係数 $\mathbf{h}, \mathbf{J}, c$ を受け取る．出力として，一意性制約の係数を加算した後の $\mathbf{h}, \mathbf{J}, c$ を返す．

次に，操作ごとのループと線形項の計算について説明する．

```
for 各操作 O_{j,k} ∈ J do
    for t ∈ [ES_{j,k}, LS_{j,k}] do
        h[idx[j,k,t]] += −γ
    end for
end for
```

各操作 $O_{j,k}$ について，時間窓内の各時刻 t に対応する変数 $x_{j,k,t}$ の線形係数に $-\gamma$ を加算する．ここで $\mathrm{idx}[j,k,t]$ は，Algorithm 3 で生成したインデックス対応表から，変数 $x_{j,k,t}$ に対応する QUBO 行列のインデックスを取得する．これは一意性制約 $\gamma(\sum_t x_{j,k,t} - 1)^2$ を展開した際に，二値性 $x^2 = x$ を適用して $\gamma - \gamma \sum_t x_{j,k,t} + 2\gamma \sum_{t<t'} x_t x_{t'}$ を得たことによる（第 3.2 節の導出を参照）．

次に，二次項の計算について説明する．

```
for t' ∈ [ES_{j,k}, LS_{j,k}], t' > t do
    J[idx[j,k,t], idx[j,k,t']] += 2γ
end for
```

同一操作内の異なる時刻 t と t'（$t' > t$）に対応する変数ペア $(x_{j,k,t}, x_{j,k,t'})$ について，二次係数に 2γ を加算する．これは展開式の $2\gamma \cdot x_{j,k,t} \cdot x_{j,k,t'}$ 項に対応する．

最後に，定数項（offset）の計算について説明する．

```
c += γ
```

各操作について定数項 c に γ を加算する．これは展開式の $\gamma \cdot 1^2 = \gamma$ に対応する．この定数項は最適化には影響しないが，エネルギー値の正確な計算に必要である．

例えば，操作 $O_{1,1}$ の時間窓が $[0,2]$ で $\gamma = 2$ の場合を考える．線形係数は $\mathbf{h}[0], \mathbf{h}[1], \mathbf{h}[2]$ にそれぞれ -2 が加算される．二次係数は $\mathbf{J}[0,1], \mathbf{J}[0,2], \mathbf{J}[1,2]$ にそれぞれ 4 が加算される．定数項は c に 2 が加算される．

5.5 h_2 制約の双方向判定

5.5.1 現状と問題

衝突回避制約 h_2 は，同一機械を使用する異なる操作が時間的にオーバーラップしないことを保証する．Venturelli の定式化では，2 つの操作 $O_{j,k}$ と $O_{j',k'}$ のオーバーラップ条件は：

$$(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t) \tag{5.9}$$

この条件は双方向の判定を要求する．

5.5.2 先行研究の問題点

h_2 制約の双方向判定について，各先行研究の問題点を整理する：

- Venturelli：理論（数式）は示したが，アルゴリズム（疑似コード）は示していない
- Carugno：penaltymodel ライブラリに委託しており，判定手続きは示していない
- Robinbux：一方向のみをチェック（$0 \leq t' - t < p_{j,k}$）し，$t' < t$ の場合を見逃す

5.5.3 理論

本アルゴリズムは，第 3.2 節で述べた Venturelli らの衝突回避制約 h_2 の定式化に基づく．

Venturelli らは，同一機械を使用する 2 つの操作 $O_{j,k}$ と $O_{j',k'}$ が時間的にオーバーラップする条件を，以下のように定義した：

$$(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t) \tag{5.10}$$

この条件は，「$O_{j,k}$ が時刻 t に開始し，$O_{j',k'}$ が時刻 t' に開始するとき，両者の実行期間が重なる」ことを意味する．

本アルゴリズムは，この双方向のオーバーラップ条件を正確に実装し，条件を満たす時刻ペア (t, t') に対してペナルティ係数 β を加算する手続きを明示化したものである．

5.5.4 本研究の改善

本研究では，Venturelli の定式化に従い，双方向の判定条件を正確に実装する．

5.5.5　前後比較

図 5.13 に，Robinbux（左）と本研究（右）の h_2 制約判定を比較する．同一機械を使用する 2 つの操作 O_1（処理時間 $p_1 = 2$）と O_2（処理時間 $p_2 = 2$）の例で，行は O_1 の開始時刻 $t \in \{0, 1, 2, 3\}$，列は O_2 の開始時刻 $t' \in \{0, 1, 2, 3\}$ を表す．青色セルはオーバーラップを正しく検出（ペナルティを加算），赤色セルはオーバーラップを見逃し（ペナルティ未加算），灰色セルは非オーバーラップ（ペナルティ不要）を示す．左図（Robinbux）では一方向条件 $0 \leq t' - t < p_1$ のみをチェックするため，$t' < t$ のケース（対角線より下）でオーバーラップを見逃す．右図（本研究）では Venturelli の双方向条件 $(t + p_1 > t') \wedge (t' + p_2 > t)$ を正確に実装し，全てのオーバーラップを検出する．

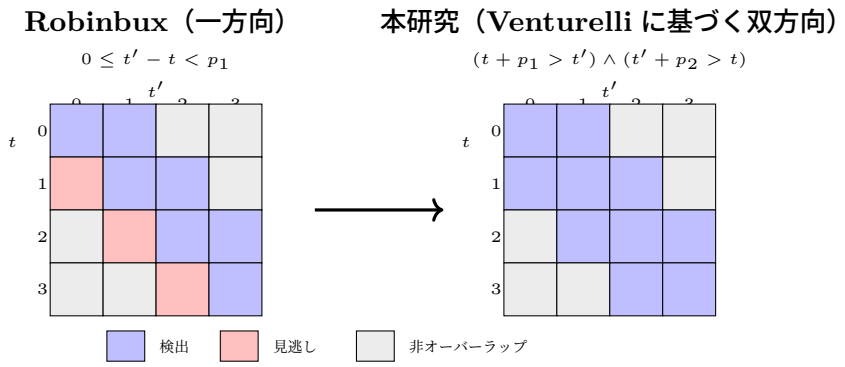


図 5.13　h_2 制約判定の比較（$p_1 = p_2 = 2$ の例）．左：Robinbux（一方向判定，赤は見逃し），右：本研究（双方向判定）．

5.5.6　アルゴリズム

Algorithm 5 に，本研究の h_2 制約係数計算手続きを示す．このアルゴリズムは，Algorithm 1 の 7 行目から呼び出され，同一機械を使用する操作ペアについて，双方向のオーバーラップ条件を満たす時刻ペアにペナルティ係数を加算する．赤枠部分が先行研究との差分である．

Algorithm 5 h_2 制約の係数計算（双方向判定）

入力: JSSP インスタンス $\mathcal{J}$，インデックス表 idx（Algorithm 3 の出力），
ペナルティ係数 β，現在の二次係数 $\mathbf{J}$

出力: 更新後の $\mathbf{J}$（Algorithm 1 の 7 行目に返す）

```
for 各機械 μ do
    for μ を使用する操作ペア (O_{j,k}, O_{j',k'}) ∈ J do
        for t ∈ [ES_{j,k}, LS_{j,k}], t' ∈ [ES_{j',k'}, LS_{j',k'}] do
            if (t + p_{j,k} > t') ∧ (t' + p_{j',k'} > t) then      ▷【変更】双方向判定
                J[idx[j,k,t], idx[j',k',t']] += β
            end if
        end for
    end for
end for
return J
```

各先行研究との差分：

- 5 行目（赤枠）：双方向のオーバーラップ判定
  - vs Venturelli：理論は示したが，アルゴリズムは示していない
  - vs Carugno：penaltymodel ライブラリに委託
  - vs Robinbux：$0 \leq t' - t < p_{j,k}$（一方向のみ）

5.5.7 アルゴリズムの説明

このアルゴリズムでは，同一機械を使用する操作ペアについて，衝突回避制約 h_2 の係数を QUBO 行列に設定する．以下，各部分について説明する．

まず，入力と出力について説明する．

入力: JSSP インスタンス $\mathcal{J}$，インデックス表 idx，ペナルティ係数 β，現在の二次係数 $\mathbf{J}$
出力: 更新後の $\mathbf{J}$

入力として，JSSP インスタンス $\mathcal{J}$，Algorithm 3 で生成したインデックス表 idx，ペナルティ係数 β，および現在の二次係数 $\mathbf{J}$ を受け取る．出力として，衝突回避制約のペナルティ係数を加算した後の $\mathbf{J}$ を返す．

次に，機械ごとのループと操作ペアのループについて説明する．

```
for 各機械 μ do
    for μ を使用する操作ペア (O_{j,k}, O_{j',k'}) do
        ...
    end for
end for
```

各機械 μ について，その機械を使用する全ての操作ペア $(O_{j,k}, O_{j',k'})$ を列挙する．衝突回避制約は同一機械を使用する操作間でのみ適用されるため，このループ構造となる．

例えば，機械 M_1 を使用する操作が $O_{1,1}$，$O_{2,2}$，$O_{3,1}$ の 3 つである場合，ペア $(O_{1,1}, O_{2,2})$, $(O_{1,1}, O_{3,1})$, $(O_{2,2}, O_{3,1})$ について処理を行う．

次に，時刻ペアのループと双方向オーバーラップ判定について説明する．

```
for t ∈ [ES_{j,k}, LS_{j,k}], t' ∈ [ES_{j',k'}, LS_{j',k'}] do
    if (t + p_{j,k} > t') ∧ (t' + p_{j',k'} > t) then
        J[idx[j, k, t], idx[j', k', t']] += β
    end if
end for
```

操作ペア $(O_{j,k}, O_{j',k'})$ について，それぞれの時間窓内の全ての時刻の組み合わせ (t, t') を調べる．条件 $(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t)$ は，操作 $O_{j,k}$ が時刻 t に開始し，操作 $O_{j',k'}$ が時刻 t' に開始した場合に，両者の実行期間が重なるかを判定する．この条件を満たす場合，二次係数辞書 $\mathbf{J}$ の該当位置にペナルティ係数 β を加算する．

例えば，操作 $O_{1,1}$（処理時間 $p_{1,1} = 3$）と操作 $O_{2,2}$（処理時間 $p_{2,2} = 2$）について，$t = 2$, $t' = 4$ の場合を考える．条件は $(2 + 3 > 4) \wedge (4 + 2 > 2)$，すなわち $(5 > 4) \wedge (6 > 2)$ となり，両方とも真なので衝突が発生する．この場合，$\mathbf{J}[\mathrm{idx}[1, 1, 2], \mathrm{idx}[2, 2, 4]]$ に β を加算する．

5.6 手続き (a)〜(d) を統合した BQM 構築手順

5.6.1 現状と問題

第 5.3 節〜第 5.2 節では，BQM 構築の各要素について個別に改善を示した．しかし，これらを統合した全体手順がなければ，再現可能な実装は困難である．

5.6.2 先行研究の問題点

統合 BQM 構築手順について，各先行研究の問題点を整理する：

- Venturelli：各制約の理論は示したが，統合的な構築アルゴリズムは示していない
- Carugno：penaltymodel ライブラリに委託しており，構築手順は示していない
- Robinbux：統合的な手順は示しているが，時間窓計算の欠落，h_2 の一方向判定，h_3 の係数誤り等の問題がある

5.6.3 理論

本アルゴリズムは，第 3.2 節で述べた Venturelli らの QUBO 定式化（h_1, h_2, h_3）と，第 2.3 節で述べた時間インデックスモデルの理論に基づく．

本アルゴリズムは，これらの理論を実装するために必要な以下の手続きを統合したものである：

- 時間窓計算と変数生成（第 5.3 節）
- h_3 制約の係数計算（第 5.4 節）
- h_2 制約の双方向判定（第 5.5 節）
- ペナルティ係数の設定（第 5.2 節）

5.6.4 本研究の改善

本研究では，第 5.3 節〜第 5.2 節の全改善点を統合した，再現可能な BQM 構築手順を提示する．

5.6.5 前後比較

図 5.14 に，各先行研究と本研究の BQM 構築フローを比較する．左から Venturelli, Carugno, Robinbux, 本研究の順に示す．赤字は問題点，赤枠は本研究での改善点を示す．

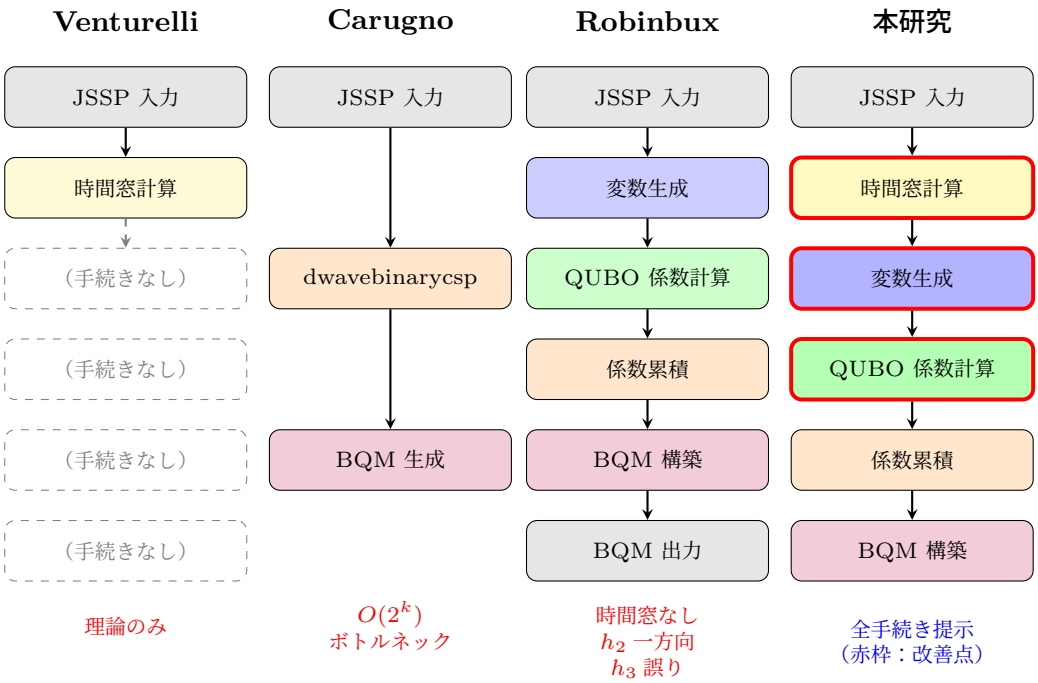


図 5.14　各手法の BQM 構築フロー比較．左から Venturelli，Carugno，Robinbux，本研究．赤枠は本研究での改善点．

5.6.6　アルゴリズム

Algorithm 6 に，本研究の統合 BQM 構築手続きを示す．このアルゴリズムは，第 5.3 節〜第 5.2 節で述べた全ての改善点を統合したものである．赤枠部分が先行研究との差分である．

Algorithm 6 統合 BQM 構築手続き

入力: JSSP インスタンス（ジョブ数 n, 機械数 m, 処理時間 $p_{j,k}$, 機械割当 $\mu_{j,k}$, timespan T）

出力: BQM オブジェクト

1: $\alpha, \beta, \gamma \leftarrow 2.0$ ▷【変更】ペナルティ係数
2: 各操作 $O_{j,k}$ の時間窓 $[ES_{j,k}, LS_{j,k}]$ を計算（T を上限として使用） ▷【追加】
3: $\mathbf{h} \leftarrow \{\}$, $\mathbf{J} \leftarrow \{\}$, $c \leftarrow 0$
4: **for** 各操作 $O_{j,k}$ **do**
5: **for** $t \in [ES_{j,k}, LS_{j,k}]$ **do** ▷【変更】時間窓内のみ
6: idx$[j,k,t] \leftarrow$ 連番割り当て
7: $\mathbf{h}[\text{idx}[j,k,t]] \mathrel{+}= -\gamma$ ▷【変更】h_3 線形項
8: **end for**
9: $\mathbf{J} \mathrel{+}= 2\gamma$（同一操作内の全ペア）, $c \mathrel{+}= \gamma$ ▷【追加】offset
10: **end for**
11: **for** 各ジョブ j の連続操作 $(O_{j,k}, O_{j,k+1})$ **do**
12: $t + p_{j,k} > t'$ を満たすペアに $\mathbf{J} \mathrel{+}= \alpha$（$h_1$ 制約）
13: **end for**
14: **for** 同一機械の無順序操作ペア $(O_{j,k}, O_{j',k'})$（$(j,k) < (j',k')$）**do**
15: $(t + p_{j,k} > t') \wedge (t' + p_{j',k'} > t)$ を満たすペアに $\mathbf{J} \mathrel{+}= \beta$ ▷【変更】h_2 双方向
16: **end for**
17: **return** `BinaryQuadraticModel`$(\mathbf{h}, \mathbf{J}, c, \texttt{BINARY})$

各先行研究との差分（赤枠部分）：

- 2 行目：ペナルティ係数 2.0
  - vs Venturelli/Carugno：理論のみ，アルゴリズムなし
  - vs Robinbux：試行錯誤ループ
- 3 行目：時間窓計算
  - vs Venturelli：理論のみ，アルゴリズムなし
  - vs Carugno/Robinbux：時間窓計算なし
- 6 行目：時間窓内のみ変数生成
  - vs Venturelli/Carugno：アルゴリズムなし
  - vs Robinbux：全時刻 0〜$T-1$

- 8, 10 行目：h_3 の正確な係数（線形項 $-\gamma$，offset γ）
  - vs Venturelli/Carugno：アルゴリズムなし
  - vs Robinbux：係数誤り，offset 欠落
- 15 行目：h_2 双方向判定
  - vs Venturelli/Carugno：アルゴリズムなし
  - vs Robinbux：一方向のみ

5.6.7 アルゴリズムの説明

このアルゴリズムは，第 5.3 節〜第 5.2 節で述べた個別のアルゴリズムを統合し，JSSP インスタンスから BQM オブジェクトを直接構築する．以下，各部分について説明する．

まず，入力と出力について説明する．

入力: JSSP インスタンス（ジョブ数 n，機械数 m，処理時間 $p_{j,k}$，機械割当 $\mu_{j,k}$）
出力: BQM オブジェクト

入力として，JSSP インスタンスの情報（ジョブ数，機械数，各操作の処理時間と使用機械）を受け取る．出力として，量子アニーリングに投入可能な BQM オブジェクトを返す．

次に，初期化処理について説明する．

$\alpha, \beta, \gamma \leftarrow 2.0$
各操作 $O_{j,k}$ の時間窓 $[ES_{j,k}, LS_{j,k}]$ を計算
$\mathbf{h} \leftarrow \{\}$, $\mathbf{J} \leftarrow \{\}$, $c \leftarrow 0$

まずペナルティ係数を理論値 2.0 に設定する．次に各操作の時間窓を計算する．最後に線形係数辞書 $\mathbf{h}$，二次係数辞書 $\mathbf{J}$，定数項 c を初期化する．

次に，変数生成と h_3 制約（一意性）の処理について説明する．

for 各操作 $O_{j,k}$ **do**
    for $t \in [ES_{j,k}, LS_{j,k}]$ **do**
        $\mathrm{idx}[j,k,t] \leftarrow$ 連番割り当て
        $\mathbf{h}[\mathrm{idx}[j,k,t]] \mathrel{+}= -\gamma$
    end for
    $\mathbf{J} \mathrel{+}= 2\gamma$（同一操作内の全ペア），$c \mathrel{+}= \gamma$
end for

各操作について，時間窓内の各時刻に変数を生成してインデックスを割り当てる．同時

に h_3 制約の係数を計算し，線形係数に $-\gamma$，同一操作内の変数ペアの二次係数に 2γ，定数項に γ を加算する．

次に，h_1 制約（順序）の処理について説明する．

```
for 各ジョブ j の連続操作 (O_{j,k}, O_{j,k+1}) do
    t + p_{j,k} > t' を満たすペアに J += α
end for
```

同一ジョブ内の連続する操作ペアについて，先行操作が後続操作の開始前に終了しない時刻の組み合わせにペナルティを設定する．

次に，h_2 制約（衝突回避）の処理について説明する．

```
for 同一機械の操作ペア (O_{j,k}, O_{j',k'}) do
    (t + p_{j,k} > t') ∧ (t' + p_{j',k'} > t) を満たすペアに J += β
end for
```

同一機械を使用する操作ペアについて，双方向のオーバーラップ条件を満たす時刻の組み合わせにペナルティを設定する．

最後に，BQM オブジェクトを構築して返す．

```
return BinaryQuadraticModel(h, J, c, BINARY)
```

計算した係数 $\mathbf{h}$, $\mathbf{J}$ および定数項 c から BQM オブジェクトを構築する．これにより，dwavebinarycsp ライブラリを介さずに，JSSP インスタンスから直接 BQM を生成できる．

5.7 まとめ

本章では，第 5.1 節で整理した未解決課題に対し，解決策を提示した．表 5.5 に，先行研究と本研究の状況を再掲する．右端の列は本研究でアルゴリズムを提示した節番号を示す．

本研究の貢献は以下の通りである：

1. **時間窓による変数削減**（第 5.3 節）：Venturelli らの時間窓計算を活用し，時間窓外の変数を除外することで，QUBO 行列のサイズを削減する．
2. h_2 **制約の双方向判定**（第 5.5 節）：Robinbux の一方向判定を修正し，Venturelli の定式化に従った双方向のオーバーラップ判定を実装する．

表 5.5　先行研究における手続きの状況（理論／アルゴリズム）と本研究の対応節

#	手続き	Venturelli	Carugno	Robinbux	本研究	対応節
1	JSSP インスタンスの読み込み	○/×	○/×	—/○	×/○	—
2	時間窓計算（ES, LS）	○/○	×/×	×/×	×/○	5.3
3	時間窓を使った変数生成	△[1]/×	×/×	×/×	×/○	5.3
4	h_3 制約（一意性）の係数	○/×	○[2]/×	×[3]/×	×/○	5.4
5	h_1 制約（順序）の係数	○/×	○[2]/×	—/○	×/○	5.6
6	h_2 制約（衝突回避）双方向	○/×	○[2]/×	×[4]/×	×/○	5.5
7	ペナルティ係数の設定	△[5]/×	○/×	×[6]/×	×/○	5.2
8	BQM 直接構築	△[7]/×	×[8]/×	—/○	×/○	5.6

各セルは「理論/アルゴリズム」の形式．○：明示的に記載，△：概念的言及，×：未記載，—：対象外
[1]「時間窓外の変数は除外可能」と概念的に言及，[2]Venturelli を引用，[3] 線形項の係数誤り・offset 項の欠落，[4] 一方向チェックのみ，[5]「A, B は十分大きく」と概念的に言及，[6] 試行錯誤による調整，[7]「Q 行列を構築」と概念的に言及，
[8]penaltymodel ライブラリに委託

3. h_3 **制約の正確な係数**（第 5.4 節）：一意性制約の展開に基づき，線形項・二次項・offset 項の係数を正確に設定する．
4. **ペナルティ係数の理論的設定**（第 5.2 節）：Carugno らの論文に基づく理論的な係数設定を採用し，試行錯誤的な調整を排除する．
5. **統合 BQM 構築手順**（第 5.6 節）：上記の改善点を統合し，再現可能な BQM 構築手順を提示する．

これにより，dwavebinarycsp の $O(2^k)$ 列挙処理（k は制約内変数数）を回避し，BQM 生成の高速化が期待できる．次章では，本手続きによる高速化効果を実験的に評価する．

第 6 章

実験・評価

6.1　はじめに

前章では，JSSP の QUBO 定式化において，dwavebinarycsp を使用せずに BQM を直接構築する手法を整理した．本章では，本手法の有効性を実験により検証する．

本章では以下の 2 つの実験を実施する：

1. **実験 1：BQM 生成時間の比較**（6.2 節）
   従来手法（Carugno らの手法）と本手法の BQM 生成時間を比較し，高速化効果を定量的に評価する．また，生成される BQM の同等性を検証し，第 4 章で測定したボトルネックが解消されたことを確認する．
2. **実験 2：QPU 性能評価**（6.3 節）
   BQM 生成のボトルネックが解消されたことで，QPU の性能を純粋に評価することが可能となった．複数の D-Wave マシン（Advantage2_system1.10, Advantage_system4.1, Advantage_system6.4;以降, 図表では各々 Advantage2_1.10, Advantage_4.1，Advantage_6.4 と略記）で同一問題を実行し，総実行時間と解品質を比較する．

6.2　実験 1：BQM 生成時間の比較

D-Wave への問題投入は，第 2 章の図 2.3 で示した Ocean SDK の実行フローに従う．図 6.1 に，本実験で使用した D-Wave 呼び出し部分のコードを示す．DWaveSampler で Leap 経由で QPU に接続し，EmbeddingComposite で論理変数を物理量子ビットに自動埋め込みした後，sample() メソッドで BQM を投入してサンプリング結果を取得する．

```python
    # ============================================================
    # 3. QPU (D-Wave) でサンプリング
    # ============================================================
    # DWaveSampler: Leap経由でD-Wave QPUに接続するサンプラー
    # solver_nameでQPUの種類を指定（例: "Advantage_system4.1"）
    if solver_name:
        base_sampler = DWaveSampler(solver=solver_name)
    else:
        base_sampler = DWaveSampler()  # デフォルトのQPUを使用

    # EmbeddingComposite: 論理変数を物理量子ビットに自動埋め込み
    sampler = EmbeddingComposite(base_sampler)

    # ------------------------------------------------------------
    # D-Waveへの問題投入（ここでLeap経由でQPUにBQMを送信）
    # ------------------------------------------------------------
    t0_qa = time.time()
    sampleset = sampler.sample(
        bqm,                            # 投入するBQM（二値二次モデル）
        chain_strength=chain_strength,  # チェーン強度（埋め込み用）
        num_reads=num_reads,            # サンプリング回数
        return_embedding=True,          # 埋め込み情報を返却
    )
    # sampleset: QPUから返却されたサンプリング結果
    t_qa_wallclock = time.time() - t0_qa

    best = sampleset.first
```

図 6.1　D-Wave 呼び出し部分のコード．DWaveSampler で Leap 経由で QPU に接続し，EmbeddingComposite で自動埋め込みを行った後，sample() メソッドで BQM を投入する．

6.2.1　目的

本実験の目的は以下の 2 点である：

1. 従来手法（Carugno らの手法）と本手法の BQM 生成時間を比較し，高速化効果を定量的に評価する
2. 第 4 章で測定したボトルネック（一意性制約の $O(2^T)$ 列挙処理）が解消されたことを確認する

6.2.2　実験手法

6.2.2.1　比較対象

以下の 2 つの手法を比較する：

- **従来手法**：Carugno らの手法（dwavebinarycsp を使用した BQM 生成）
- **本手法**：本研究で整理した直接 BQM 構築手法

6.2.2.2 問題インスタンス

$n \times n$ の正方インスタンス（n 個のジョブ，各ジョブが n 個の操作を持つ）を使用した．これは Carugno ら [5] の実験で使用されたものと同一の問題設定である．問題サイズは $n = 2, 3, \ldots, 20$ とし，各操作の処理時間は 1 に固定した．機械割当は巡回パターン（操作 $O_{j,k}$ を機械 $M_{((j-1+k-1) \mod m)+1}$ に割り当てる）に従う．

6.2.2.3 評価指標

評価指標は時間，高速化，同等性の 3 つの観点で設定する．

時間に関しては，BQM 生成時間（総時間）および第 4 章で定義した処理ステップ別時間（`t_one_start`，`t_precedence`，`t_share_machine`，`t_stitcher`）を測定する．

高速化に関しては，従来手法の処理時間を本手法の処理時間で除した高速化率を算出する．

同等性に関しては，両手法で生成される BQM が同一であることを検証する．

6.2.3 実験設定

実験に使用した環境およびパラメータを表 6.1 に示す．

ペナルティ係数 α, β, γ は Carugno ら [5] の設定に倣い，各制約違反で約 2 のエネルギー上昇となるよう統一した．これらの値は両手法で共通とし，生成される BQM の比較条件を揃えた．初回実行時のキャッシュ効果を排除するため，測定前にウォームアップ処理を実施した．

本実験の実行環境は第 4 章のボトルネック分析と同一である．

6.2.4 結果

本実験で測定する処理は, 第 5 章で整理した BQM 直接構築アルゴリズム（Algorithm 6）の各ステップである．図 6.2 に測定対象を示す．赤枠で囲んだ部分が本実験の測定対象であり，表 6.2 の各列に対応する．

表 6.1　実験設定

カテゴリ	項目	仕様/値
ハードウェア	CPU	Intel Core i7-10700（8 コア 16 スレッド）
	メモリ	32GB
	OS	Windows 11
ソフトウェア	Python	3.10.11
	dimod	0.12.21
	dwave-system	1.34.0
	dwavebinarycsp*	0.3.1
	penaltymodel*	1.3.0
	NumPy	2.2.6
パラメータ	timespan T	$n+1$
	α, β, γ	2.0
	反復回数	1 回

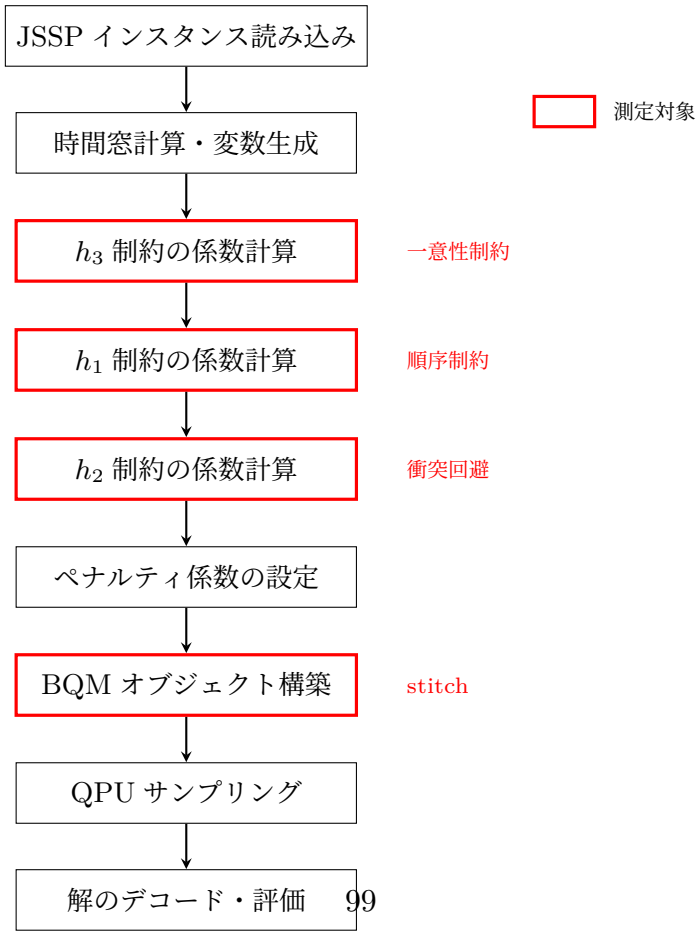


図 6.2　実験 1 の測定対象（赤枠：処理時間の測定箇所）

6.2.4.1 処理時間

問題サイズ $n = 2, 3, \ldots, 20$ の全てについて実験を実施した．第 4 章の表 4.4 に従来手法の処理時間を，表 6.2 に本手法の処理時間を示す．

表 6.2　本手法の処理時間（単位：秒）*

n	一意性制約	順序制約	衝突回避	stitch	合計
2	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.01
8	0.00	0.00	0.00	0.00	0.01
9	0.00	0.00	0.01	0.00	0.01
10	0.00	0.00	0.01	0.00	0.02
11	0.00	0.00	0.01	0.00	0.02
12	0.00	0.00	0.02	0.00	0.03
13	0.00	0.00	0.02	0.00	0.03
14	0.00	0.00	0.03	0.00	0.04
15	0.00	0.01	0.03	0.00	0.04
16	0.00	0.01	0.05	0.01	0.06
17	0.00	0.01	0.07	0.00	0.08
18	0.00	0.01	0.05	0.00	0.07
19	0.00	0.01	0.08	0.00	0.10
20	0.00	0.01	0.07	0.00	0.09

*0.00 は 0.01 秒未満を示す．

6.2.4.2 高速化率

表 6.3 に，従来手法と本手法の合計時間および高速化率を示す．

表 6.3　高速化率の比較

n	従来手法（秒）	本手法（秒）	高速化率
2	0.02	0.005	4×
5	0.21	0.003	70×
10	1.72	0.016	108×
15	13.00	0.042	310×
20	453.48	0.087	5212×

20×20 インスタンスにおいて，本手法は従来手法と比較して約 5200 倍（表では 5212 倍）の高速化を達成した．従来手法では約 7.5 分（453.48 秒）を要する BQM 生成が，本手法では約 0.087 秒で完了する．

従来手法では一意性制約の処理が全体の 95.4% を占めていた（表 4.4）のに対し，本手法では衝突回避制約の処理が支配的となっている．これは，本手法において一意性制約のボトルネックが完全に解消されたことを示している．

6.2.4.3　解の品質評価

表 6.4 に，本手法で生成した BQM を QPU で実行した結果を示す．

表 6.4　解の品質評価（QPU 実行結果）*

n	理論最適 makespan**	得られた makespan	最適解	best_energy
2	2	2	達成	0.0
3	3	3	達成	0.0
4	4	4	達成	0.0
5	5	5	達成	0.0
6	6	6	達成	0.0
7	7	7	達成	0.0
8	8	8	達成	0.0
9	9	9	達成	0.0
10	10	10	達成	0.0
11	11	11	達成	0.0
12	12	12	達成	0.0
13	13	13	達成	0.0
14	14	14	達成	0.0
15	15	15	達成	0.0
16	16	16	達成	0.0
17	17	17	達成	0.0
18	18	18	達成	0.0
19	19	19	達成	0.0
20	20	20	達成	0.0

* 正方形 JSSP（n ジョブ $\times n$ 機械，処理時間すべて 1）を使用.

** 本インスタンスでは理論最適 makespan $= n$．一般の JSSP では成り立たない.

全ての問題サイズにおいて，best_energy $= 0.0$（制約違反なし）かつ makespan $= n$（本インスタンスの理論最適値）が得られた．また，両手法で生成される BQM の変数数および係数値（数値誤差 10^{-10} 以内）が一致することを確認した．これにより，本手法は従来手法と一致する BQM を生成し，正しく最適解を導くことが確認された.

6.2.4.4　可視化（秒スケール）

図 6.3 に，問題サイズに対する BQM 生成時間の推移を秒スケールで示す.

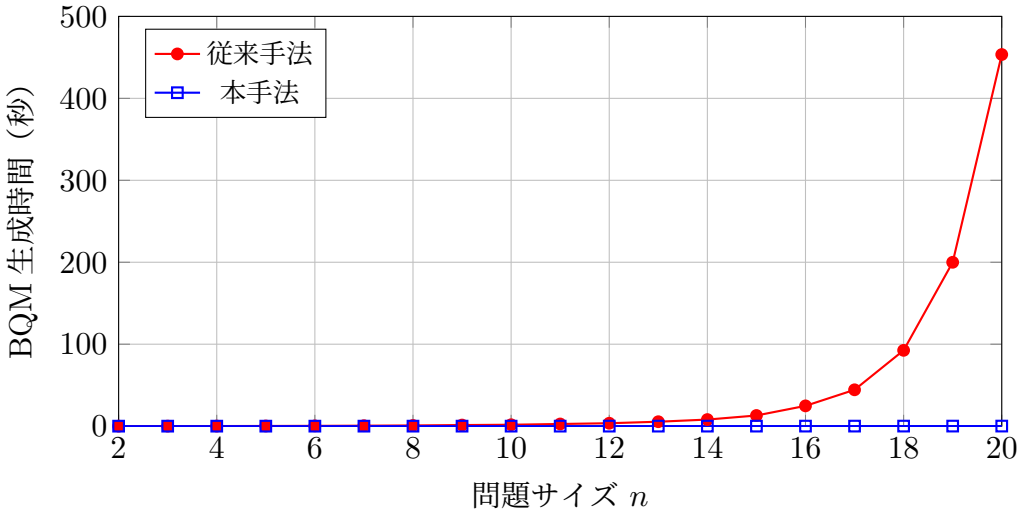


図 6.3　問題サイズと BQM 生成時間の関係（秒スケール）．従来手法は $n \geq 16$ で急激に増加する．

秒スケールでは，従来手法の BQM 生成時間が $n \geq 16$ で急激に増加する様子が明確に観察できる．一方，本手法の処理時間は全ての問題サイズで 0.1 秒以下にとどまっている．

6.2.4.5　可視化（対数スケール）

図 6.4 に，問題サイズに対する BQM 生成時間の推移を対数スケールで示す．

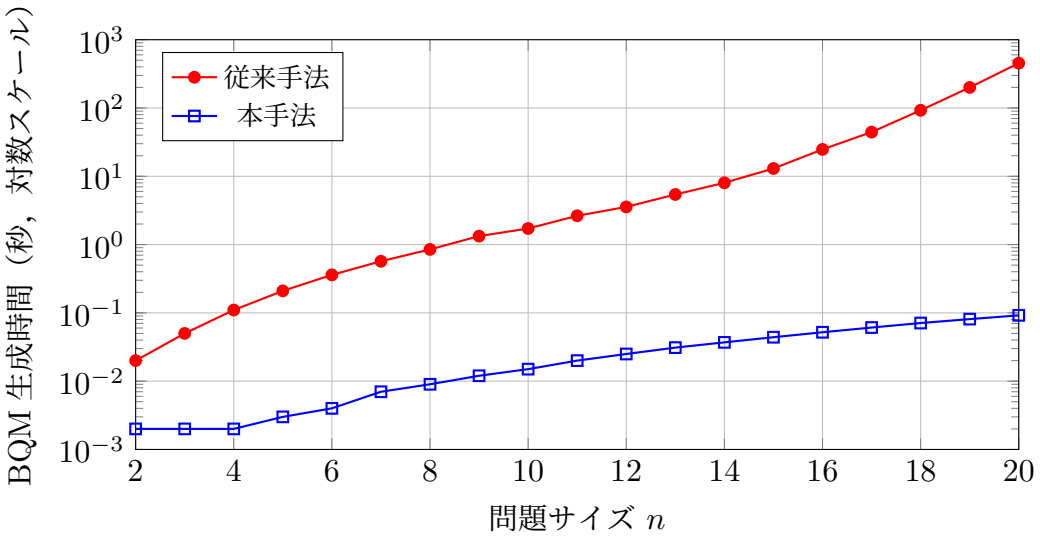


図 6.4　問題サイズと BQM 生成時間の関係（対数スケール）．従来手法は指数的に増加するのに対し，本手法は緩やかな増加にとどまる．

従来手法では，問題サイズの増加に伴い BQM 生成時間が指数的に増加する．これは，第 4 章で分析したように，一意性制約における `Constraint.from_func()` の $O(2^T)$ 列挙処理が支配的であるためである．一方，本手法では，BQM 生成時間の増加が緩やかである．これは，QUBO 係数を解析的に計算するため，$O(2^T)$ の列挙処理が発生せず，計算量が $O(n^2T)$（本研究の正方インスタンスでは $T = O(n)$ のため $O(n^3)$）に削減されるためである．

6.2.5 考察

6.2.5.1 高速化の要因分析

本手法における高速化の主要因は以下の通りである：

1. $O(2^T)$ **列挙の排除**：dwavebinarycsp の `Constraint.from_func()` による指数的構成列挙を回避し，QUBO 係数を多項式時間で解析的に計算する
2. **penaltymodel の排除**：penaltymodel によるペナルティ係数探索のオーバーヘッドが完全に排除される
3. **時間窓による変数削減**：時間窓計算により不要な変数を事前に除外し，処理対象を削減する

6.2.5.2 本手法の計算量

表 6.2 の実測データから，本手法の計算量を推定する．問題サイズ n に対する BQM 生成時間の増加率を表 6.5 に示す．

表 6.5　問題サイズの増加に対する BQM 生成時間の増加率

基準	比較	サイズ比	時間比（実測）	理論値
$n = 10$	$n = 20$	2 倍	$0.094/0.018 = 5.2$ 倍	$O(n^2)$: 4 倍, $O(n^3)$: 8 倍
$n = 10$	$n = 30$	3 倍	$0.324/0.018 = 18$ 倍	$O(n^2)$: 9 倍, $O(n^3)$: 27 倍

実測では，問題サイズが 2 倍で約 5 倍，3 倍で約 18 倍の時間増加となっている．これは $O(n^2)$（2 倍で 4 倍，3 倍で 9 倍）と $O(n^3)$（2 倍で 8 倍，3 倍で 27 倍）の中間であり，$O(n^3)$ に近い．

理論的には，本手法は QUBO 係数を解析的に計算する際に，変数ペア（$O(n^2)$ 個）に対して時間窓内の処理（$O(T)$）を行うため，$O(n^2T)$ 程度の計算量となると考えられる．

本研究で用いた正方形インスタンス（$T = 2n - 1$）では $O(n^3)$ となる．従来手法は $O(2^k)$（k は制約内変数数，一意性制約では $k = T$）であり，本研究設定では $T = O(n)$ のため指数的計算量となる．したがって，指数的増加から多項式的増加への本質的な改善が達成されている．

6.2.5.3 有効範囲

本手法は以下の条件で特に有効である：

- 問題サイズが大きい場合（$n \geq 10$）
- timespan が大きい場合（時間窓変数が多い場合）
- BQM 生成を繰り返し実行する場合（パラメータ探索など）

小規模問題（$n < 5$）では，Python の関数呼び出しオーバーヘッドが相対的に大きくなるため，高速化率は低下する．ただし，絶対的な処理時間は両手法とも十分に短い（1 秒未満）ため，実用上の問題はない．

6.3 実験 2：QPU 性能評価

6.3.1 目的

実験 1 により，BQM 生成のボトルネックが解消されたことが確認された．これにより，従来は測定困難であった QPU の性能を純粋に評価することが可能となった．

本実験の目的は以下の 2 点である：

1. 本手法で生成した BQM を用いて，QPU で最適解が得られることを確認する
2. 複数の D-Wave マシンで同一問題を実行し，マシン間の性能差を比較する

6.3.2 実験手法

6.3.2.1 実験の流れ

実験は以下の手順で実施した：

1. 本手法で BQM を生成（CPU 上で実行）
2. 生成した BQM を D-Wave QPU に送信
3. QPU でアニーリングを実行（num_reads 回サンプリング）
4. 得られたサンプルから最良解を抽出し，品質を評価

6.3.2.2 評価指標

表 6.6 に実験 2 で使用する評価指標を示す.

表 6.6 実験 2 の評価指標

カテゴリ	評価指標	説明
時間	BQM 生成時間	本手法による BQM 生成時間
	QPU アクセス時間	D-Wave が返す `timing["qpu_access_time"]` の値．QPU 上で QMI を実行している時間であり，キュー待ち・ネットワーク遅延は含まない [36]
解品質	最良エネルギー	得られた最良解のエネルギー値（0.0 で実行可能解）
	最適解達成	best_energy = 0.0（制約違反なし）かつ makespan = n（理論最適値）を達成したか．本研究は decision 版 JSSP を扱い，h_4（makespan 最小化項）は使用しない
	出現回数	最良解の出現回数（解の安定性を示す）

6.3.2.3 問題インスタンス

実験 1 と同一の正方インスタンス（$n \times n$）を使用した．問題サイズは $n = 2, 3, \ldots, 30$ とし，本手法により高速に BQM を生成できるため，従来より大きな問題サイズまで評価を拡張した.

6.3.3 実験環境

6.3.3.1 QPU 環境

D-Wave Leap クラウドサービスを使用し，3 種類の D-Wave マシンにアクセスした．使用したソルバの詳細を表 6.7 に示す.

Advantage2_system1.10 は次世代の Zephyr トポロジ [13] を採用した新型マシンであ

表 6.7　使用した D-Wave マシンの比較 *

項目	Advantage2_system1.10	Advantage_system4.1	Advantage_system6.4
トポロジ	Zephyr	Pegasus	Pegasus
量子ビット数	4,589	5,627	5,612
結合数	41,729	40,279	40,088
利用形態	D-Wave Leap クラウドサービス		

*D-Wave 公式ドキュメント（Per-QPU Solver Properties[30]）に基づく working graph 値（2026 年 1 月 19 日取得）．トポロジーの詳細は [37] を参照．working graph は fabricated（製造上限）より較正等で減少し得る．

り，2025 年 5 月 20 日に一般提供（GA）が開始された [31]．Advantage_system4.1 および Advantage_system6.4 は現行世代の Pegasus トポロジ [23] を採用している．

6.3.4　実験条件

QPU 実行に関するパラメータを表 6.8 に示す．

表 6.8　QPU 実行のパラメータ

パラメータ	値	説明
num_reads	1,000	1 回のジョブあたりのサンプリング回数
chain_strength	2.0	チェーン強度 [25]（エンベディング時の結合強度）
annealing_time	20 μs	アニーリング時間（デフォルト値）
試行回数	10	各問題サイズ・各マシンでの実行回数

6.3.5　結果

本実験で測定する処理は，本手法で生成した BQM を QPU で実行する部分である．図 6.5 に測定対象を示す．

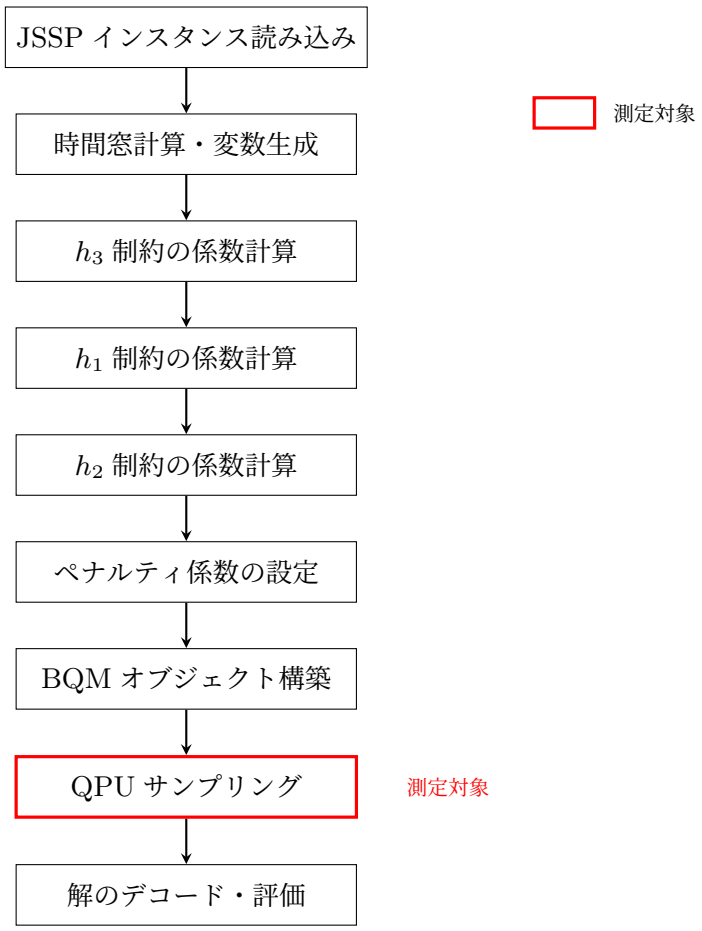


図 6.5　実験 2 の測定対象（赤枠：QPU 実行時間の測定箇所）

6.3.5.1　最適解達成状況

表 6.9 に，各問題サイズにおける最適解達成率（best_energy $= 0.0$ となった試行の割合）を 3 種類のマシンで比較して示す．

$n \leq 13$ の問題サイズでは，3 種類のマシンすべてで全試行において最適解が達成された．$n = 14$ 以降では最適解達成が不安定となり始め，$n \geq 26$ では達成率が大幅に低下する．Advantage2_system1.10（Zephyr トポロジ）が最も安定した結果を示し，$n = 30$ においても 20% の試行で最適解を達成した．一方，Pegasus トポロジを採用する Advantage_system4.1 および Advantage_system6.4 は，$n \geq 28$ で最適解達成率が著しく低下した．なお，一部のマシンで $n = 28$ より $n = 29$ の達成率が高くなる非単調な挙動が観測されている（例：Advantage_system6.4 で $n = 28$ が 2/10，$n = 29$ が 4/10）．これは量子アニーリングの確率的性質およびマイナー埋め込みの問題依存性に起因すると考えられ，各条件 10 回という限られた試行数では統計的揺らぎが結果に現れやすい．

表 6.9　最適解達成回数の比較（各 10 回試行）

問題サイズ n	Advantage2_1.10	Advantage_4.1	Advantage_6.4
2	10/10	10/10	10/10
3	10/10	10/10	10/10
4	10/10	10/10	10/10
5	10/10	10/10	10/10
6	10/10	10/10	10/10
7	10/10	10/10	10/10
8	10/10	10/10	10/10
9	10/10	10/10	10/10
10	10/10	10/10	10/10
11	10/10	10/10	10/10
12	10/10	10/10	10/10
13	10/10	10/10	10/10
14	9/10	10/10	10/10
15	9/10	10/10	10/10
16	6/10	10/10	10/10
17	10/10	10/10	10/10
18	9/10	10/10	10/10
19	10/10	10/10	10/10
20	10/10	9/10	10/10
21	10/10	8/10	9/10
22	7/10	8/10	8/10
23	7/10	8/10	7/10
24	6/10	7/10	9/10
25	10/10	7/10	3/10
26	6/10	5/10	4/10
27	4/10	2/10	2/10
28	3/10	0/10	2/10
29	3/10	1/10	4/10
30	2/10	0/10	0/10

図 6.6 に，問題サイズに対する最適解達成回数（10 回試行中）の推移を示す．$n \leq 13$ では全マシンで 10 回中 10 回達成を維持するが，$n \geq 20$ 付近から差が生じ始め，$n \geq 26$ で急激に低下する様子が視覚的に確認できる．

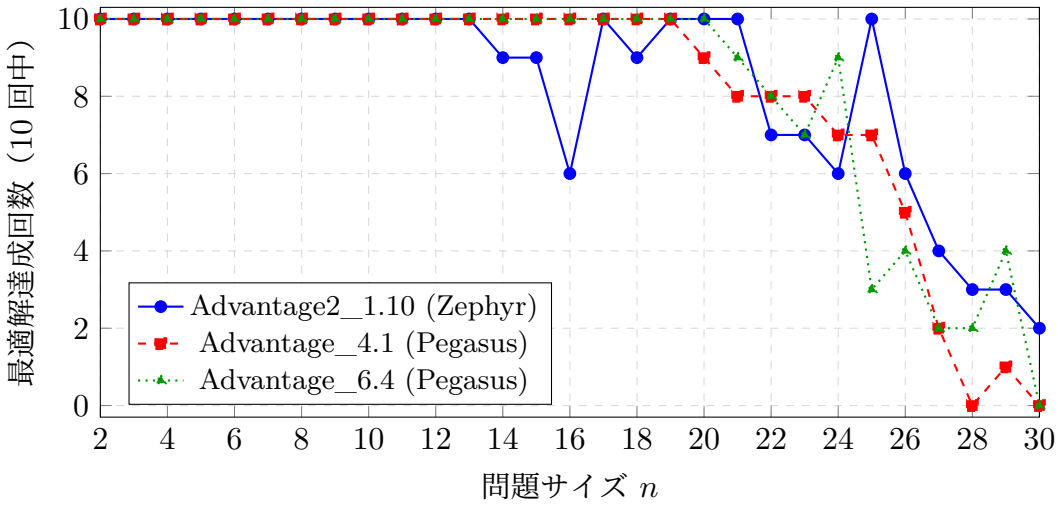


図 6.6　問題サイズに対する最適解達成回数の推移（$n = 2 \sim 30$，各 10 回試行）

6.3.5.2　第 4 章の結果との比較

第 4 章では，Carugno らの手法で生成した BQM を Advantage_system2.1 で実行し，$n = 2 \sim 20$ で最適解が得られることを確認した（表 4.5）．本実験では，本手法で生成した BQM を用いて同様の結果が得られることを確認した．

表 6.10 に，$n = 20$ における両手法の解品質を比較する．

表 6.10　$n = 20$ における解品質の比較

項目	従来手法（第 4 章）	本手法（本実験）
best_energy	0.0	0.0
makespan	20（最適）	20（最適）
最適解達成	達成	達成

両手法で同一の解品質が得られており，本手法による高速化が解品質に影響を与えていないことが確認された．

6.3.5.3 BQM 生成時間と QPU 実行時間の比較

表 6.11 に，本手法における BQM 生成時間と QPU 実行時間の内訳を示す（Advantage2_system1.10 での測定値，10 回試行の平均）.

表 6.11　BQM 生成時間と QPU 実行時間の比較（Advantage2_system1.10，10 回試行の平均）

問題サイズ n	BQM 生成（秒）	QPU アクセス（秒）	比率
2	0.002	0.15	0.01
3	0.002	0.16	0.01
4	0.003	0.16	0.02
5	0.003	0.17	0.02
6	0.004	0.17	0.02
7	0.006	0.18	0.03
8	0.010	0.18	0.06
9	0.012	0.18	0.06
10	0.018	0.18	0.10
11	0.020	0.19	0.11
12	0.028	0.19	0.14
13	0.032	0.20	0.16
14	0.042	0.19	0.21
15	0.043	0.19	0.22
16	0.062	0.20	0.31
17	0.071	0.20	0.35
18	0.078	0.20	0.39
19	0.089	0.21	0.42
20	0.094	0.20	0.46
21	0.118	0.20	0.57
22	0.142	0.21	0.66
23	0.137	0.21	0.65
24	0.158	0.22	0.73
25	0.171	0.21	0.80
26	0.188	0.21	0.89
27	0.211	0.21	1.00
28	0.294	0.21	1.38
29	0.274	0.21	1.29
30	0.324	0.22	1.50

本手法では，$n = 26$ まで BQM 生成時間が QPU アクセス時間を下回っている（比率 < 1.0）．$n = 27$ で比率が 1.00 となり両者が拮抗し，$n \geq 28$ では BQM 生成時間が QPU アクセス時間を上回るが，$n = 30$ でも約 0.3 秒であり実用上問題のないレベルである．

従来手法（Carugno らの手法を再現実装）では $n = 20$ の BQM 生成に約 453 秒を要していたのに対し，QPU アクセス時間は約 0.2 秒である．従来手法では BQM 生成時間が QPU 実行時間の 2000 倍以上となり，QPU の性能評価が実質的に不可能であった．本手法により，この比率は $n = 20$ で 0.46 と大幅に改善された．

6.3.5.4　Carugno らの実験との比較

表 6.12 に，Carugno らの実験 [5] と本研究の差分を示す．

表 6.12　Carugno らの実験と本研究の比較

項目	Carugno 法 †	本研究
評価した問題サイズ	$n = 2 \sim 20$	$n = 2 \sim 30$
BQM 生成時間（$n = 20$）	453 秒	約 0.09 秒 ‡
QPU 性能限界の特定	未実施	$n \geq 26$ で不安定と特定
比較したマシン数	1 種類	3 種類

†Carugno 法の列は，本研究が Carugno らの手法を再現実装し，本研究の環境で計測した値である．Carugno 原論文 [5] が報告した値ではない．

‡ 実験 1（表 6.3）では 0.087 秒，実験 2（表 6.11）では 0.094 秒．試行間のばらつきおよび計測条件の違いによる．

Carugno らの実験では $n = 2 \sim 26$ を対象としたが，BQM 生成のボトルネックにより $n = 26$ では 8 時間以上を要した．QA 単独では $n \leq 21$ まで最適解（エネルギー 0）に到達し，Reverse Annealing により一部のサイズ（size 16, 20, 22 等）で改善が報告されている．本研究では，本手法によりこのボトルネックを解消し，$n = 30$ までの評価を実現した．これにより，先行研究より広いサイズ領域で系統的な成功率曲線を取得し，複数の QPU 世代（Advantage2 を含む）で $n \geq 26$ 付近に性能限界があることを確認した．

6.3.6　考察

本実験により，以下の知見が得られた：

1. **BQM ボトルネックの解消**：本手法により，$n = 30$ でも BQM 生成は約 0.3 秒と

なり，QPU の性能を純粋に評価できるようになった．これにより，BQM 生成時間が QPU 実行時間を大幅に超過するという従来の問題が解消され，QPU の本来の性能を純粋に評価できる環境が整った．

2. **QPU 性能限界の特定**：$n \leq 15$ では全マシンで安定して最適解が得られる．$n = 16 \sim 25$ では達成率が 80–100% で推移し，$n \geq 26$ で急激に低下することを特定した．このような性能限界の特定は，BQM 生成がボトルネックとなっていた従来手法では不可能であった．Carugno らの実験 [5] では $n = 26$ までを対象としたが，本研究では $n = 30$ までの評価を実現し，QPU の実用的な限界点を明らかにした．

$n \geq 26$ で最適解達成率が低下する要因として，問題サイズの増大に伴い QPU への埋め込みが困難になることが考えられる．具体的には，QUBO 変数数が $O(n^2T)$ で増加するため，QPU のトポロジ上で論理変数 1 つを表現するために必要な物理量子ビットのチェーンが長くなる．チェーンが長いほどチェーン内の量子ビットが異なる値を取るチェーンブレイクが発生しやすくなり，解の劣化につながると考えられる．これは量子アニーリングのハードウェア制約に起因する本質的な問題であり，現行の QPU では $n \geq 26$ 程度が実用的な限界であることを示唆している．

この限界を克服するためには，いくつかのアプローチが考えられる．第一に，より多くの量子ビットとより高い接続性を持つ次世代 QPU の登場を待つことである．D-Wave は Zephyr トポロジーを採用した Advantage2 を 2025 年 5 月 20 日に一般提供開始しており [31]（本研究でも使用），これにより埋め込み効率の向上が期待される．第二に，問題分割やハイブリッド手法の適用である．大規模問題を複数の小規模部分問題に分割し，それぞれを QPU で解くことで，より大きな問題への対応が可能となる可能性がある．第三に，エンベディング手法の改良である．より効率的なエンベディングアルゴリズムの開発により，同一問題でもより短いチェーン長での埋め込みが実現できれば，解品質の向上が期待できる．

6.4 課題

本実験にはいくつかの課題がある．

まず，構成上の課題として，本実験では $n \times n$ の正方インスタンスのみを対象としている．一般的な JSSP インスタンス（ジョブ数と機械数が異なる場合）への適用性は直接検証していない．ただし，本手法の理論的基盤は一般の JSSP に適用可能である．また，す

べての操作の処理時間を 1 に固定しており，処理時間が不均一な場合は時間窓の形状が変化し，結果が異なる可能性がある．

次に，測定に関する課題として，処理時間の測定には Python の `time.time()` を使用しており，ミリ秒以下の精度に限界がある．小規模問題での測定値には誤差が含まれる可能性がある．また，量子アニーリングは確率的プロセスであり，同一問題でも実行ごとに結果が変動する．本実験では各条件 10 回の試行で統計的な傾向を把握したが，特に性能が急変する境界領域（$n = 25 \sim 27$ 付近）では試行回数が十分でない可能性がある．より厳密な評価には，試行回数の増加や信頼区間の算出が必要である．

最後に，一般化に関する課題として，本実験は特定の CPU 環境および D-Wave QPU で実施しており，他の環境では性能が異なる可能性がある．D-Wave Ocean SDK のバージョンアップにより従来手法の性能が変化する可能性があり，QPU のキャリブレーション状態によって結果が変動する可能性もある．

6.5 まとめ

本章では，本手法の有効性を 2 つの実験により検証した．

実験 1：BQM 生成時間の比較では，本手法と従来手法（Carugno らの手法）の BQM 生成時間を比較した．20×20 インスタンスにおいて，本手法は従来手法と比較して約 5200 倍の高速化を達成した．生成される BQM は従来手法と一致しており，高速化による解品質の劣化は観測されなかった．これにより，第 4 章で特定した一意性制約のボトルネックが完全に解消されたことを確認した．

実験 2：QPU 性能評価では，BQM 生成のボトルネックが解消されたことで可能となった QPU の性能評価を実施した．3 種類の D-Wave マシンで $n = 2 \sim 30$ の正方インスタンスを各 10 回試行した結果，$n \leq 13$ では全マシンで 100% の最適解達成率を示し，$n = 14 \sim 25$ では 60–100% で推移し，$n \geq 26$ で急激に低下することを確認した．BQM 生成時間は $n = 30$ でも約 0.3 秒であり，QPU の性能を純粋に評価できる状態が実現された．

第 7 章

結言

7.1 まとめ

本研究では，ジョブショップスケジューリング問題（JSSP）への量子アニーリング適用において，BQM 生成のボトルネックを分析し，先行研究で散在していた手法を整理することでその解消を図った．

第 4 章では，Carugno らの手法を再現実装し，処理時間を詳細に計測した．その結果，dwavebinarycsp ライブラリ [15] 内部の `Constraint.from_func()` メソッドが $O(2^k)$ の列挙処理（k は制約が扱う変数数）を行うことがボトルネックの原因であることを特定した．一意性制約（h_3）では $k = T$（時刻変数数）となるため，この処理が特に顕著であり，20×20 インスタンスで約 7.5 分を要し，これ以上の問題サイズでは実質的に BQM 生成が困難となることを確認した．

第 5 章では，dwavebinarycsp を経由せずに BQM を直接構築する手法を整理した．この手法は，Venturelli らの論文では明示されていなかった 4 つの手続き，すなわち (a) 変数インデックス割り当て，(b)QUBO 係数への変換，(c) 係数の累積，(d)BQM オブジェクト構築を明示化したものである．本手法では，ペナルティ関数を解析的に展開して QUBO 係数を直接計算するため，$O(2^T)$ の列挙処理が不要となる．

第 6 章では，本手法の有効性を 2 つの実験により検証した．BQM 生成時間の比較実験では，20×20 インスタンスにおいて本手法が従来手法と比較して約 5200 倍の高速化を達成することを確認した．生成される BQM は従来手法と一致しており，高速化による解品質の劣化は観測されなかった．QPU 性能評価実験では，複数の D-Wave マシンで同一問題を実行し，本手法により生成した BQM が QPU 上で正しく最適解を導出できることを確認した．

以上より，本研究の貢献は以下の 5 点にまとめられる：

1. dwavebinarycsp のボトルネック原因を特定し，一意性制約における $O(2^T)$ の列挙処理が原因であることを明らかにした
2. Venturelli らの論文では明示されていなかった 4 つの手続き (a)〜(d) を明示化し，BQM を直接構築する手法を整理した
3. 整理した手法により，BQM 生成の計算量を $O(2^T)$ から $O(n^3)$ へ削減した
4. 高速化により，従来（Carugno ら）の $n = 26$ から $n = 30$ まで評価範囲を拡張した
5. Carugno らの 1 種類に対し 3 種類の D-Wave マシンで実験を拡張し，$n \geq 26$ 付近で QPU 性能が急激に低下することを確認した

7.2 今後の課題

本研究の成果を踏まえ，以下の課題に取り組む予定である．

まず，さらに大規模な問題サイズでの QPU 性能評価が挙げられる．本研究では $n = 30$ までの評価を行った．$n \geq 26$ 付近で最適解達成率が急激に低下し，QPU の性能限界の傾向が十分に確認できたため $n = 30$ を上限としたが，埋め込み限界に到達するサイズは未特定である．今後は 35×35 や 40×40 等のさらに大規模なインスタンスに対して評価を拡張し，QPU 埋め込み限界に到達するサイズを特定することが課題である．

次に，QPU 埋め込み限界の調査が挙げられる．現在の D-Wave QPU は物理量子ビット数に制限があるため，一定サイズ以上の問題は埋め込みが不可能となる．Venturelli らの時間窓計算による変数削減が QPU 埋め込み可能サイズに与える効果を定量評価し，Zephyr トポロジにおける埋め込み限界を特定することが課題である．

また，解品質の詳細評価も重要である．本研究では最適解の導出可否を評価したが，大規模問題では最適解が得られない場合の解品質（makespan）の評価も必要である．古典ソルバ（OR-Tools 等）との比較により，QPU の実用的な適用範囲を明らかにすることが課題である．

最後に，他の組合せ最適化問題への適用が挙げられる．本研究で整理した直接 BQM 構築アプローチは，dwavebinarycsp を経由することで生じるボトルネックを回避する一般的な手法である．巡回セールスマン問題（TSP）やグラフ彩色問題など，他の組合せ最適化問題への適用可能性を検討することが課題である．

謝辞

本研究プロジェクトを進めるにあたり，研究の方向性から論文の執筆に至るまで, 丁寧なご指導と温かい励ましをいただいた主指導教員の井口 寧 教授，高野 恵輔 助教に深く感謝申し上げます．また，中間審査・最終審査において有益なご助言をいただいた副指導教員，副テーマ指導教員の先生方，北陸地区ハードウェア関連研究室合同セミナーにてご助言を頂いた福井大学, 金沢大学の先生方, および聴講頂いた皆様方, 本研究で使用した D-Wave 量子アニーリングマシンへのアクセスおよび技術支援にご協力いただいた D-Wave Japan 株式会社様，各種手続き, 長期履修制度の利用等でご支援いただいた北陸先端科学技術大学院大学の情報社会基盤研究センター, 関係各所の皆様, そして, 長らくゼミや研究室での議論を通じて刺激を頂いた井口研究室の皆様方に，心より感謝申し上げます．

社会人院生として長らく業務との兼業を承認いただき，研究テーマ，プログラミング技術，ソフトウェアの設計手法について多大なご助言, ご支援をいただいた小林様，楊様，申様，お仕事全般のご支援をいただいた田中様に感謝申し上げます．

また，修士論文の審査直前に遭った交通事故に際し，救命措置を行っていただいた石川県消防本部の皆様，治療にあたっていただいた石川県立中央病院および能美市立病院の皆様，事故処理にあたっていただいた石川県警の皆様，事故現場で対応に助力いただいた谷口君に深く感謝いたします．さらに，特例措置をいただいた北陸先端科学技術大学院大学の皆様，審査にあたっていただいた先生方，交通事故に係る各種手続きや退院後のご助言をいただいた井口 寧 教授，IEICE 総合大会での代理発表をお引き受けいただいた高野 恵輔 助教に，重ねて御礼申し上げます．

研究業績

国内学会発表

1. 河本 大貴，井口 寧，“巡回セールスマン問題における D-Wave 量子アニーリングマシンの有効性調査,” 2024 年度 電気・情報関係学会北陸支部連合大会，G11-3，1 page，金沢工業大学，Sep. 14, 2024.
2. 河本 大貴，高野 恵輔，井口 寧，“量子アニーリングによる JSSP 求解のための GPU を用いた BQM 生成高速化,” 電子情報通信学会 総合大会 2026，D-6-05，1 page，九州産業大学，Mar. 12, 2026.

参考文献

[1] A. Lucas, “Ising formulations of many NP problems,” Frontiers in Physics, vol. 2, Article 5, 2014.

[2] S. Jain, “Solving the Traveling Salesman Problem on the D-Wave Quantum Computer,” Frontiers in Physics, vol. 9, Article 760783, 2021.

[3] 河本大貴，井口寧，“巡回セールスマン問題における D-Wave 量子アニーリングマシンの有効性調査,” 2024 年度 電気・情報関係学会北陸支部連合大会，G11-3，Sep. 2024.

[4] D. Venturelli, D. J. J. Marchand, and G. Rojo, “Quantum annealing implementation of job-shop scheduling,” arXiv preprint arXiv:1506.08479, 2016.

[5] C. Carugno, M. Ferrari Dacrema, and P. Cremonesi, “Evaluating the job shop scheduling problem on a D-wave quantum annealer,” Scientific Reports, vol. 12, no. 1, p. 6539, 2022.

[6] Robinbux, “JSP-Quantum-Annealing,” GitHub Repository, `https://github.com/Robinbux/JSP-Quantum-Annealing`, 2020. (Accessed: 2025-01-11).

[7] B. Denkena, F. Schinkel, J. Pirnay, and S. Wilmsmeier, “Quantum algorithms for process parallel flexible job shop scheduling,” CIRP Journal of Manufacturing Science and Technology, vol. 33, pp. 100–114, 2021.

[8] D. Amaro, M. Rosenkranz, N. Fitzpatrick, K. Hirano, and M. Fiorentini, “A case study of variational quantum algorithms for a job shop scheduling problem,” EPJ Quantum Technology, vol. 9, no. 10, 2022.

[9] K. Kurowski, J. Węglarz, M. Subocz, R. Różycki, and G. Waligóra, “Hybrid Quantum Annealing Heuristic Method for Solving Job Shop Scheduling Problem,” in ICCS 2020, LNCS 12142, pp. 502–515, 2020.

[10] Z. Zhang, G. Lo Bianco, and J. C. Beck, “Solving job-shop scheduling problems

with QUBO-based specialized hardware," in ICAPS 2022, vol. 32, pp. 404–412, 2022.

[11] D-Wave Systems Inc., "D-Wave System Documentation," https://docs.dwavesys.com/, 2024. (Accessed: 2025-01-02).

[12] D-Wave Systems Inc., "dimod: A shared API for QUBO/Ising samplers," https://docs.dwavequantum.com/en/latest/ocean/api_ref_dimod/models.html, 2024. (Accessed: 2025-01-02).

[13] K. Boothby, A. D. King, and J. Raymond, "Zephyr Topology of D-Wave Quantum Processors," D-Wave Technical Report 14-1056A-A, 2021.

[14] C. McGeoch, P. Farré, and K. Boothby, "The D-Wave Advantage2 Prototype," D-Wave Technical Report 14-1063A-A, 2022.

[15] D-Wave Systems Inc., "dwavebinarycsp: Map constraint satisfaction problems to binary quadratic models," GitHub Repository, https://github.com/dwavesystems/dwavebinarycsp, 2024. (Accessed: 2025-01-05).

[16] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse Ising model," Physical Review E, vol. 58, no. 5, pp. 5355–5363, 1998.

[17] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda, "A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem," Science, vol. 292, no. 5516, pp. 472–475, 2001.

[18] M. W. Johnson, M. H. S. Amin, S. Gildert, et al., "Quantum annealing with manufactured spins," Nature, vol. 473, no. 7346, pp. 194–198, 2011.

[19] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[20] M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 5th ed. Springer, 2016.

[21] F. Glover, G. Kochenberger, and Y. Du, "Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models," 4OR-Q J Oper Res, vol. 17, pp. 335–371, 2019.

[22] C. C. McGeoch and C. Wang, "Experimental evaluation of an adiabatic quantum system for combinatorial optimization," in Proc. ACM International Conference on Computing Frontiers (CF '13), pp. 1–11, 2013.

[23] D-Wave Systems Inc., "Pegasus Topology," D-Wave Technical Documentation, https://docs.dwavesys.com/docs/latest/c_gs_4.html, 2024. (Ac-

cessed: 2025-01-10).

[24] D-Wave Systems Inc., "Minor-Embedding," D-Wave Technical Documentation, https://docs.dwavesys.com/docs/latest/c_gs_7.html, 2024. (Accessed: 2025-01-10).

[25] D-Wave Systems Inc., "Chain Strength," D-Wave Technical Documentation, https://docs.dwavesys.com/docs/latest/c_qpu_chains.html, 2024. (Accessed: 2025-01-10).

[26] J. Carlier and E. Pinson, "Adjustment of heads and tails for the job-shop problem," European Journal of Operational Research, vol. 78, no. 2, pp. 146–161, 1994.

[27] T. F. Rønnow, Z. Wang, J. Job, et al., "Defining and detecting quantum speedup," Science, vol. 345, no. 6195, pp. 420–424, 2014.

[28] V. Choi, "Minor-embedding in adiabatic quantum computation: I. The parameter setting problem," Quantum Information Processing, vol. 7, no. 5, pp. 193–209, 2008.

[29] D-Wave Systems Inc., "Ocean Software Documentation," https://docs.ocean.dwavesys.com/, 2024. (Accessed: 2025-01-10).

[30] D-Wave Quantum Inc., "Per-QPU Solver Properties," D-Wave Documentation, https://docs.dwavequantum.com/en/latest/quantum_research/solver_properties_specific.html, 2025. (Accessed: 2026-01-19).

[31] D-Wave Quantum Inc., "D-Wave Announces General Availability of Advantage2 Quantum Computer," Press Release, May 20, 2025. https://www.dwavequantum.com/company/newsroom/press-release/d-wave-announces-general-availability-of-advantage2-quantum-computer-its-most- (Accessed: 2026-01-19).

[32] D-Wave Quantum Inc., "dwavebinarycsp API Reference," D-Wave Documentation, https://docs.dwavequantum.com/en/latest/ocean/api_ref_binarycsp.html, 2025. (Accessed: 2026-01-19).

[33] D-Wave Systems Inc., "dwave-ocean-sdk," Python Package Index (PyPI), https://pypi.org/project/dwave-ocean-sdk/, 2025. (Accessed: 2026-01-19).

[34] D-Wave Systems Inc., "dwave-ocean-sdk Releases," GitHub, https://github.com/dwavesystems/dwave-ocean-sdk/releases, 2025. (Accessed: 2026-01-19).

[35] D-Wave Quantum Inc., "Solver Parameters," D-Wave Documentation,

https://docs.dwavequantum.com/en/latest/quantum_research/solver_parameters.html, 2025. (Accessed: 2026-01-19).

[36] D-Wave Quantum Inc., "Operation and Timing," D-Wave Documentation, https://docs.dwavequantum.com/en/latest/quantum_research/operation_timing.html, 2025. (Accessed: 2026-01-19).

[37] D-Wave Quantum Inc., "QPU Topologies," D-Wave Documentation, https://docs.dwavequantum.com/en/latest/quantum_research/topologies.html, 2025. (Accessed: 2026-01-19).

[38] D-Wave Quantum Inc., "Zephyr Topology of D-Wave Quantum Processors," Technical White Paper, https://www.dwavequantum.com/media/2uznec4s/14-1056a-a_zephyr_topology_of_d-wave_quantum_processors.pdf, 2021. (Accessed: 2026-01-19).

[39] D-Wave Systems Inc., "dwavebinarycsp Releases," GitHub, https://github.com/dwavesystems/dwavebinarycsp/releases, 2024. (Accessed: 2026-01-19).

付録 A

具体例

A.1　第 2 章の具体例

A.1.1　基本制約の具体例

前節の 2 ジョブ ×2 機械の例で考える．

順序制約：$J_1 = (O_{1,1}, O_{1,2})$ では，$O_{1,1}$ の終了後に $O_{1,2}$ を開始する必要がある：

$$S_{1,1} + 3 \leq S_{1,2}$$

$S_{1,1} = 0$ なら，$S_{1,2} \geq 3$ となる．

資源競合制約：$O_{1,1}$ と $O_{2,2}$ は同じ機械 M_1 を使用するため，同時に実行できない：

$$S_{1,1} + 3 \leq S_{2,2} \quad \text{または} \quad S_{2,2} + 4 \leq S_{1,1}$$

$O_{1,1}$ を先に処理するなら $S_{2,2} \geq 3$，$O_{2,2}$ を先に処理するなら $S_{1,1} \geq 4$ となる．

A.2　第 4 章の具体例

本節では，第 4 章で述べた各制約の登録処理について，詳細な手順を説明する．

A.2.1　制約の登録方法

dwavebinarycsp の `add_constraint` メソッドには，以下の 2 つの形式で制約を渡すことができる：

- **関数を渡す場合**：CSP 内部で 2^n 通りを列挙し，関数で判定して許可構成を抽出

- **集合を渡す場合**：その集合をそのまま許可構成として使用（列挙不要）

Carugno らは，一意性制約には関数渡し，順序制約と衝突回避制約には集合渡しを採用している．以下では，各制約がどのように登録されるかを説明する．

A.2.2 一意性制約の登録

第 3.2 節で述べた一意性制約は，各操作について「正確に 1 つの時刻変数のみが 1 となる」ことを要求する：

$$\sum_{t=0}^{T-1} x_{j,k,t} = 1 \quad \forall j, k \tag{A.1}$$

Carugno らは，この制約を以下の判定関数として表現している：

Listing A.1　一意性制約の判定関数

```
1 def sum_to_one(*args):
2     return sum(args) == 1
```

この判定関数は，引数の合計が 1 のときのみ `True` を返す．

ある操作 $O_{j,k}$ に対して，その操作が開始しうる全時刻の変数 $\{x_{j,k,0}, x_{j,k,1}, \ldots, x_{j,k,T-1}\}$（$T$ 個）をまとめて判定関数に渡し，CSP に登録する．これを全 n^2 個の操作について繰り返す：

Listing A.2　一意性制約の登録

```
1 for task_times in all_task_times:  # for each operation
2     csp.add_constraint(sum_to_one, task_times)
```

結果として，n^2 個の一意性制約が登録され，各制約は T 個の変数を参照する．

図 A.1 に 1 つの操作に対する登録処理を，図 A.2 に全体のループ構造を示す．

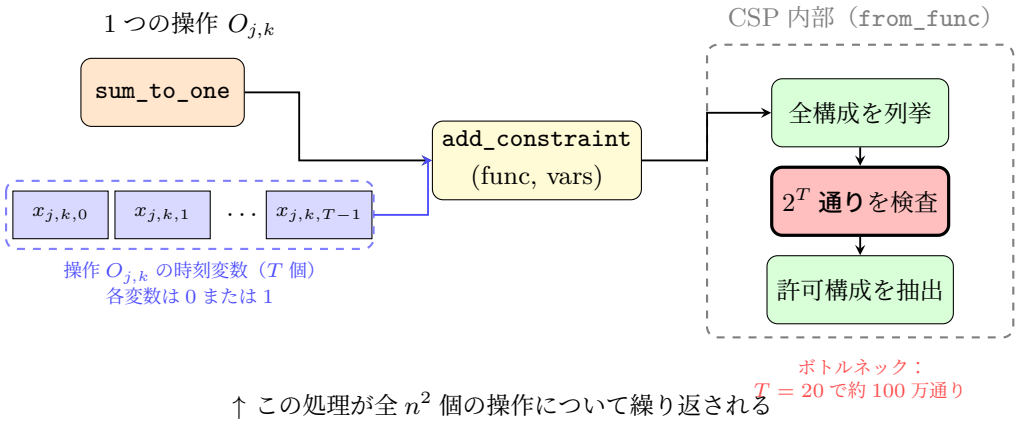


図 A.1　一意性制約の登録（1 つの操作あたり）．操作 $O_{j,k}$ に対する T 個の時刻変数と判定関数が CSP に渡され，内部で 2^T 通りの全構成が列挙・検査される．

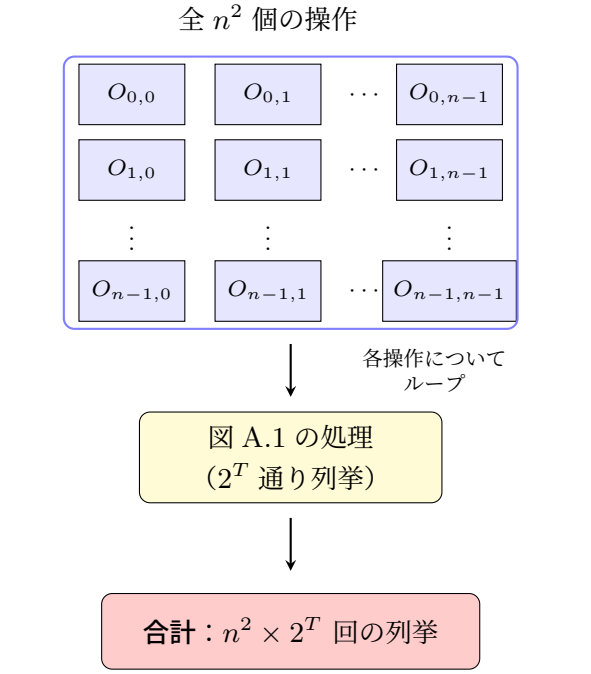


図 A.2　一意性制約登録の全体ループ．全 n^2 個の操作それぞれに対して図 A.1 の処理が実行され，合計 $n^2 \times 2^T$ 回の列挙が発生する．

A.2.3　順序制約の登録

第 3.2 節で述べた順序制約は，同一ジョブ内の連続する操作について「前の操作が完了してから次の操作が開始する」ことを要求する：

$$t + p_{j,k} \leq t' \quad (x_{j,k,t} = 1 \text{ かつ } x_{j,k+1,t'} = 1 \text{ のとき}) \tag{A.2}$$

Carugno らは，この制約を許可される構成の集合として表現している：

Listing A.3　順序制約の許可構成

```
valid_edges = {(0, 0), (1, 0), (0, 1)}  # (1, 1) is forbidden
```

一意性制約では判定関数を渡したが，ここでは許可構成の集合を直接渡している．この集合の各要素 $(x_{j,k,t}, x_{j,k+1,t'})$ は，2 つの時刻変数の値の組み合わせを表す：

- $(0, 0)$：前の操作が時刻 t に開始しない，次の操作も時刻 t' に開始しない → 許可
- $(1, 0)$：前の操作が時刻 t に開始する，次の操作は時刻 t' に開始しない → 許可
- $(0, 1)$：前の操作が時刻 t に開始しない，次の操作が時刻 t' に開始する → 許可
- $(1, 1)$：両方が同時に 1 → **禁止**（$t' < t + p$ のとき順序違反）

ただし，$t' \geq t + p$ の場合は $(1, 1)$ も許可されるため，制約の登録対象は $t' < t + p$ の範囲に限定される．

連続する 2 つの操作 $O_{j,k}$ と $O_{j,k+1}$ に対して，前の操作の各時刻 t と，次の操作の各時刻 t'（ただし $t' < t + p_{j,k}$）のペアについて制約を登録する：

Listing A.4　順序制約の登録

```
for current_task, next_task in consecutive_pairs:
    for t in range(max_time + 1):
        for t_prime in range(min(t + duration, max_time + 1)):
            csp.add_constraint(valid_edges, {x[t], x_next[t_prime
                ]})
```

1 つの順序制約には 2 個の変数が関与するため，CSP 内部での列挙は $2^2 = 4$ 通りのみである．一意性制約の 2^T 通りと比較して，個々の制約の処理は軽量である．

図 A.3 に 1 つの時刻ペアに対する登録処理を，図 A.4 に全体のループ構造を示す．

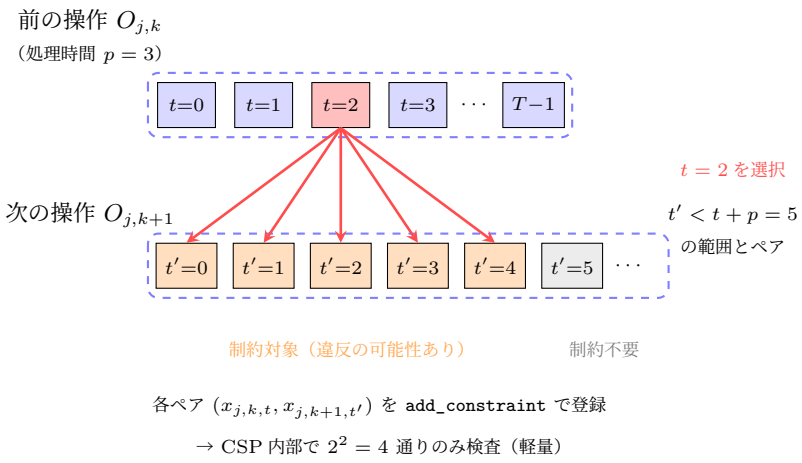


図 A.3　順序制約のペア生成．前の操作 $O_{j,k}$ の時刻 t（例：$t = 2$）に対して，次の操作 $O_{j,k+1}$ の $t' < t + p$ の範囲とペアを作り制約を登録する．

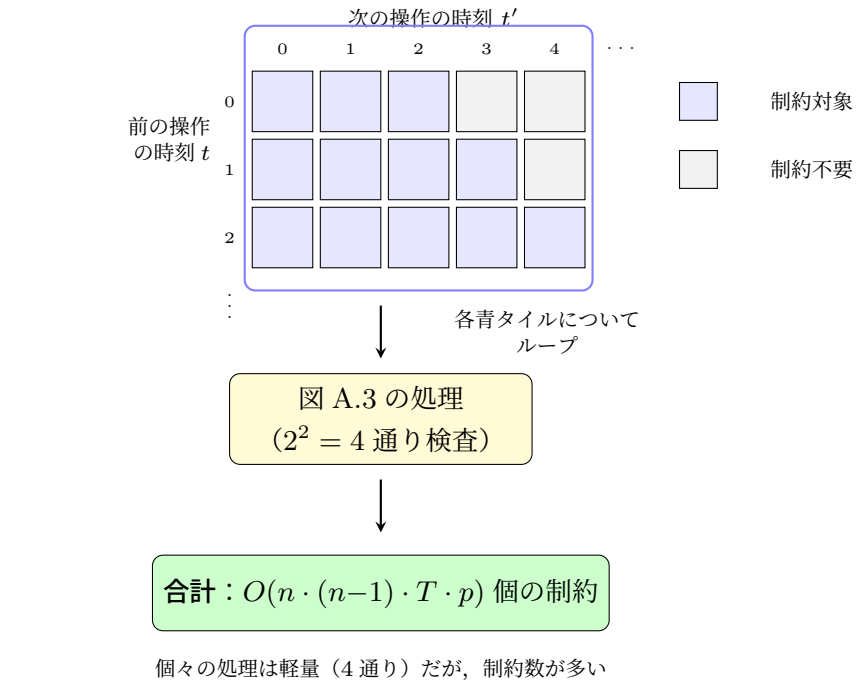


図 A.4　順序制約登録の全体ループ．1 つの操作ペア $(O_{j,k}, O_{j,k+1})$ に対して，青タイルの数だけ `add_constraint` が実行される．

A.2.4　衝突回避制約の登録

第 3.2 節で述べた衝突回避制約は，同一機械を使用する異なる操作について「処理期間が重複しない」ことを要求する：

$$t \leq t' < t + p_{j,k} \text{ のとき } x_{j,k,t} \cdot x_{j',k',t'} = 0 \tag{A.3}$$

順序制約と同様に，Carugno らはこの制約を許可される構成の集合として表現している：

Listing A.5　衝突回避制約の許可構成

```
valid_values = {(0, 0), (1, 0), (0, 1)}  # (1, 1) is forbidden
```

この集合の各要素 $(x_{j,k,t}, x_{j',k',t'})$ は，同一機械上の 2 つの操作の時刻変数の値の組み合わせを表す：

- $(0,0)$：操作 A が時刻 t に開始しない，操作 B も時刻 t' に開始しない → 許可
- $(1,0)$：操作 A が時刻 t に開始する，操作 B は時刻 t' に開始しない → 許可
- $(0,1)$：操作 A が時刻 t に開始しない，操作 B が時刻 t' に開始する → 許可
- $(1,1)$：両方が同時に 1 → **禁止**（処理期間が重複する場合）

同一機械 M_m を使用する操作の集合を $\mathcal{O}_m$ とする．$\mathcal{O}_m$ 内の各操作ペア (O_A, O_B) について，処理期間が重複しうる時刻ペア (t, t') に制約を登録する：

Listing A.6　衝突回避制約の登録

```
for machine in machines:
    for (task_a, task_b) in pairs_on_machine:
        for t in range(max_time + 1):
            for t_prime in overlap_range(t, duration_a):
                csp.add_constraint(valid_values, {x_a[t], x_b[
                    t_prime]})
```

1 つの衝突回避制約には 2 個の変数が関与するため，CSP 内部での列挙は $2^2 = 4$ 通りのみである．

図 A.5 に時刻ペアの構造を，図 A.6 に全体のループ構造を示す．

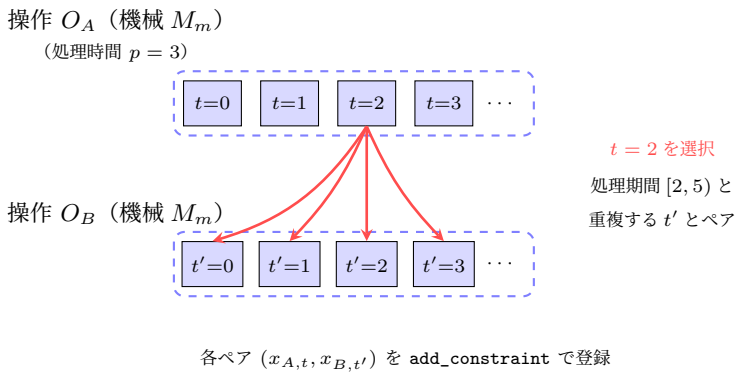


図 A.5　衝突回避制約のペア生成．同一機械上の操作 O_A と O_B について，処理期間が重複しうる時刻ペアに制約を登録する．

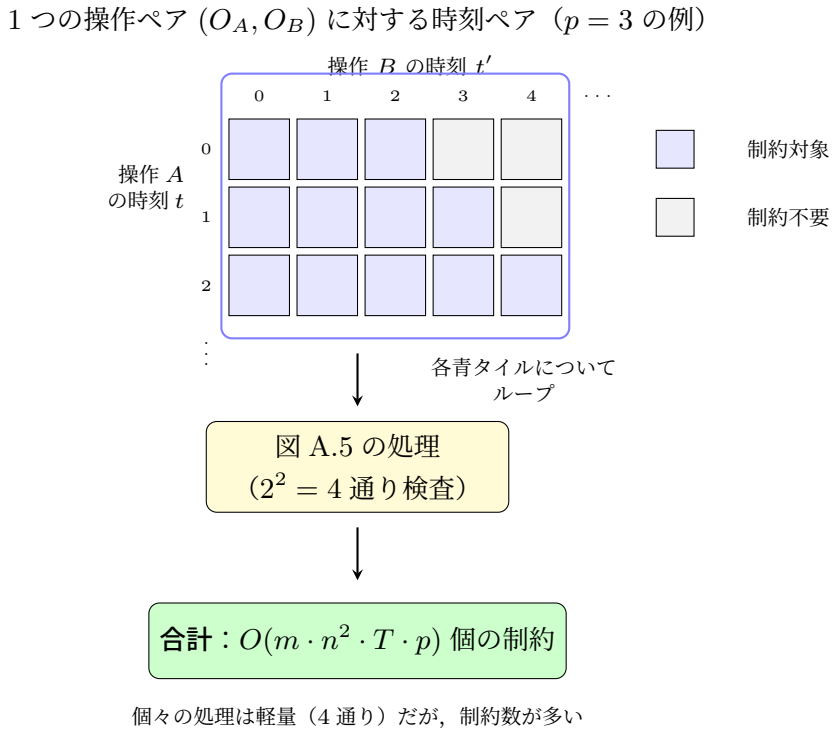


図 A.6　衝突回避制約登録の全体ループ．同一機械上の操作ペア (O_A, O_B) に対して，青タイルの数だけ `add_constraint` が実行される．

A.2.5　不要変数の除去

3 種類の制約を CSP オブジェクトに登録した後，Carugno らの実装では「不要変数の除去」処理を行う．これは，明らかに実行不可能な時刻の変数を CSP オブジェクト内で

0 に固定する処理である．

具体的には，以下の 2 種類の時刻を除去する：

1. **早すぎる時刻**：前の操作が完了する前の時刻．例えば，ジョブ内で 2 番目の操作は，1 番目の操作の処理時間が経過するまで開始できない．
2. **遅すぎる時刻**：後続の操作を実行する時間が残らない時刻．例えば，ジョブ内で最後から 2 番目の操作は，最後の操作の処理時間を残して完了する必要がある．

この処理は，dwavebinarycsp が提供する `fix_variable()` メソッドを用いて実装されている：

Listing A.7　不要変数の除去（早すぎる時刻）

```
predecessor_time = 0
for task in tasks:
    for t in range(predecessor_time):
        label = get_label(task, t)
        csp.fix_variable(label, 0)  # 変数を0に固定
    predecessor_time += task.duration
```

`fix_variable(label, 0)` は，指定した変数を値 0 に固定する．これにより，その変数は以降の処理（stitch 関数による BQM 変換）において定数として扱われる．

この処理は，第 3.2 節で述べた Venturelli らの時間窓計算と類似の効果を持つ．ただし，Venturelli らの手法が「最初から不要な変数を生成しない」のに対し，この処理は「すべての変数を生成した後に固定する」という点で異なる．

図 A.7 に，不要変数の除去の概念を示す．

図 A.7　不要変数の除去．各操作について，早すぎる時刻（前の操作が未完了）と遅すぎる時刻（後続操作の時間が不足）を 0 に固定する．$T = 6$ の例．

A.2.6　変数数の比較

表 A.1 に，各制約に関与する変数数をまとめる．

表 A.1　Step 1 で登録される各制約の変数数

制約	Venturelli の記号	関与する変数数	備考
一意性制約	h_3	T 個	1 操作あたりの全時刻変数
順序制約	h_1	2 個	連続する操作ペアの時刻変数
衝突回避制約	h_2	2 個	同一機械上の操作ペアの時刻変数

重要な観察：一意性制約のみが T 個という多数の変数を含む．$T = 20$ の場合，一意性制約は 20 個の変数を扱う一方，順序制約と衝突回避制約は常に 2 個である．この違いが CSP 内部での列挙回数に大きく影響し，ボトルネックの原因となる．