量子アニーリングによるバッチスケジュール最適化の評価

道後 千尋† 斉藤 和広†

†KDDI 総合研究所 〒356-8502 埼玉県ふじみ野市大原 2 丁目 1 番 15 号

E-mail: †ch-dogo@kddi.com, ku-saitou@kddi-research.jp

あらまし 情報システムには物理上同時に処理できない工程やフロー上前後してはいけない工程がある．バッチスケジューリングとはこれらの制約を守りながら，複数の情報処理タスク(バッチ)の処理順序を決定する組合せ最適化問題である．この問題は特にバッチ数が多くなると解の候補が多くなり，計算量が指数的に増えることが知られている．本研究では，組合せ最適化問題に有効と期待される量子アニーリング技術を用いて，バッチスケジューリング問題を解く．スケジュール作成上の制約と最適化すべき目標を QUBO 式で表し，パラメータを変更して考察を行い，大規模情報システムのバッチスケジューリング問題における量子アニーリングの実用性を評価した．結果として求解可能であったものの，バッチ数が増えるにつれて実行可能解を得られる確率が下がる点が難点となり，大規模システムでは非現実的な実行時間を必要とすることが分かった．

キーワード 先進ハードウェア活用，分散並列処理，チューニング，効率性・スケーラビリティ，探索アルゴリズム

1. 背景

コンピュータとインターネットが普及し，データの扱いが盛んになったことから，多くの業界で情報処理システムが活用されている．情報処理システムにはリアルタイム処理の他に，蓄積した情報をまとめて一括で処理するバッチ処理が存在する．バッチ処理では夜間などのリアルタイム処理が少ない時間帯に，高負荷の処理を行う事が多い．バッチ処理を行うシステムでは，扱う業務が多くなるにつれて処理すべきバッチの数が多くなる．実例としてある大規模なシステムでは 300～1000 の日次または月次バッチが処理されており，概ね夜間の 22:00~1:00 の間に完了させる必要がある．これらのバッチ処理を夜間という期限内に完了させるためには，リソースを最大限に活用したスケジュールを組む必要がある．この問題は，一般にジョブショップスケジューリング問題と呼ばれ，要素数によって組合せ数が指数的に増えるスケジューリング問題であり，NP 困難であることが知られている [1].

近年このような多項式時間での計算が困難な組合せ最適化問題の解決に，量子アニーリング [2]の活用が期待されている．

本論文では量子アニーリング技術を用いてバッチスケジューリング問題を解き，その精度と必要時間について評価・考察を行った．

2. 技術背景

2.1. 組合せ最適化問題とは

目標と誓約と複数の要素を与えられ，制約を守りながら目標をよい値にする要素の組合せを選び出すタスクである．有名な組合せ最適化問題には，ナップザック問題や巡回セールスマン問題がある．本研究で題材にするスケジューリング問題も組合せ最適化問題のひとつである [3].

2.2. 量子アニーリングとは

現在汎用的に使われているコンピュータ(以降古典コンピュータと呼ぶ)は，電圧の高低等で{0/1}のビットを表し，そのビット操作によって情報を処理している．一方で量子コンピュータは，量子のスピン等で{0~1}のビットを表し，情報処理を行う．{0~1}という表現は，量子で表現されたビット(量子ビット/qbit)は量子力学に則って重ね合わせ状態を作ることが可能であり，観測時に{0/1}の値が確定する性質を持つためである．重ね合わせ状態の量子を操作して情報を処理することで，古典コンピュータよりも多くの情報を一度の処理で完了できる特性を持っている．

本研究で利用するのは量子コンピュータ技術の中でも，量子アニーリングと呼ばれる手法である．量子コンピュータの構成は，量子の利用方法によって量子ゲート方式と量子アニーリングとに大別される．量子ゲート方式は古典コンピュータが持つ NAND ゲートや NOT ゲートに相当する回路を量子で実装することで構築される量子コンピュータであり，古典コンピュータと同じように汎用的な用途で使われることが期待されている．一方で量子アニーリングは，古典コンピュータ上でもシミュレート可能な「アニーリング方(焼きなまし法)」というアルゴリズム [4]を，量子ビットを用いて回路実装したコンピュータである．ここでアニーリング法とは，焼いた鉄をゆっくりと冷ますことで内部構造の均一化を図る工業処理のように，場のエネルギーが高い状態から低い状態へ時間をかけて

遷移させることで理想状態を得やすくする探索アルゴリズムである．エネルギーの極小点で探索が止まりにくく，最小値を得られる確率が高いため，組合せ最適化問題のような変数が多い探索問題に使用されている．

2.3. 量子アニーリングの回路設計

ここでは量子のスピンを観測対象とし，ある量子ビットのスピンが上向きであるときは{+1}，下向きであるときは{-1}として扱う．量子アニーリングの回路は量子ビットを並べ，それぞれ隣の量子と縦磁場エネルギーで相互作用を起こすように繋げられた構成を持つ．この回路に対して横磁場を強くかけた状態からゆっくりと横磁場を弱めていくことで，回路のエネルギーが低い量子スピン状態へ収束していく現象が観測可能となる．この観測中で回路の持っているエネルギーをハミルトニアンと呼び，$\mathcal{H}$で表す．最終的に回路に存在する量子ビットは上スピンまたは下スピンの状態になるため，{+1/-1}に読み替えて計算結果とする．つまり量子ビット間の接続や相互作用の強さを解きたい問題に合わせて設計することで，その回路のエネルギーが低くなる上下スピンの組合せが得られる．

古典コンピュータ上のアニーリング法と同じく，量子アニーリングも得られる結果にはランダム性が存在し，また最終的に得られる$\mathcal{H}$の理想状態への近さは横磁場を弱める速度にも影響を受ける．そのためアニーリングを行う際には数回～数千回の試行を行い，もっとも理想状態に近い解($\mathcal{H}$が最小の解)を採用するのが一般的な使い方である．

2.4. 組合せ最適化問題への応用原理

量子アニーリングの物理現象を組合せ最適化問題に応用する為には，組合せ要素 1 つに対して 1 量子ビットを用意し，その要素を採用する{+1}と採用しない{-1}として表現させ，それらの要素を間の関係性を相互作用で表現して回路を実装させる必要がある．

まずは量子アニーリングで起こる現象を式で表す．物理現象を言い換えると，二つの量子ビットσ_i, σ_jの相互作用係数J_{ij}，及び量子ビットσ_iに影響する横磁場エネルギー係数h_iから，ハミルトニアン$\mathcal{H}$を最小化する量子ビットσ_iのスピン方向{+1/-1)を決定するアルゴリズムであるとまとめることができる．この仕組みはイジングモデルと呼ばれる以下の式で表現される．

$$\mathcal{H} = \sum_{i,j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i \qquad (1)$$

組合せ最適化問題の制約と目標をこの式の形で表し，目的関数の最小化問題としてこのイジングモデルに定式化することで，それに対応するよう相互作用を持つ量子回路を構築することができ，量子アニーリングの物理現象によって解くことが可能となる．

3. 問題設定

本研究では，情報システムにおけるN個のバッチを処理する問題を取り上げる．各バッチの工程毎に必要な時間(Required Time)である「読込処理R^{in}」「計算処理R^{pro}」「書込処理R^{out}」は事前に測定し，固定値で与えるものとする．

バッチ処理の制約として，並行処理ができない工程を設ける．任意のバッチが読込処理または書込処理を行っている間は，他のバッチが読込処理または書込処理を行う事はできない．一方で計算処理については，他のバッチの他の工程と並行処理可能とする．

制約を守った上で，バッチスケジューリングで最適化すべき目標値はいくつか考えることが可能である．本論文では本章で後述する以下の 3 つの目標について，量子アニーリングで解く際に適切な目的関数となるかを検討する．

① 総処理時間最小
② オーバーラップ最大
③ フェアスケジューリング

本論文ではNが小規模の問題を解いたのち，大規模システムへの拡張について考察する．

3.1. 目標①総処理時間最小

総処理時間最小では，出来る限り短い総処理時間(makespan)で処理を完了させることを目指す．大規模なシステムのバッチ処理では，あるシステムのバッチ処理によって生成されるデータが，後続のシステムへと渡され，さらに別のバッチ処理に使用される構成を取ることがある．その為バッチ処理にかかる総処理時間は，小さいほど良いスケジュールといえる．

3.2. 目標②オーバーラップ最大

オーバーラップ最大では，出来る限り処理を並列で行うことを目指す．単位時刻にバッチ処理が並列で処理されている工程の数をオーバーラップ数と定義する．本問題設定ではリソースを最大限使うことを考え，大きいほど良いスケジュールとする．問題設定によっては平均的にリソースを使用することが求められたり，最大値を制約にしたりする場合もありうる．

3.3. 目標③フェアスケジューリング

フェアスケジューリングでは，ある時刻において出来る限り多くのジョブが終わっている状態を作ることを目指す．バッチ処理は通常，処理完了した情報を待っている後続業務が存在している．目標①のように後続システムに後続業務がある場合も，自システムの中で後続業務がある場合もありうる．スケジュールを作成している一連のバッチ処理と並列して行うことが可能な後続業務がある場合，総処理時間が等しいスケジュールであっても，その途中のある時刻において完了状態のバッチが多いスケジュールほど，後続業務に

早く取り掛かることが可能な為，良いスケジュールといえる．

4. 量子アニーリングのための定式化

量子アニーリングで問題を解くにあたって，最初に決定変数を定義する．スケジューリング問題では組み合わせる要素の定義法方はいくつか考えられるが，本研究では各タスク(バッチ)を開始する時刻の組合せを要素とする．他にも各タスクを開始する順序を要素とする設計なども可能である．具体的には決定変数$x_{nt} \in \{0,1\}$をバッチ数N×時刻Tの行列で用意し，バッチnの処理が開始した時刻tを 1 として定義する．

例として各バッチ処理工程に必要な時間が表 1 のようなシステムを考える．この時バッチ A～E を時間制限 25 コマの中で A→B→C→D→E の順に詰めて処理するスケジュールは 5×25 行列で表現ができ，図 1 のようになる．図 1 のスケジュールは 3 章で設定した制約を満たすことから実行可能解といえる．ただし目的が良い数値かは未検討状態である為，大域最適解であるかは吟味の余地が残されている．このようにしてスケジュールを立てるというタスクは，この決定変数が制約を満たした上で，目的関数がよりよい値になるよう探索することに置き換わる．

本問題設定は決定変数が 125 個ある為，この量子回路を作成する為には 125qbit 必要になる．

4.1. QUBO 式の利用

2 章で述べたように,量子アニーリングはハミルトニアン$\mathcal{H}$を最小化するスピン方向を決定するアルゴリズムである．そのハミルトニアン$\mathcal{H}$のエネルギー最小問題へと読み替えるにあたって，スピン方向は{+1/-1}であるが,本問題で設定した決定変数は{0/1}であるため,数式(1)のイジングモデルにおいて以下のように量子ビットσ_iをバイナリ変数$x_i \in \{0,1\}$に変換する．

$$x_i = \frac{\sigma_i + 1}{2} \tag{2}$$

量子アニーリング利用時は，これを数式(1)のイジングモデルに適用して以下のような QUBO (Quadratic Unconstrained Binary Optimization) 問題とする．

$$\mathcal{H} = \sum_{i,j} Q_{ij} x_i x_j \tag{3}$$

QUBO 式はイジングモデルと 1 対 1 で対応する為，本課題の目的と制約を QUBO 式で表すことで，量子アニーリングを使って求解可能となる．

4.2. 総処理時間最小(目的関数)$\mathcal{H}_{time}$

総処理時間をN番目のバッチが終了した時刻と定義すると，式の通りに表せる．

表 1　各バッチ工程毎に必要な時間

バッチ n	読込処理R_{in}	計算処理R_{pro}	書込処理R_{out}
A	2	4	2
B	2	6	2
C	1	10	1
D	1	4	1
E	3	4	2

図 1 スケジュール一例

$$\mathcal{H}_{time} = \max\left(R_n^{all} + \sum_{t=0}^{T} t * \mathrm{x}_{nt} \mid n \in \{A,B,C,D,E\}\right) \tag{4}$$

ただし

$$R_n^{all} = R_n^{in} + R_n^{pro} + R_n^{out} \tag{5}$$

QUBO 式は量子回路で実装されるという特性上，計算結果によって分岐が必要になる max 関数のような処理は構築不可能である．そのため式を下記の通りに変形する．

$$\mathcal{H}_{time} = \sum_{t=0}^{T} \prod_{n\in\{A,B,C,D,E\}} \sum_{t'=0}^{t-R_n^{all}} \mathrm{x}_{nt'} \tag{6}$$

数式(6)は全てのバッチが完了した時刻tをカウントしており，数値が大きいほど良いスケジュールである．

4.3. オーバーラップ最大(目的関数) $\mathcal{H}_{over}$

オーバーラップ数の QUBO 式は以下の通りになる．

$$\mathcal{H}_{ovre} = \sum_{t=0}^{T} \sum_{n=0}^{N-1} \sum_{m=n+1}^{N} \sum_{t'=0}^{R_n^{all}} \sum_{t''=0}^{R_m^{all}} x_{n(t+t'-R_n^{all})} * x_{m(t+t'-R_m^{all})} \tag{7}$$

本問題設定においては,数値が大きいほどよいスケジュールである．

4.4. フェアスケジューリング (目的関数) $\mathcal{H}_{fair}$

本問題設定では任意のバッチの処理が中断することは無いため，開始時間の和を QUBO 式で表す．

$$\mathcal{H}_{fair} = \sum_{t=0}^{T} \sum_{n\in\{A,B,C,D,E\}} t * x_{nt} \tag{8}$$

数値が小さいほどよいスケジュールである．

4.5. バッチ起動回数の制約$\mathcal{H}_{row}$

ここまで問題設定として明記はしていなかったが，

各バッチは 1 回のみ処理すればよい．この制約を QUBO 式で表すと以下の通りになる．

$$\mathcal{H}_{row} = \sum_{n\in\{A,B,C,D,E\}} \left(-1 + \sum_{t=0}^{T} x_{nt}\right)^2 \quad (9)$$

4.6. 並列不可処理の制約$\mathcal{H}_{in}$ $\mathcal{H}_{inout}$ $\mathcal{H}_{out}$

問題設定で制約とした並列処理を表す QUBO 式は以下の 3 式になる．

(1)入力処理と入力処理が重ならない制約

$$\mathcal{H}_{in} = \sum_{t=0}^{T} \sum_{n\in\{A,B,C,D,E\}} \sum_{t'=0}^{R_n^{in}} \sum_{\substack{n\in\{A,B,C,D,E\} \\ |n\neq m}} x_{nt} * x_{m(t+t')} \quad (10)$$

(2)出力処理と出力処理が重ならない制約

$$\mathcal{H}_{out} = \sum_{t=0}^{T} \sum_{n\in\{A,B,C,D,E\}} \sum_{t'=0}^{R_m^{out}} \sum_{\substack{n\in\{A,B,C,D,E\} \\ |n\neq m}} x_{n(t-R_n^{all}-R_n^{pro})} * x_{m(t+t'-R_m^{all}-R_m^{pro})} \quad \cdots(11)$$

(3)入力処理と出力処理が重ならない制約

$$\mathcal{H}_{inout} = \sum_{t=0}^{T} \sum_{n\in\{A,B,C,D,E\}} \sum_{t'=0}^{R_n^{in}+R_m^{out}} \sum_{\substack{n\in\{A,B,C,D,E\} \\ |n\neq m}} x_{nt} * x_{m(t+t'+R_m^{all})} \quad (12)$$

4.7. バッチ完了の制約$\mathcal{H}_{fin}$

各バッチの出力が重ならない制約を有効にするため，総処理時間が最大時刻Tを越えてはいけない．この制約を QUBO 式で表すと下記の通りになる．

$$\mathcal{H}_{fin} = \sum_{t=0}^{T} \sum_{n\in\{A,B,C,D,E\}} \left|t * x_{nt} + R_n^{all} + T\right| \quad (13)$$

4.8. 定式化

4.1～4.6 の式(6)～(13)を$\mathcal{H}_1$～$\mathcal{H}_8$と置き，本研究の最適化問題を以下のように定式化する．ただしω_1～ω_8は各$\mathcal{H}$の式に対応する重みパラメータである.量子アニーリングはハミルトニアン$\mathcal{H}$が最小になるように収束する為ω_1及びω_2は負，それ以外は正の数とする．

$$\mathcal{H} = \sum_{n=1}^{8} \omega_n * \mathcal{H}_n \quad (14)$$

制約のハミルトニアン$\mathcal{H}$($\mathcal{H}_{row}$，$\mathcal{H}_{in}$，$\mathcal{H}_{inout}$，$\mathcal{H}_{out}$，$\mathcal{H}_{fin}$)が 0 となるものが，実行可能解となる．そして本問題のゴールは，実行可能解の中でも(14)式の値を最小にする決定変数x_{nt}の組合せを得ることである．

表 2 コンパイル時間

QUBO 式	コンパイル時間(秒)
$\mathcal{H}_{time}$	90698.890
$\mathcal{H}_{over}$	0.183
$\mathcal{H}_{fair}$	0.003
$\mathcal{H}_{row}$	0.005
$\mathcal{H}_{in}$	0.063
$\mathcal{H}_{out}$	0.046
$\mathcal{H}_{inout}$	0.067
$\mathcal{H}_{fin}$	0.001

5. 量子アニーリングでの評価

4 章の QUBO 式について，量子アニーリングのシミュレータを古典コンピュータ上で実行し，最適化問題を解いた．今回は小規模な問題設定として，バッチ数 N=5，最大時刻T=25 とし，3 章の表 1 の通りに各バッチの処理時間を定義した．

5.1. 環境

クラウドサービス AWS を利用した Amazon SageMaker Notebook Instance 上に以下の環境を作成し，実行した．

インスタンス：ml.m5.xlarge
モデル：Intel(R) Xeon(R) Platinum 8259CL CPU
vCPU：4
メモリ：16GiB
GPU：使用なし
Python：3.9.13
pyQUBO：1.2.0
OpenJij：0.6.9

量子アニーリングのシミュレータとして OpenJij [5] の Simulated Quantum Annealing (SQA) Sampler を利用した．この手法は以降 SQA 法と呼ぶ．設定は特に断りが無い場合 beta=10, gamma=1.0, trotter=10, num_reads=100, sweeps=1000 とした．この設定のうち beta, gamm, trott，は量子の設定に関する値であり，num_reads は 1 回の実行で得られる解の数(＝アニーリングの試行回数)， sweeps は磁場の変化の速度を決めている．

5.2. 事前検証

評価の前に，実行環境で立式した QUBO 式が解けることを確認する．求解では，式を古典コンピュータ上で疑似的に計算可能な量子回路にコンパイルし，その後に実行して解を得るという手順を踏む．数式で記載できても物理的に接続しようのない式は，コンパイル不能となる．

5.2.1. コンパイル結果

各 QUBO 式はすべてコンパイル可能であり，処理時間は表 2 の通りとなった．$\mathcal{H}_{time}$のコンパイル時間が桁違いに長いことが分かる．理由として式(6)が総

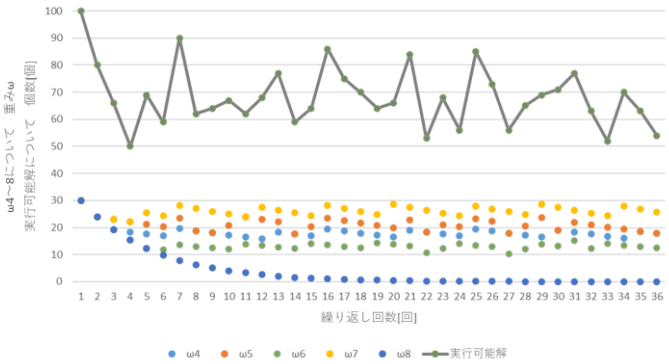

図 2　チューニング結果

乗を含んでいる点が挙げられる．現在はバッチ数 5 としているため現実時間内でコンパイルを完了できるが，今後の拡張性を考えると現実的でなく，また解く時間も同様に長くなることが予想される．よって以降の検討は$\mathcal{H}_{time}$を除いた以下のハミルトニアン$\mathcal{H}_{II}$について，エネギーを最小化する問題として考える．

$$\mathcal{H}_{II} = \sum_{n=2}^{8} \omega_n * \mathcal{H}_n \tag{15}$$

5.2.2. 求解結果

コンパイルした後に SQA Sampler を用いてアニーリングを実施し，スケジュールを得た．$\mathcal{H}_{II}$のコンパイル時間は 0.386 秒であり，求解時間は 5.089 秒であった．また得られた 100 の解のうち，98 が実行可能解であり，QUBO 式が解けることが確かめられた．

5.3. パラメータチューニング

制約の各 QUBO 式に掛けている重みωのチューニングを行った．チューニングでは，違反することの多い制約の重みを大きくすることで，アニーリングに際して局所解に陥りにくくし，実行可能解を得られる確率を高める．

5.3.1. チューニング方法

1. 初期値として各制約に割り当てた重み$\omega_{4\sim8}$を，30 とする．目的関数の重み$\omega_{2,3}$は 1 で固定する．
2. 解を 100 回求め，各制約の違反回数を集計する．
3. 違反が 50 以下の場合$\omega_{4\sim8}$を 0.8 倍にし，2 に戻る．そうでない場合，4 へ進む．
4. 違反が平均より多かった QUBO 式について重みを 1.2 倍に更新し，2 に戻る．

上記の作業を，100 回行う．

5.3.2. チューニング結果

結果は図 2 の通りになった．違反が 50 以下の重みについて，構成比を求めた後に平均化し，近似することで，制約のパラメータを以下の比率となった．以降

図 3 大域最適解のスケジュール

の$\mathcal{H}_{II}$ではこの重みパラメータを使用する．

$$\omega_4 : \omega_5 : \omega_6 : \omega_7 : \omega_8 = 5 : 6 : 4 : 7 : 1$$

5.4. 実行可能解を得られる時間(TTS)の考察

アニーリング法はランダム性がある為，得られた解が実行可能とは限らない．99%の確率で実行可能解を得る為にかかる時間を TTS と呼び，次式の通りに表される．

$$\left\lceil \frac{\log(1-0.99)}{\log(1-r_1)} \right\rceil * t \tag{16}$$

ただし 1 回のアニーリングで解が得られる確率をr_1，1 回のアニーリングにかかる時間を t とする．

チューニング後の$\mathcal{H}_{II}$について計測した結果，$r_1 = 0.98$,t=0.051 であり，式(16)を解くと 1.18 である．よって必要なアニーリング回数(Num read 数)は 2，必要実行時間は 0.102 秒となる．このことから本問題設定では，実用的時間内にスケジュールを決定可能といえる．

5.5. 古典手法による大域最適解取得との比較

本問題設定は NP 困難な問題であるが，N＝5 という小規模な問題設定においては全ての作成しうるスケジュールに対して制約違反を調べることで，大域最適解を取得可能である．この手法の汎用解法を考えると，決定変数が 125 個あることから2^{125}通りの解を調べることになる．一方で専用解法として，バッチ起動回数の制約$\mathcal{H}_{row}$を加味することで$({}_{25}C_1)^5$通りに抑えることができる．本節ではこの専用解法と比較する

5.5.1. 精度の考察

SQA 法と同じ実行環境でこの古典的専用解法で解いたところ，図 3 の大域最適解を取得できた．この解はチューニング後に SQA 法を 1 回実行して得た$\mathcal{H}_{II}$の解に含まれており，SQA 法においても大域最適解を得られることが確かめられた．よって精度は古典手法と差が無いといえる．

5.5.2. 時間の考察

古典的専用解法は解が得られるまで 4.5H を要した．一方で SQA 法では 1 回の実行 5.071 秒で得られる 100 の解のうち，8 が大域最適解であった．TTS と同様に，99%の確率で大域最適解が得られるまでの時間を計算すると，2.840 秒となる．これより SQA 法の時間

表 3 SQA と SA の比較

	solve(秒)	実行可能解	大域最適解	TTS(秒)
SQA	5.071	98	8	2.840
SA	0.606	86	2	1.382

的優位が確かめられた．ただしアニーリングで解いた場合には，得られた解の中で最もハミルトニアン$\mathcal{H}$が小さくなる実行可能解を知ることは可能だが，その解が大域最適解であるかの判断は不能であることに注意する．

5.6. シミュレーテッドアニーリング(SA)法との比較

本実験では Simulated Quantum Annealing (SQA) を用いたが，これを古典手法である Simulated Annealing (SA) [6]と性能比較する．

同環境で SA Sampler を実行した結果が表 3 である．古典コンピュータに最適化された手法である SA 法が 1 桁短い実行時間で解が得られる結果となった．古典コンピュータ上で実行しているため，量子の状態を逐次更新しながらシミュレートする必要のある SQA 法は必要な計算処理量が多かったと考えられる．一方で解の質に関しては，SQA 法で大域最適解の比率が大きい傾向がみられた．その為，大域最適解が 99%の確率で得られるまでの時間で比較を行うと，時間差は 2 倍程度にまで近づく．量子コンピュータで実行した場合には，SQA 法の実行時間が改善される可能性があり，実機実用に期待ができる．

5.7. 大規模システムへの応用の考察

バッチ数を増やしていった場合に，実行可能解を得る為に必要な時間がどのように変化するかを調べ，その傾向から大規模システムへの適用時の必要時間を予測した．

5.7.1. 大規模システムの問題設定

バッチ数 N を増やした場合，制約を破らないようにスケジュールの時間枠 T も大きくする必要がある．本考察では以下の通りに問題を設定した．

読込処理R_{in}　：1～3 の整数でランダム
計算処理R_{pro}　：1～10 の整数でランダム
書込処理R_{out}　：1～3 の整数でランダム
時間T　：$1.5 * \sum(R_{in} + R_{out})$

5.7.2. 実行結果

$\mathcal{H}_{II}$についてバッチ数Nを 5～10 まで増やした時のコンパイル時間及び求解時間は図 4 の通りになり，N=10 でカーネルが落ちコンパイル不能という結果になった．この時のバッチ数の増加と TTS の関係は図 5 のグラフの通りとなった．

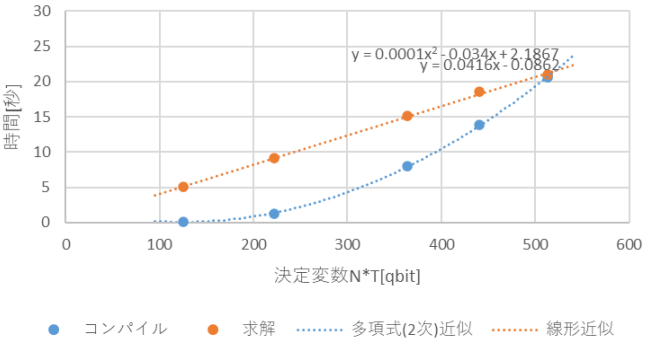

図 4　バッチ数の増加と処理時間の変化

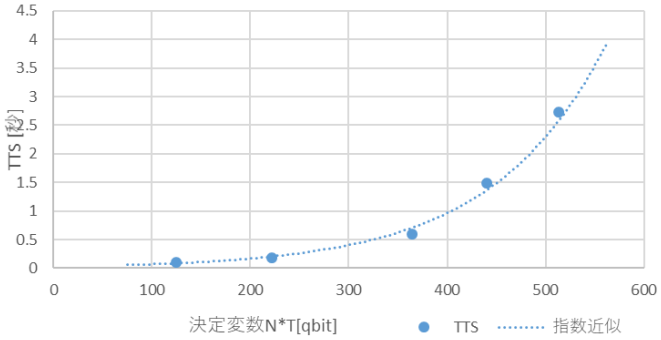

図 5　バッチ数の増加と TTS の変化

5.7.3. コンパイル及び求解時間に関する考察

十分な計算能力を有する環境が用意できるものとして，500 バッチ処理する大規模システムに応用した場合の処理時間を予測する．500 バッチを問題設定に従って作成したところT=616 となり，決定変数は 308,000qbit であった．

図 4 より決定変数の数に対するコンパイル時間の増加は，多項式に近いことが読み取れる．バッチ数及び総時間が増えると入れ子になっている総和の処理が多重で影響するためと考えられる．近似式は y = 0.0001x² - 0.034x となったため，代入すると 9475928 秒≒109 日であった．一方で図 4 より決定変数の数に対する求解時間の増加は，比例に近いことが読み取れる．近似式は y = 0.0416x となったため代入すると 12812.8≒3.5 時間であった．

結論として，システムのスケジュールを決めるタスクとして，非現実的な時間がかかることが示された．実務として活用可能にするには，まず二乗で影響のあるコンパイル時間を削減する必要がある．4.6.3 よりコンパイル時間は QUBO 式に大きく依存することが判明しているため，よりコンパイル時間が短くなるような立式や，補助変数を使用した総和処理の多重度の軽減が改善方法のひとつと考えられる．

5.7.4. TTS に関する考察

本節も十分な計算能力を有する環境が用意できるものとして考察する．図 5 より決定変数の数に対する

表 4　各 QUBO 式のコンパイル時間と可否

N	T	N＊T	$\mathcal{H}_{over}$(秒)	$\mathcal{H}_{fair}$(秒)	$\mathcal{H}_{row}$(秒)	$\mathcal{H}_{in}$(秒)	$\mathcal{H}_{out}$(秒)	$\mathcal{H}_{inout}$(秒)	$\mathcal{H}_{fin}$(秒)
5	25	125	0.183	0.004	0.005	0.063	0.046	0.067	0.001
6	37	222	0.757	0.010	0.012	0.386	0.249	0.493	0.003
7	52	364	2.292	0.021	0.028	1.858	1.716	3.954	0.004
8	55	440	4.257	0.029	0.037	2.874	2.896	6.418	0.005
9	57	513	5.402	0.036	0.047	4.836	3.639	9.876	0.005
10	60	600	9.600	0.045	0.061	7.809	5.077	不可	0.006
12	70	840	35.407	0.079	0.102	不可	不可		0.006
14	81	1134	109.435	0.135	0.165				0.009
16	102	1632	437.392	0.255	0.329				0.012
18	106	1908	不可	0.371	0.440				0.015
20	118	2360		0.536	0.738				0.017

TTS の増加は，指数的であることが読み取れる．TTS の増加は実行可能解が得られる確率が低くなっていることに起因している．Sweep=1000 に固定して実行しているため，決定変数の増加に対して探索の深さが追い付かなくなり，実行可能解が得られにくくなっていると考えられる.近似式は y = 0.0299e0.0087x であり，500 バッチの場合には非現実的な時間となる．
この点に関しては Sweep 数のチューニングによる改善が必須となる．

5.7.5. コンパイル可否に関する考察

コンパイルが不能になった点について，考察する．

考察の為に$\mathcal{H}_{II}$を構成する各 QUBO 式を個別でコンパイルしたところ表 4 の通りになった．「不可」が実行時にカーネルの落ちた箇所である．特に総和操作の多い QUBO 式のコンパイルで，マシンスペックが足りなくなっていることが読み取れる．一方で総和の多重度が最多の$\mathcal{H}_{over}$のコンパイル可能範囲から，総和の多重度のみに可否が依存しているとは言えない．

マシンスペックのうちメモリサイズに問題がある可能性を考え，コンテナ仮想環境を提供するオープンソースソフトウェア Kubernetes [7]上でメモリ 256GB の環境を用意し$\mathcal{H}_{II}$をコンパイルを行った．しかし結果は同様にN=10 でコンパイルが不可能になったため，原因の究明には至らなかった．

6. おわりに

本実験によって，大規模情報システムのバッチスケジューリング問題に対する量子アニーリングの実用性が確かめられた．

今後，本研究で定式化した QUBO 式を元に量子コンピュータ上で実行し，シミュレートとの実行速度の差や，ノイズによる結果の精度変化を考察する．

今後の研究では総処理時間最小化の目的関数や大規模システムへ応用した場合のコンパイル時間や TTS などの，現実時間内で解けなくなった箇所について，補助変数の利用や時刻Tのスケーリングによって改善を試みる．一方でユースケース上必要となる解の精度を評価し，必ずしも大域最適解でなくても，実用に足る実行可能解を得えられるスキームを構築し，実用化に向けた研究を行う．

7. 謝辞

DEIM2023 の期間中に質疑やコメントをくださった方々に感謝します．今後の研究に役立てさせていただきます．

参　考　文　献

[1] 田中俊二，“組み合わせ最適化問題とスケジューリング,” *J-STAGE,* pp. Vol.64,No.6,pp.200-206, 2020.

[2] T. K. a. H. Nishimori, “‘‘Quantum annealing in the transverse Ising model”,” *Phys. Rev. E - Stat.Physics, Plasmas, Fluids, Relat. Interdiscip Top.,* pp. vol. 58, no. 5, pp. 5355~5363, 1998.

[3] 茨木俊秀，“組合せ最適化とスケジューリング問題: 新解法とその動向,” *計測と制御,,* pp. 34(5), 340-346., 1995.

[4] 中. 良平，“シミュレーテッドアニーリング：基礎と最新技術,” *人工知能学会誌,* 第 巻 9, 第 3, pp. 365-372, 1994.

[5] OpenJij. [オンライン]. Available: https://github.com/OpenJij/OpenJij.

[6] I. Kacem, “Multiobjective Simulated Annealing: Principles and Algorithm Variants,” Hindawi, 2019.

[7] B. e. a. Burns, “Borg, omega, and Kubernetes,” Comm. ACM, 2016, pp. Vol.59, No.5, pp.50-57.