一次元アーキテクチャにおける量子ビット割り当て問題

東京大学大学院　情報理工学系研究科　修士1年

内藤 壮俊

Twitter: @hamburg_soshun

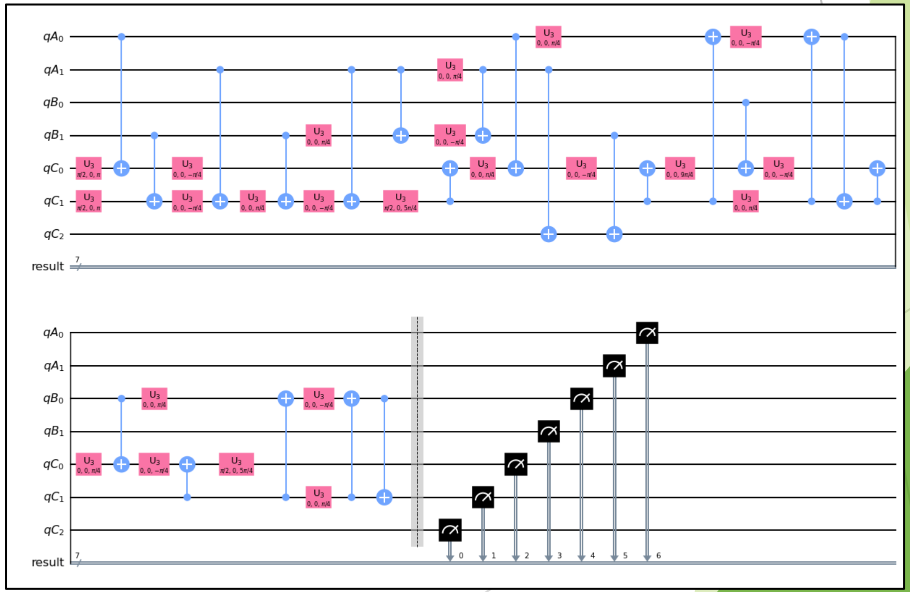
問題設定

量子ゲート型計算機

- 量子ビットがゲートを通過 → 状態変化により計算を行う

- 量子回路はU3ゲートとCXゲートに展開可能
  - U3ゲート：1ビットの状態を任意に操作
  - CXゲート：2ビット間でXOR演算を行う

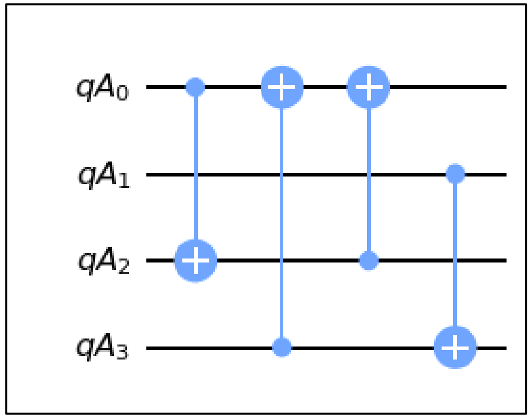
量子回路の一例



量子ビット割り当て問題

- 設計図上の「論理ビット」とデバイス上の「物理ビット」を対応させる問題
- CXゲートを作用させる物理ビットは**隣り合っている必要がある**
- SWAPゲートを使えば論理ビットの交換が可能．しかし，**エラー率が高いためあまり使いたくない**

＜設計図＞

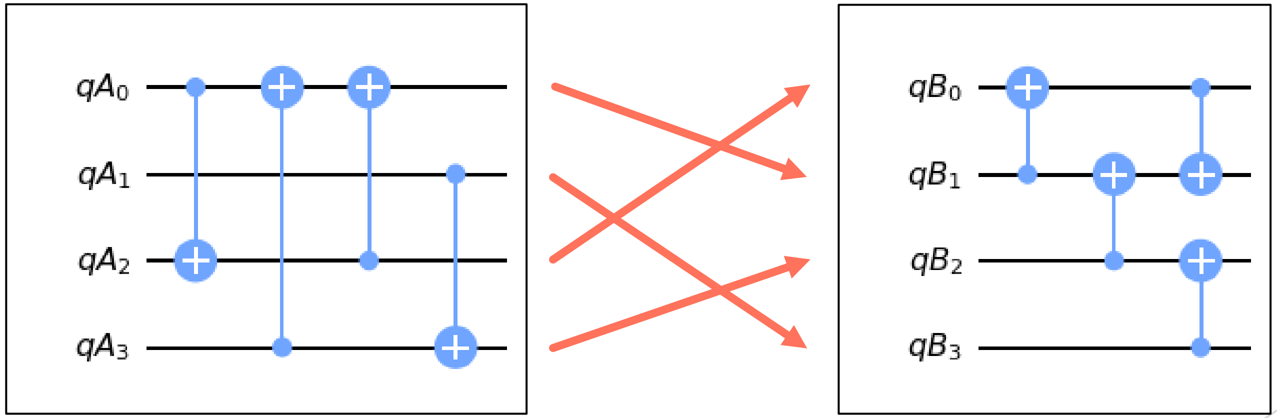


＜デバイス＞

量子ビット割り当て問題

- 設計図上の「論理ビット」とデバイス上の「物理ビット」を対応させる問題
- CXゲートを作用させる物理ビットは**隣り合っている必要がある**
- SWAPゲートを使えば論理ビットの交換が可能．しかし，**エラー率が高いためあまり使いたくない**

＜設計図＞ ＜デバイス＞



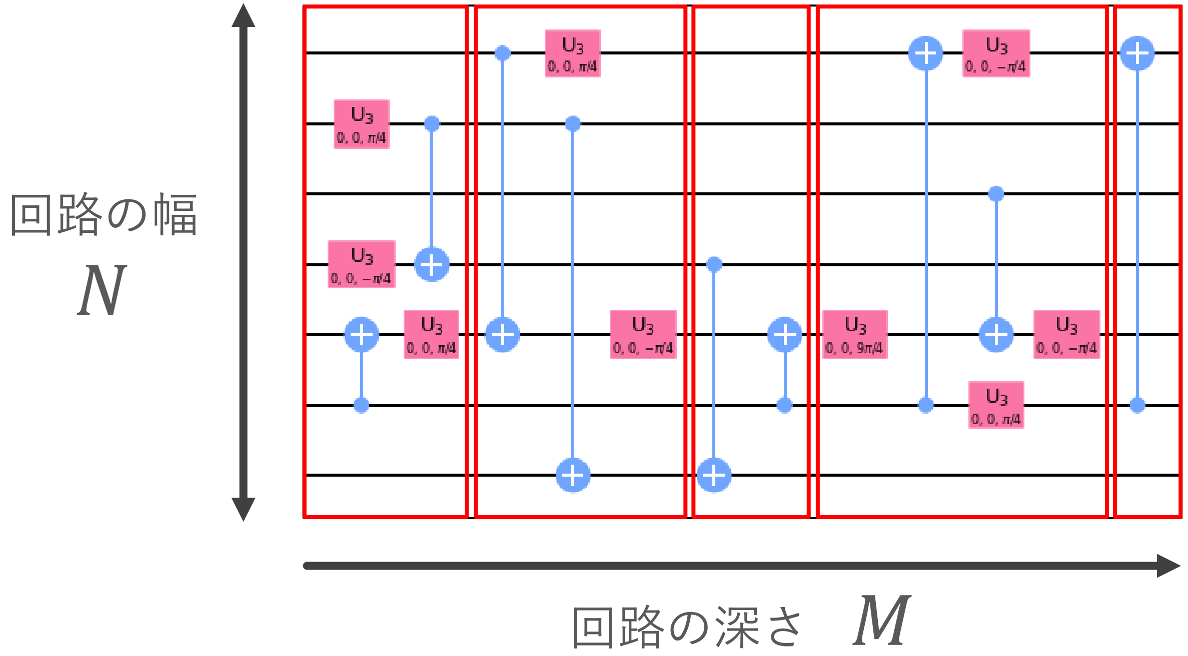
先行研究

- “A QUBO formulation for qubit allocation" (2020) [1]
  Bryan Dury and Olivia Di Matteo
  - エラー率と回路の深さを考慮して，QUBOの係数を設定
  - 論理ビットの初期配置を最適化
  - **ダイナミックな並び替えは考慮せず**

[1] https://arxiv.org/abs/2009.00140

今回扱いたい問題

- 初期配置だけでなく，**並び替え全体を最適化する**問題をAmplifyで解く
- 古典的解法では，$\boldsymbol{O\left(M \cdot (N!)^2\right)}$ **の計算量**



実装・評価

バイナリ変数を用いた定式化

- 論理ビットの並び替えを，MN^2 個のバイナリ変数で表現
- one-hot 制約
  - 物理ビットに関する one-hot 制約
  - 論理ビットに関する one-hot 制約
- CXゲートによる制約
  - 作用させる物理ビットは隣り合っていなければならない

バイナリ変数を用いた定式化

- コスト = SWAPゲートの個数
- 厳密には，コストは **4次式** で表される
  - 一応，補助変数を追加すれば定式化が可能
  - しかし，探索空間が広すぎたせいか上手くいかなかった…

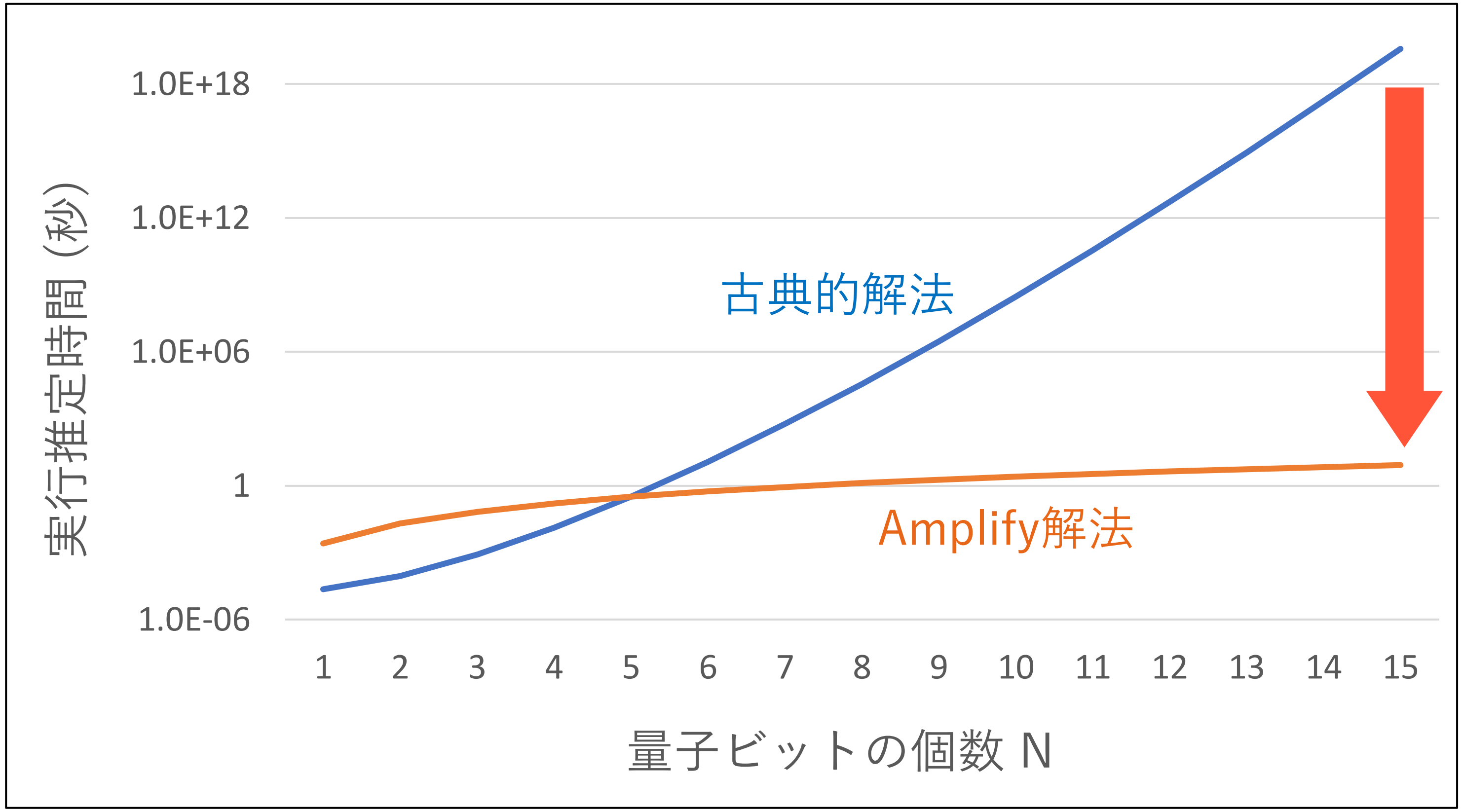
バイナリ変数を用いた定式化

- コスト = SWAPゲートの個数
- 厳密には，コストは **4次式** で表される
  - 一応，補助変数を追加すれば定式化が可能
  - しかし，探索空間が広すぎたせいか上手くいかなかった...

↓

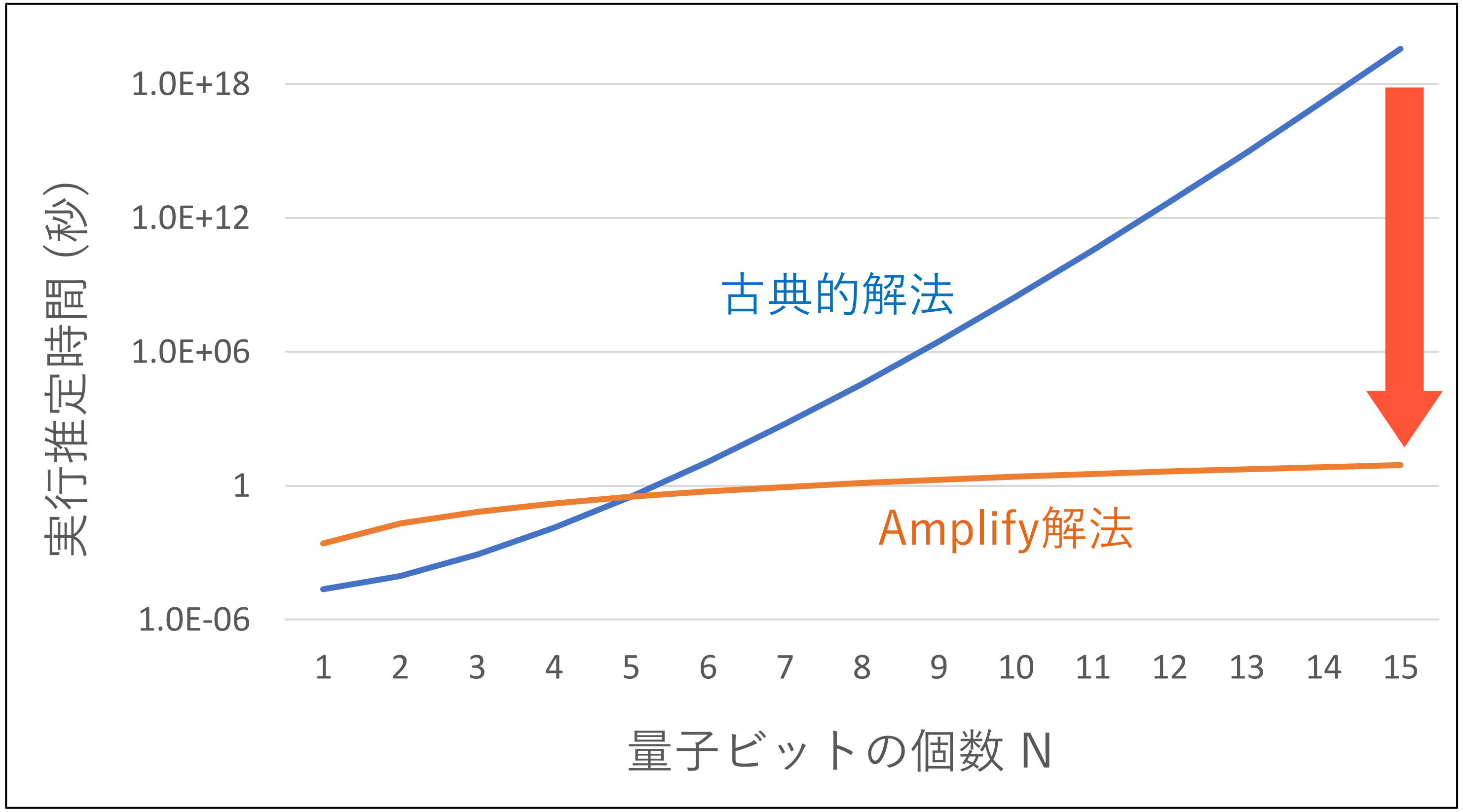
- **コストを 2次式 で近似する**作戦に
  - 補助変数は不要になり，モデルの規模は $O(MN^3)$ に

実行時間の比較



実行推定時間（秒）
1.0E+18
1.0E+12
1.0E+06
1
1.0E-06
古典的解法
Amplify解法
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
量子ビットの個数 N

実行時間の比較



100京倍
の高速化

コストの比較

古典的解法（低速・最適解）　vs　Amplify解法（高速・近似解）

- 小規模な回路 → **最適解にほとんど一致**
- 中規模な回路 → パラメータ調整で対処可能

それぞれの解法におけるコストの平均値

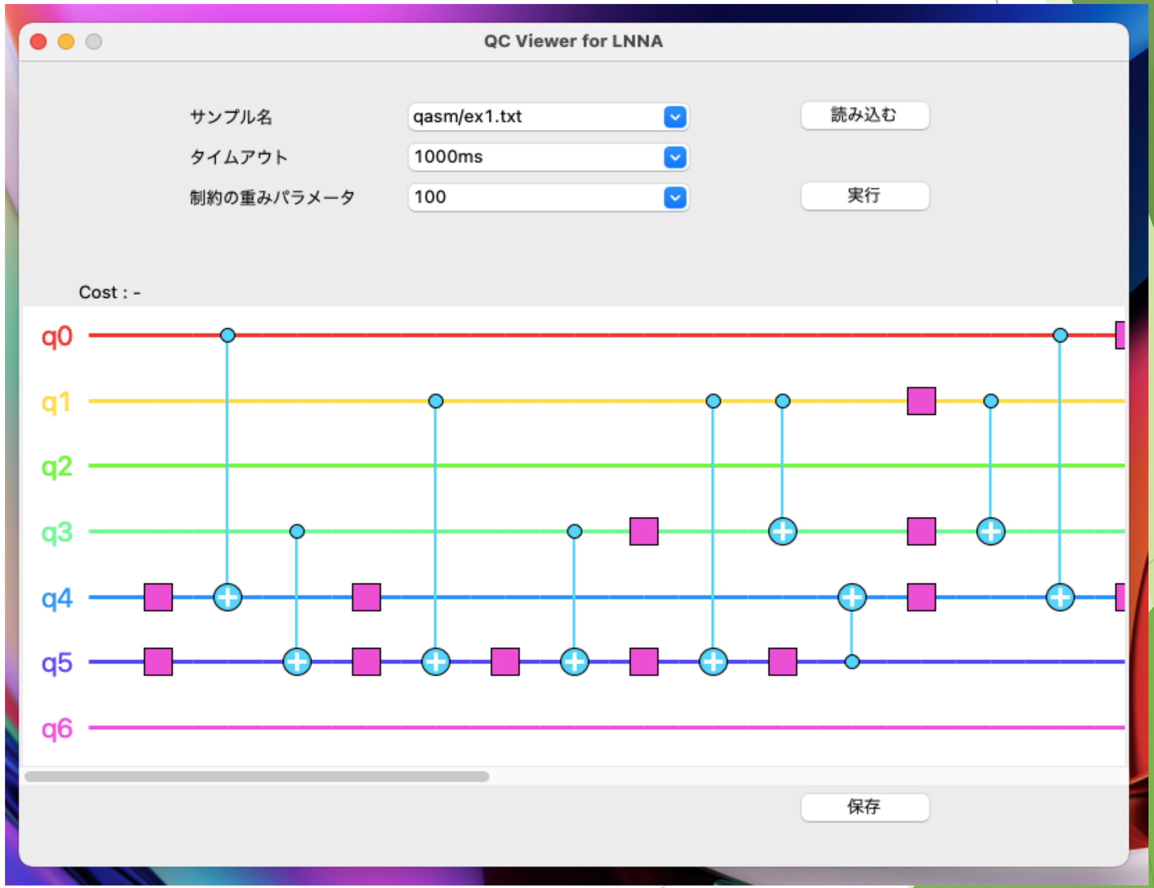
N	3	4	5	6
古典的解法（M = 5）	0.6	1.4	2.1	4.1
Amplify解法（M = 5）	0.6	1.4	2.1	4.5
古典的解法（M = 20）	3.8	13.0	15.0	24.9
Amplify解法（M = 20）	3.8	13.1	18.9	34.0

アプリへの応用

量子回路の描画機能

- GUIアプリを実装
- 量子回路をリアルタイムで描画

量子回路が描画されたアプリ画面の例

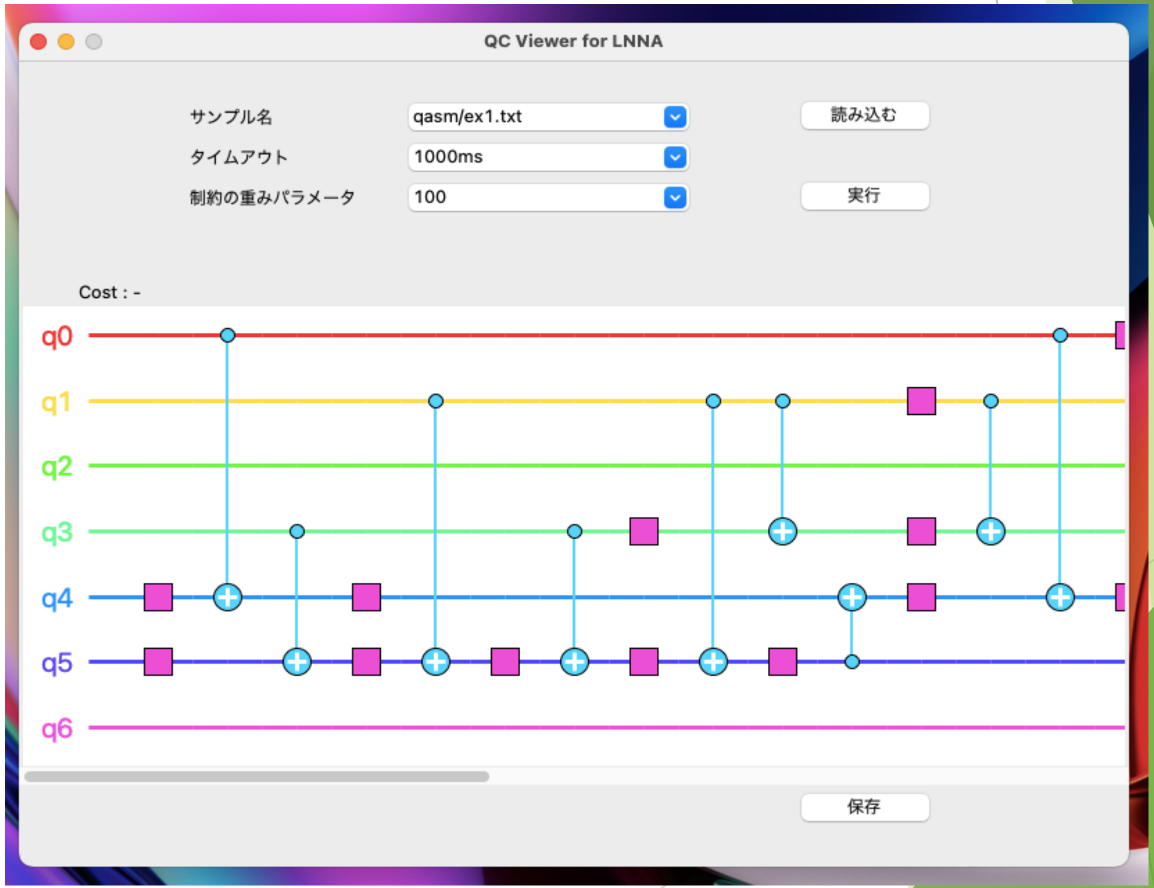


量子回路の描画機能

- GUIアプリを実装
- 量子回路をリアルタイムで描画

作業の様子が一目でわかる！

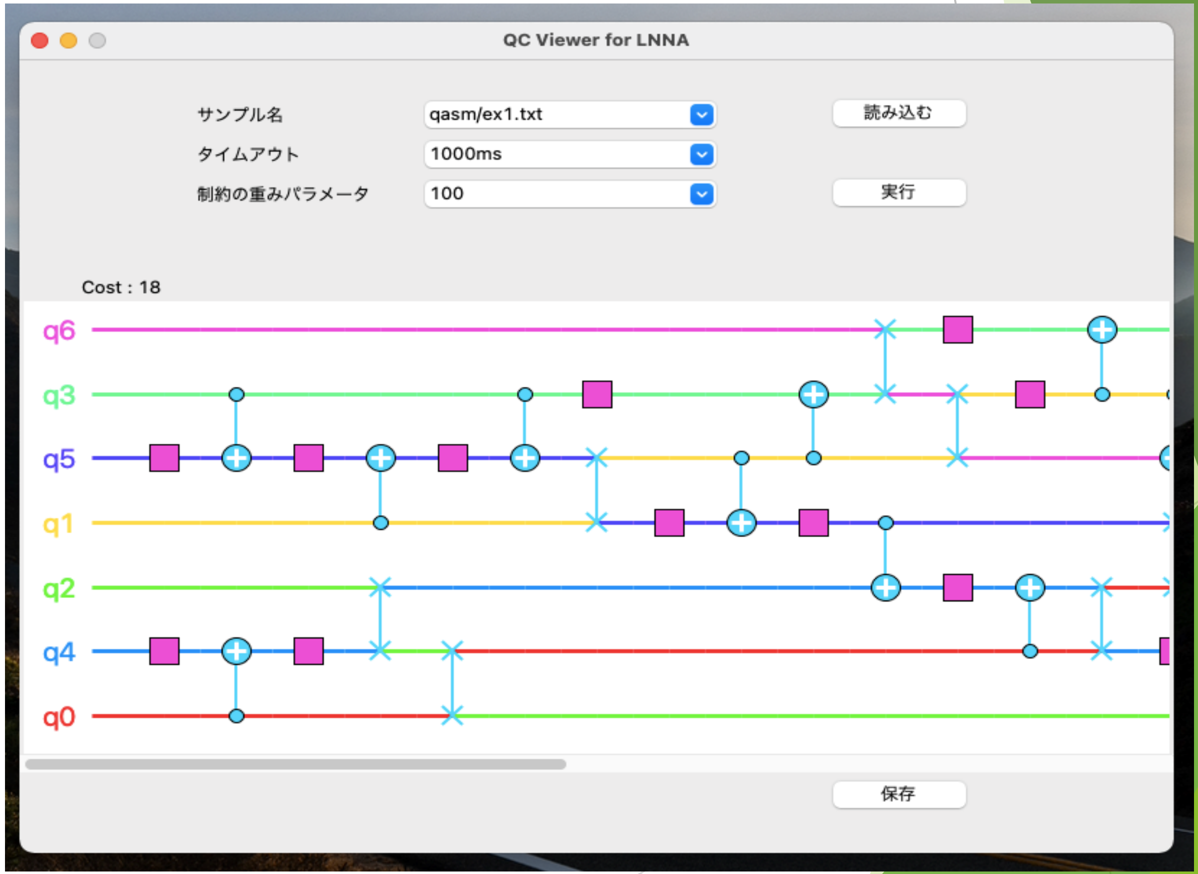
量子回路が描画されたアプリ画面の例



Amplifyの呼び出し機能

- パラメータ調整 + Amplify解法の実行
- 良い解が見つかるまで調整が可能

Amplify解法 実行結果の例

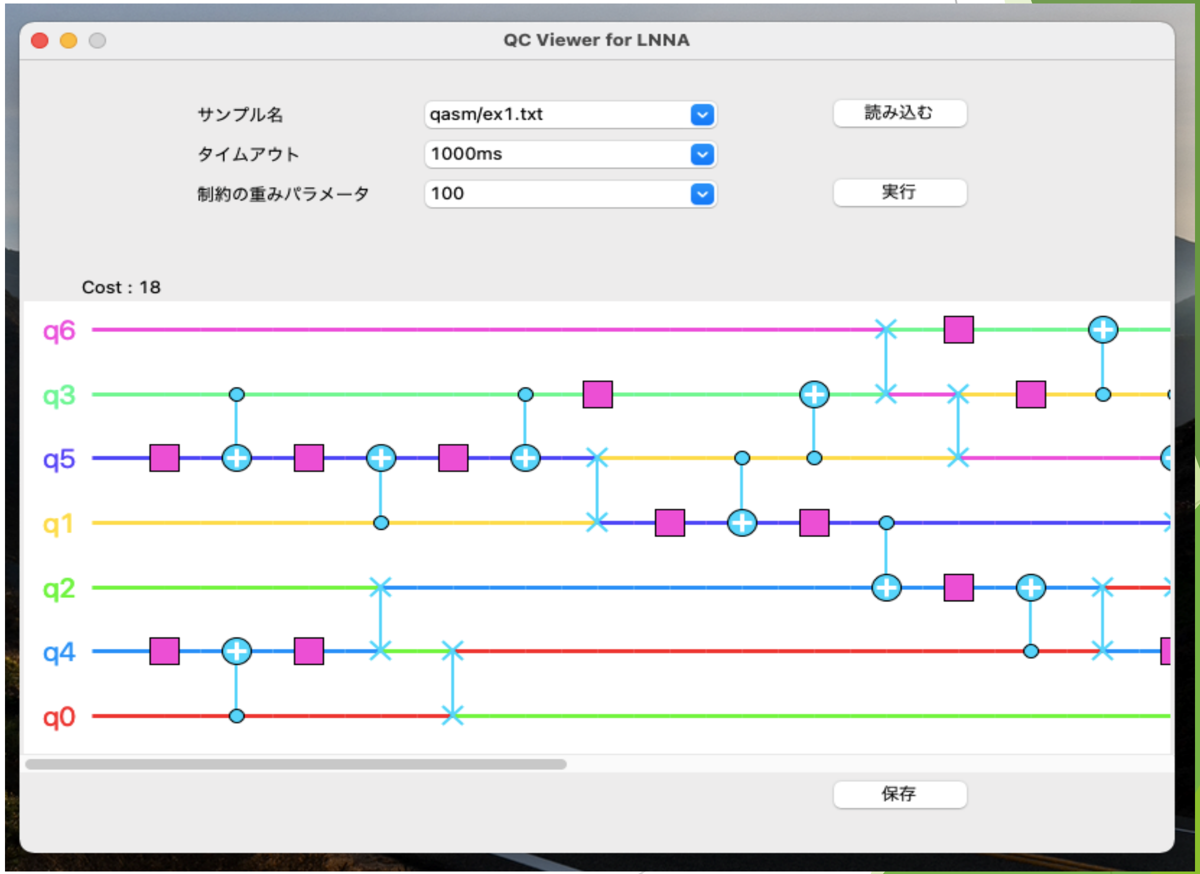


Amplifyの呼び出し機能

- パラメータ調整 + Amplify解法の実行
- 良い解が見つかるまで調整が可能

作業の効率化につながる！

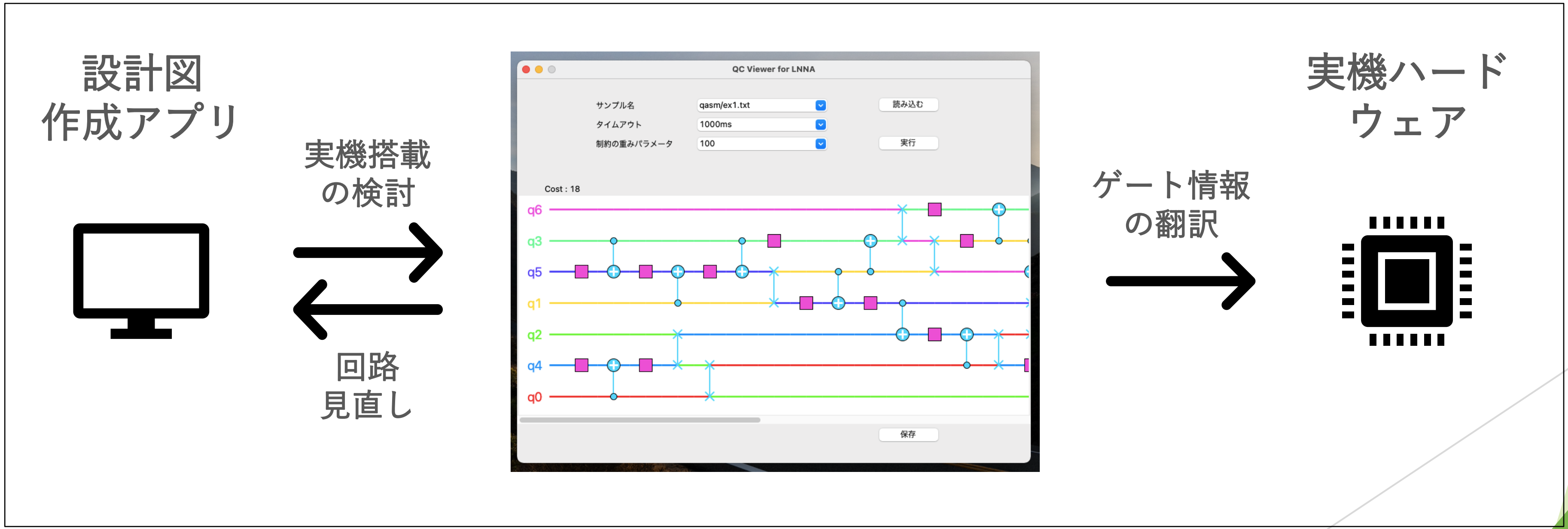
Amplify解法 実行結果の例



OpenQASMを用いた入出力

- 「OpenQASM」という言語を使って読み込み + 書き出しが可能
- 多くのソフトウェアと互換性がある

本アプリの位置付け



アピールポイント

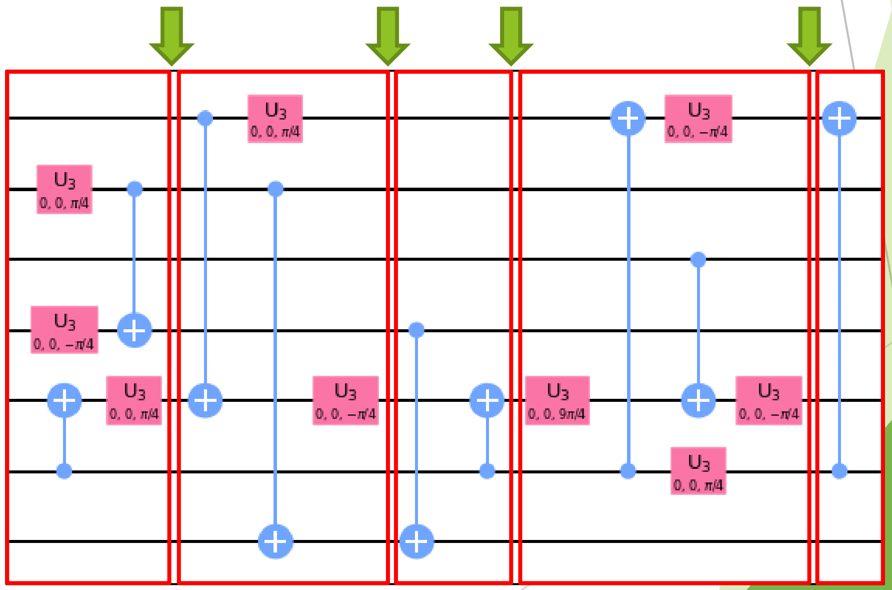
問題設定	実験・評価	アプリへの応用
▶ 「量子ビット割り当て問題」をAmplifyで解く ▶ 「回路全体の最適化」は，**まだ誰も挑戦していない**	▶ 近似により探索空間を削減 → **性能向上** ▶ **圧倒的な高速化に成功**	▶ 回路のリアルタイム描画 → **作業の効率化** ▶ **他アプリとの互換性**
新規性・進歩性	技術力	実用性

補助スライド

問題設定の詳細

- CXゲートを含むそれぞれのレイヤーに対して，論理ビットの配置を決定する
  - 全てのレイヤーで，CXゲートの物理的制約が満たされる必要がある
  - レイヤー間に挿入するSWAPゲートの個数を減らしたい

- 1次元アーキテクチャの場合，SWAPゲートの個数は転倒数に等しい
  - 転倒数 ＝「順番が入れ替わったペアの数」



バイナリ変数を用いた定式化

- 各レイヤーにおける論理ビットは $[0, 1, \cdots, N-1]$ の並び替えとなる
- $\boldsymbol{Q_{mnv}}$：「レイヤー m において，物理ビット n は論理ビット v に対応する」
  - MN^2 個のバイナリ変数が必要

- one-hot 制約
  - 「物理ビットは単一ビットのみに対応する」：$\sum_{v=0}^{N-1} \boldsymbol{Q_{mnv}} = 1$
  - 「論理ビットは単一ビットのみに対応する」：$\sum_{n=0}^{N-1} \boldsymbol{Q_{mnv}} = 1$

- CXゲートによる制約
  - 作用させる物理ビットは隣り合っていなければならない
  - ペナルティ関数：$\sum_{(a,b)\in[CX-gates]} \sum_{(i,j),|i-j|\geq 2} \boldsymbol{Q_{mia}} \boldsymbol{Q_{mjb}}$

コスト関数の定式化 (厳密解法)

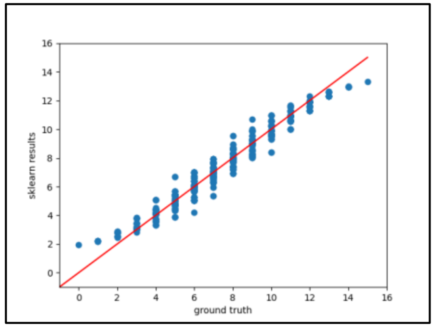
- コスト ＝ SWAPゲートの個数 ＝ 転倒数
  - 厳密に定式化が可能
- 隣り合うレイヤー A, B 間における，論理ビットの移動 C を考える
  - $C_{ij} = 1 \quad \leftrightarrow \quad A[i] = B[j]$
- 補助変数+制約の追加による定式化
  - $S_{ijv} = A_{iv} \cdot B_{jv} \quad \leftrightarrow \quad penalty = 3S_{ijv} + A_{iv}B_{jv} - 2S_{ijv}(A_{iv} + B_{jv})$
  - $C_{ij} = \sum_v S_{ijv}$
  - $cost = \sum_{0 \le i_1 < i_2 < N} \sum_{0 \le j_2 < j_1 < N} \ C_{i_1 j_1} \cdot C_{i_2 j_2}$

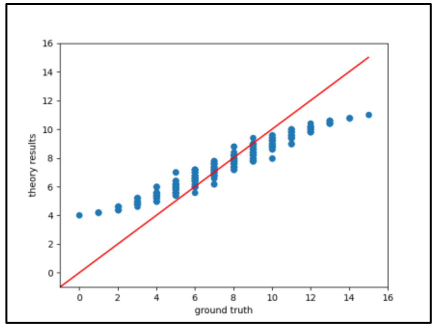
厳密解法の実行結果

- 出力結果のコストが大きすぎる or 解が見つからないという結果に．．．
- 探索空間の大きさに対して，制約を満たす組が少ないことが原因か
  - 用いたバイナリ変数の合計 $= \log_2(\text{探索空間の状態数}) = MN^3 + 2MN^2 - N^3 - N^2$
  - $\log_2(\text{解となる状態数}) < \log_2(N!)^M \approx \frac{1}{\log 2} M(N \log N - N)$
- ほとんど全てが「ハズレ」だった

転倒数の近似によるアプローチ

- 性能向上のため，近似解法を採用して状態数を削減することに
  - 最初に用意したバイナリ変数だけを使って，転倒数を2次で近似する
- 重回帰分析によるフィッティング，期待値による推定を行った





重回帰分析，期待値による転倒数の推定結果
（横軸が正しい値，縦軸が推定値）

- 2つの推定結果は，互いに一次関数の関係になっていた
  - [重回帰分析による推定結果] $= \frac{2N-2}{\mathrm{N}}$ [期待値による推定結果] $- \frac{(N-1)(N-2)}{4}$

コスト関数の定式化 (近似解法)

- 期待値による推定結果 $\approx \left\{\sum_{0\le i,j<N}\left(\frac{i+j}{2}-\frac{ij}{N-1}\right)\cdot C_{ij}\right\}$
- 重回帰分析による推定結果へ変換： $\mathrm{y}=\frac{2N-2}{\mathrm{N}}x-\frac{(N-1)(N-2)}{4}$

- → 転倒数 $\approx \frac{2N-2}{\mathrm{N}}\left\{\sum_{0\le i,j<N}\left(\frac{i+j}{2}-\frac{ij}{N-1}\right)\left(\sum_{v=0}^{N-1}A_{iv}\,B_{jv}\right)\right\}-\frac{(N-1)(N-2)}{4}$ と書ける
  - レイヤー $m,\ m+1$ 間においては， $A_{iv}=\boldsymbol{Q}_{\boldsymbol{miv}}$， $B_{jv}=\boldsymbol{Q}_{(\boldsymbol{m+1})\boldsymbol{jv}}$

- 代入すると，以下のように整理できる
  - $\underbrace{\sum_{m=0}^{M-2}\left\{\sum_{0\le i,j<N}\frac{(N-1)(i+j)-2ij}{N}\left(\sum_{v=0}^{N-1}\boldsymbol{Q}_{\boldsymbol{miv}}\,\boldsymbol{Q}_{(\boldsymbol{m+1})\boldsymbol{jv}}\right)\right\}}_{\text{この部分を最小化したい}}-\underbrace{(M-1)\frac{(N-1)(N-2)}{4}}_{\text{定数項（無視してOK）}}$

Amplify解法モデルの構成

- $cost = \sum_{m=0}^{M-2}\left\{\sum_{0\le i,j<N}\frac{(N-1)(i+j)-2ij}{N}\left(\sum_{v=0}^{N-1}\boldsymbol{Q}_{\boldsymbol{miv}}\,\boldsymbol{Q}_{(\boldsymbol{m+1})\boldsymbol{jv}}\right)\right\}$
- $constraint = \sum_{m=0}^{M-1}\left\{\underbrace{\sum_{v=0}^{N-1}\left(1-\sum_{n=0}^{N-1}\boldsymbol{Q}_{\boldsymbol{mnv}}\right)^2+\sum_{n=0}^{N-1}\left(1-\sum_{v=0}^{N-1}\boldsymbol{Q}_{\boldsymbol{mnv}}\right)^2}_{\text{one-hot 制約}}+\underbrace{\sum_{(a,b)\in[CX-gates]}\ \sum_{(i,j),|i-j|\ge 2}\ \boldsymbol{Q}_{\boldsymbol{mia}}\,\boldsymbol{Q}_{\boldsymbol{mjb}}}_{\text{CXゲートによる制約}}\right\}$

- $model = constraint\ \times\ \lambda + cost$ として構成した
  - $model$ の項数(= モデルの規模)は $O(MN^3)$ 個