LNNA向け回路生成における swapゲート個数最小化問題

東京大学大学院　情報理工学系研究科　電子情報学専攻
長谷川研究室　修士1年　内藤壮俊

自己紹介

- プログラミング経験
  - 競技プログラミング (C++)
  - ゲーム開発 (Unity C#)
  - 研究，ウェブ開発 (Python, JavaScript)

- 量子コンピューティングの経験
  - 量子ゲート型：IBM Quantum Challenge 2020 に参加した程度
  - 量子アニーリング型：今回が初めて

参加したきっかけ

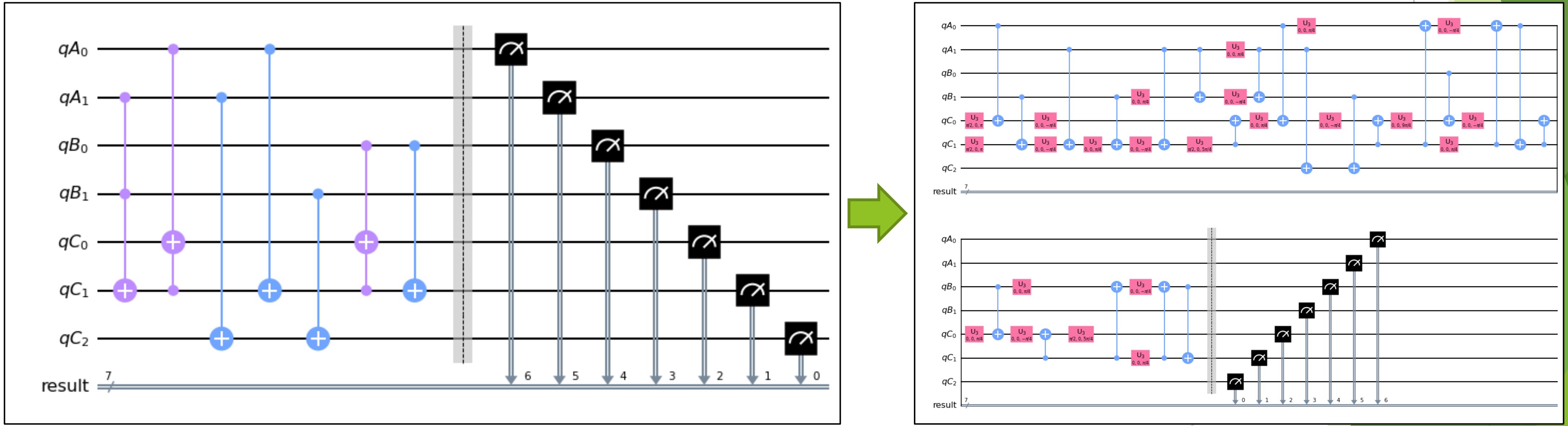
- 量子アニーリングを使って，量子ゲートの回路設計を支援できないか？
  - コラボって感じがしてカッコいい

- 現存するゲート型量子コンピュータは数十ビット程度の規模なので，扱う問題の大きさとしてちょうど良い…？
  - アニーリングが先行している分だけ，アドバンテージが活かせそう

テーマ説明

量子ゲート型計算機

- ゲート通過 = ユニタリ変換 による状態の変化を利用
- 任意の回路はU3ゲート(1入力)とCXゲート(2入力)に展開することができる
  - CXゲートにおけるエラー率はU3ゲートの10倍程度



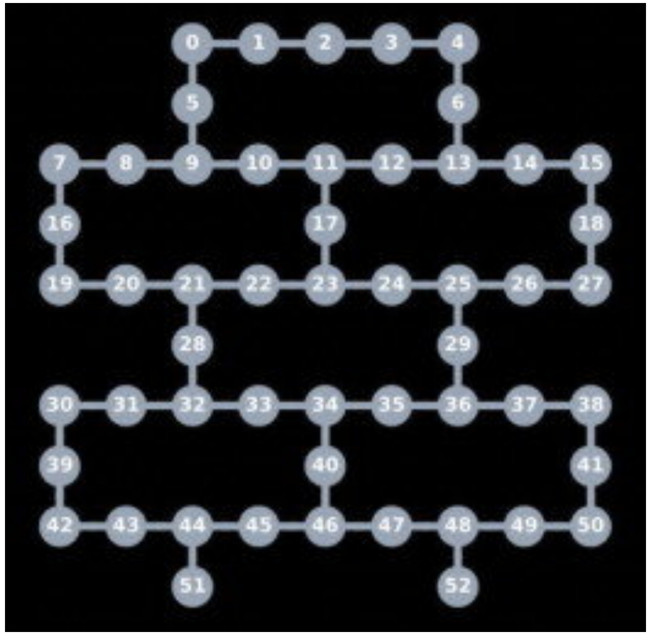
図：IBM Qにて作成した量子回路の例．
(左：回路設計段階における見た目．　右：U3ゲートとCXゲートに展開した結果．)

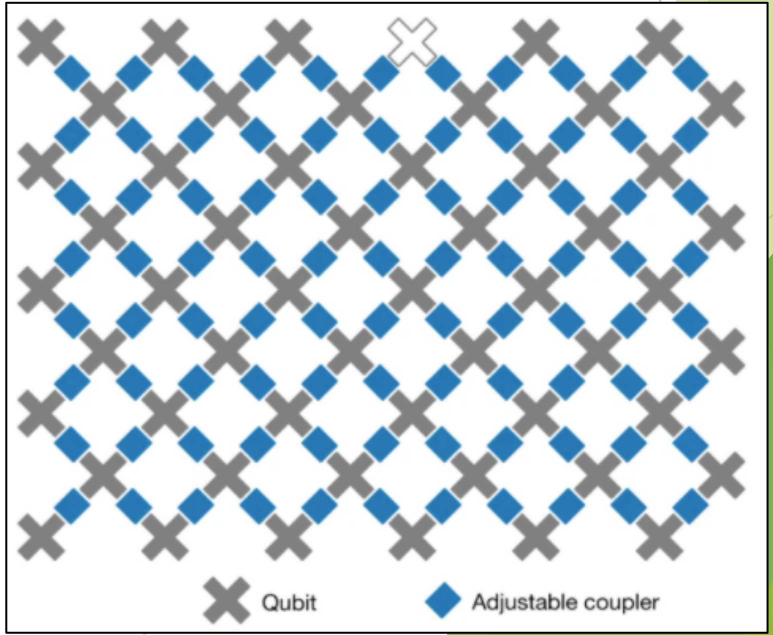
量子回路の実機搭載

- 「設計図上のビット」と「デバイス上のビット」の対応を考える必要がある
- CXゲートは物理的に隣り合う2ビットにしか作用できない
- 離れている場合は？
  - → SWAPゲートを使って入れ替える必要がある

左図：
IBM「Rochester」における
量子ビットの配置.

右図：
Google「Sycamore」における
量子ビットの配置.





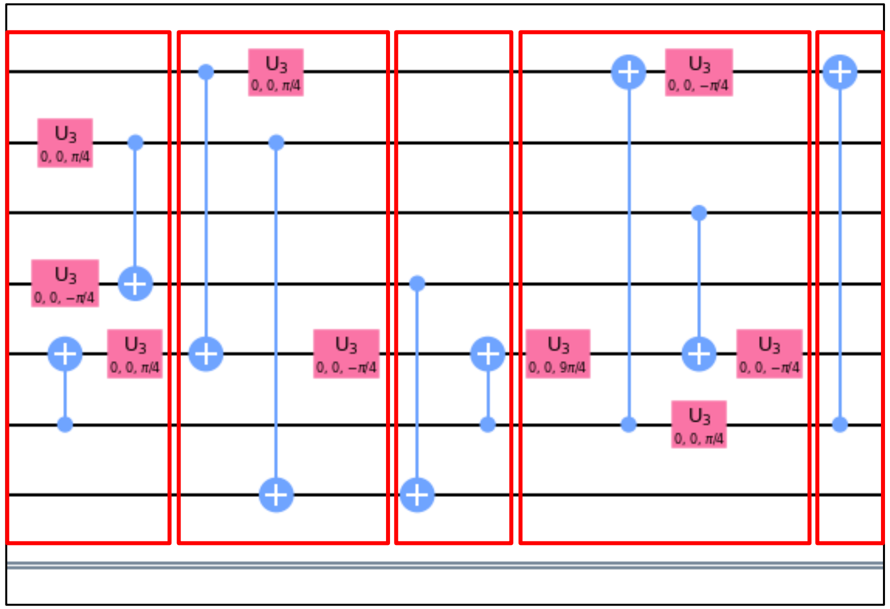
Linear Nearest Neighbor Architecture

- 量子ビットが1次元状に並んでいるアーキテクチャのこと

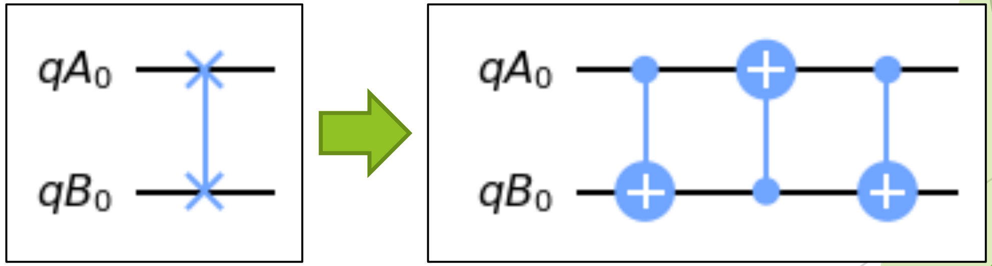
- SWAPゲートによる並び替えはバブルソートと非常に似ている
  - 挿入する個数 = バブルソートの交換回数 = 転倒数 となるため，扱いやすい問題に
  - 一般の場合は，グラフの形状に大きく左右されてしまう

扱う問題

- 量子ビットを共有しないように，CXゲートをレイヤーにまとめる
- CXゲートを含むレイヤーのそれぞれに対し，量子ビットの配置を決定する
- 並び替えの際に使用するSWAPゲートの個数を減らしたい
  - SWAPゲート1つに，CXゲートを3つも使ってしまうため



図：レイヤーの構成.



図：SWAPゲートの構成.

古典的なアプローチ (動的計画法による高速化)

- 量子ビットの個数 N に対し，各レイヤーにおける配置は $N!$ 通り
- レイヤーの枚数 M に対して，全体の取りうる状態数は $(N!)^M$ 通り

- 配置に対する暫定的なコストを持っておくことで，
空間計算量 $O(M \cdot N!)$，時間計算量 $O(M \cdot (N!)^2)$ で解くことができる

- $N = 10$ で $(N!)^2 \approx 1.3 \times 10^{13}$ なので，小規模の回路にしか適用できない．

ソルバーの実装

バイナリ変数を用いた定式化

- 各レイヤーにおける量子ビットは $[0, 1, \cdots, N-1]$ の並び替えとなる
- Q_{mnv}：「レイヤー m における n 番目は，設計図における v 番目に対応する」
  - MN^2 個の量子ビットが必要

- one-hot 制約
  - 「設計図におけるビットは1つのビットに対応する」：$\sum_{n=0}^{N-1} Q_{mnv} = 1$
  - 「レイヤーにおけるビットは1つのビットに対応する」：$\sum_{v=0}^{N-1} Q_{mnv} = 1$

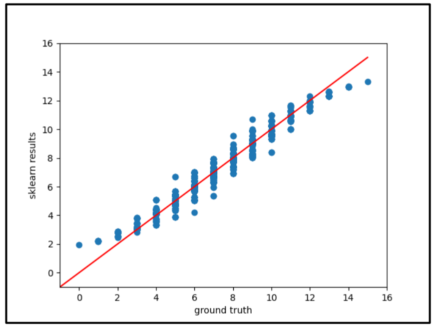
- CXゲートによる制約
  - 作用させる2ビットは隣り合っていなければならない
  - ペナルティ関数：$\sum_{(a,b)\in[2-input-gates]} \sum_{(i,j),|i-j|\geq 2} Q_{mia} Q_{mjb}$

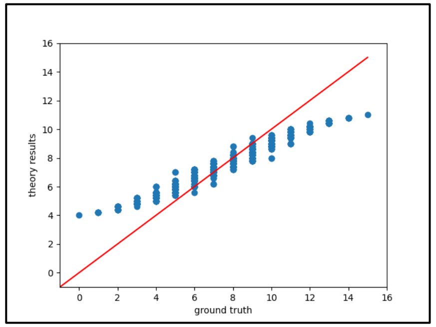
コスト関数の定式化 (初期案)

- $C_{ij} = 1 \;\leftrightarrow\; C[i] = j \;\leftrightarrow\; A[i] = B[j]$ を用意
  - レイヤー A, B 間でシンボルが一致している組み合わせ
  - $C_{ij} = \sum_{v=0}^{N-1} A_{iv} B_{jv}$
- 転倒数 (= シンボルどうしが入れ替わった回数) を以下のように定式化
  - $cost = \sum_{0 \le i_1 < i_2 < N} \sum_{0 \le j_2 < j_1 < N} C_{i_1 j_1} \cdot C_{i_2 j_2}$
- しかし，うまくいかず…
  - 制約条件 $C_{ij} = \sum_{v=0}^{N-1} A_{iv} B_{jv}$ は，2次多項式の形をしている
  - ペナルティの関数は $\underbrace{\left(C_{ij} - \sum_{v=0}^{N-1} A_{iv} B_{jv}\right)^2}_{\text{2次式の2乗 = 4次式}} = 0$ となり，<u>2次以下の多項式で表せない</u>

転倒数のフィッティング

- 重回帰分析を試した結果
  - 10^{14} オーダーの係数が出てきてしまったが，まあまあ綺麗にフィットしていた
- 期待値による推定を行った結果
  - 係数は計算できるが，のっぺりした分布に…





図：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，縦軸が推定値である．

- 驚くべきことに，両者は互いに一次関数の関係にあった
  - そのため，重回帰分析のフィッティング結果を生成できるように

転倒数の推定モデルの実装

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

- → 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, $A_{iv}=Q_{miv}$, $B_{jv}=Q_{(m+1)jv}$

- 代入すると，以下のように整理できる
  - $\underbrace{\sum_{m=0}^{M-2}\left\{\sum_{0\le i,j<N}\frac{(N-1)(i+j)-2ij}{N}\left(\sum_{v=0}^{N-1}Q_{miv}\,Q_{(m+1)jv}\right)\right\}}_{\text{この部分を最小化したい.}}-\underbrace{(M-1)\frac{(N-1)(N-2)}{4}}_{\text{定数項. 最小化においては無視される.}}$

QUBO形式への変換

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

- $model = constraint \times \lambda + cost$ として構成した
  - $model$ の項数(= モデルの規模)は $O(MN^3)$ 個
  - 制約 >> コストとするために，$\lambda = 100$ と設定

パフォーマンスの評価

評価：コスト最小化の性能比較

- ランダムに生成したデータ10個に対するコストの平均値を記録した
  - 古典的解法は最適解を出力する (ので，必ず 古典的解法 ≤ Amplify解法 となる)
  - Amplify解法においては，出力した結果をもとに厳密なコストを計算した

- $\mathrm{M} = 20$ での誤差が大きくなっている
  - 転倒数の近似が原因？ λ を調整してみる？ $timeout$ を伸ばす？

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

表：それぞれの解法における
コストの平均値.

評価：実行時間の比較

- $M = 5$ にて， N を動かした時の実行時間(秒)を比較した
  - 古典的な $O(M \cdot (N!)^2)$ 解法では $N = 7$ が限界だった

N	3	4	5	6	7	8	9	10	15	20
古典的解法	0.0007	0.0084	0.4551	8.9532	566.95	—	—	—	—	—
Amplify解法	1.8336	1.8184	1.4685	1.1751	1.2766	1.3620	1.3654	1.6483	6.1014	20.047

表： N を動かした時の実行時間の比較.
$6 \leq N$ においてAmplify解法の方が高速となっている.

- N が大きくなると，Amplify解法でも時間がかかる傾向に
  - 量子ビット，制約，コスト関数の準備に時間がかかっていた

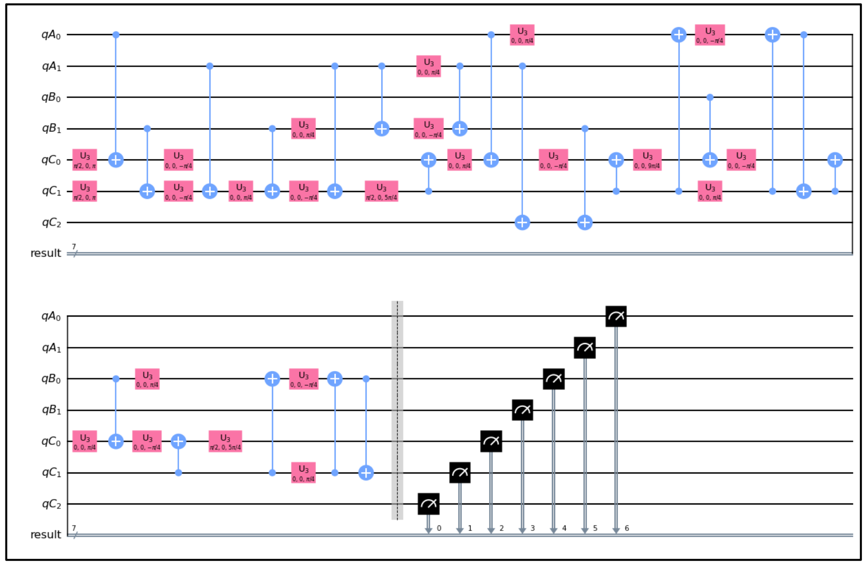
評価：実行時間の見積もり

- 探索にかかる時間は固定だが，制約条件とコスト関数が $O(MN^3)$ 項あるため古典計算がオーバーヘッドとなり，全体的で $O(MN^3)$ となると推測される
- といっても，古典的解法は $O(M \cdot (N!)^2)$ なので飛躍的向上と言える
- $N = 50$ (現時点での最大級の量子ビット数) でも数分あれば計算できるはず

アプリ制作

OpenQASMとの連携

- 「OpenQASM」という言語で書かれた回路を入力できるようにしたい
- U3ゲートとCXゲートに分解済みの回路を入力に用いることとする



図：展開した後の回路.

```
・・・
u3(pi/2,0,pi) qC[0];
cx qA[0],qC[0];
u3(0,0,-pi/4) qC[0];
u3(pi/2,0,pi) qC[1];
cx qB[1],qC[1];
u3(0,0,-pi/4) qC[1];
cx qA[1],qC[1];
u3(0,0,pi/4) qC[1];
cx qB[1],qC[1];
u3(0,0,pi/4) qB[1];
u3(0,0,-pi/4) qC[1];
cx qA[1],qC[1];
cx qA[1],qB[1];
u3(0,0,pi/4) qA[1];
u3(0,0,-pi/4) qB[1];
```

```
cx qA[1],qB[1];
u3(pi/2,0,5*pi/4) qC[1];
cx qC[1],qC[0];
u3(0,0,pi/4) qC[0];
cx qA[0],qC[0];
u3(0,0,pi/4) qA[0];
u3(0,0,-pi/4) qC[0];
cx qC[1],qC[0];
u3(0,0,9*pi/4) qC[0];
cx qB[0],qC[0];
u3(0,0,-pi/4) qC[0];
cx qC[1],qA[0];
u3(0,0,-pi/4) qA[0];
u3(0,0,pi/4) qC[1];
cx qC[1],qA[0];
cx qA[0],qC[1];
```

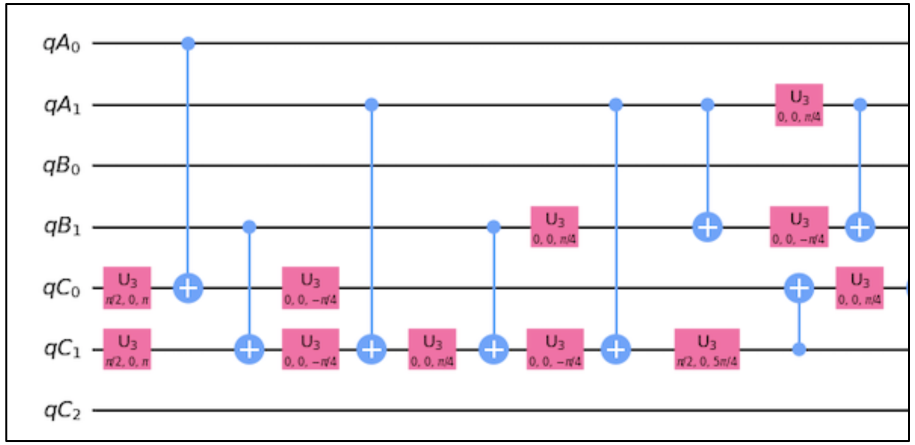
```
cx qC[1],qC[0];
u3(0,0,pi/4) qC[0];
cx qB[0],qC[0];
u3(0,0,pi/4) qB[0];
u3(0,0,-pi/4) qC[0];
cx qC[1],qC[0];
u3(pi/2,0,5*pi/4) qC[0];
cx qC[1],qB[0];
u3(0,0,-pi/4) qB[0];
u3(0,0,pi/4) qC[1];
cx qC[1],qB[0];
cx qB[0],qC[1];
cx qA[1],qC[2];
cx qB[1],qC[2];

・・・
```

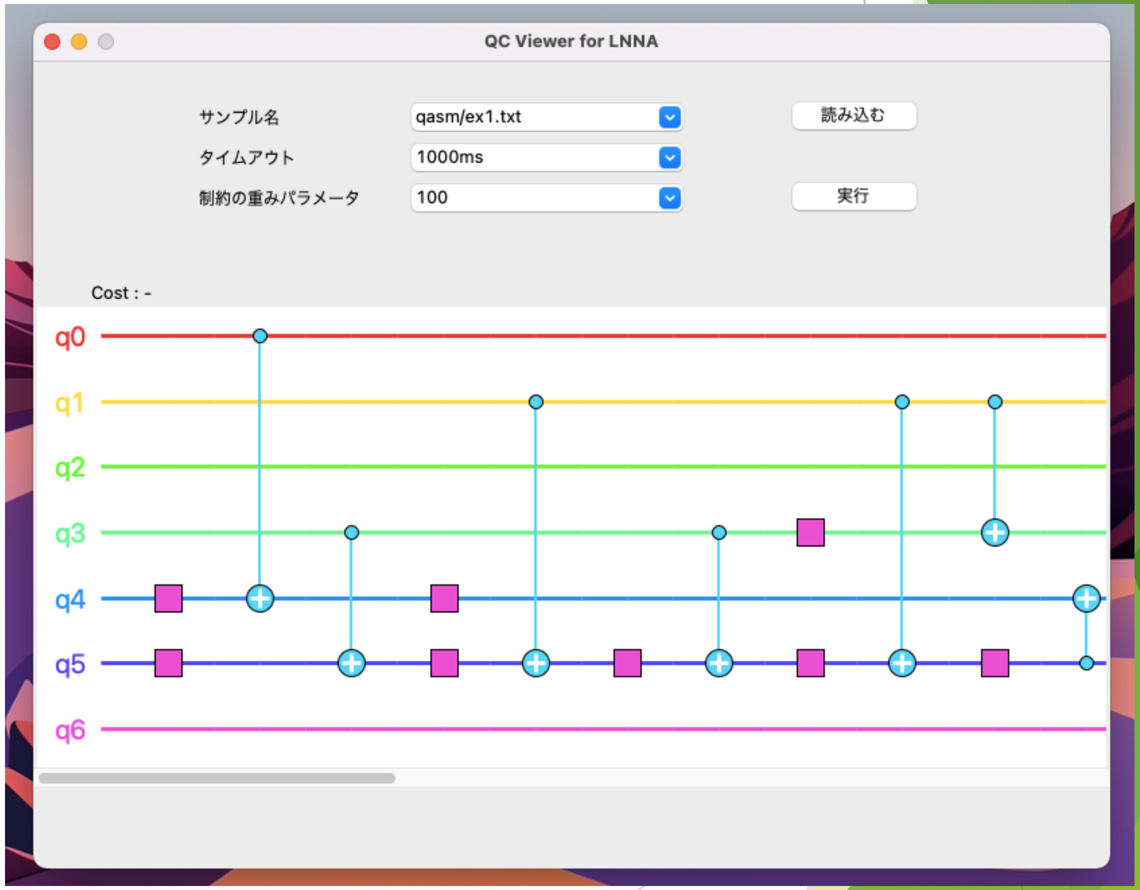
図：出力されるOpenQASMプログラム.

アプリ機能紹介：量子回路の描画

- tkinterというライブラリで実装
- 設計図における回路と，LNNA向けの回路 (Amplify解法の結果) を描画する



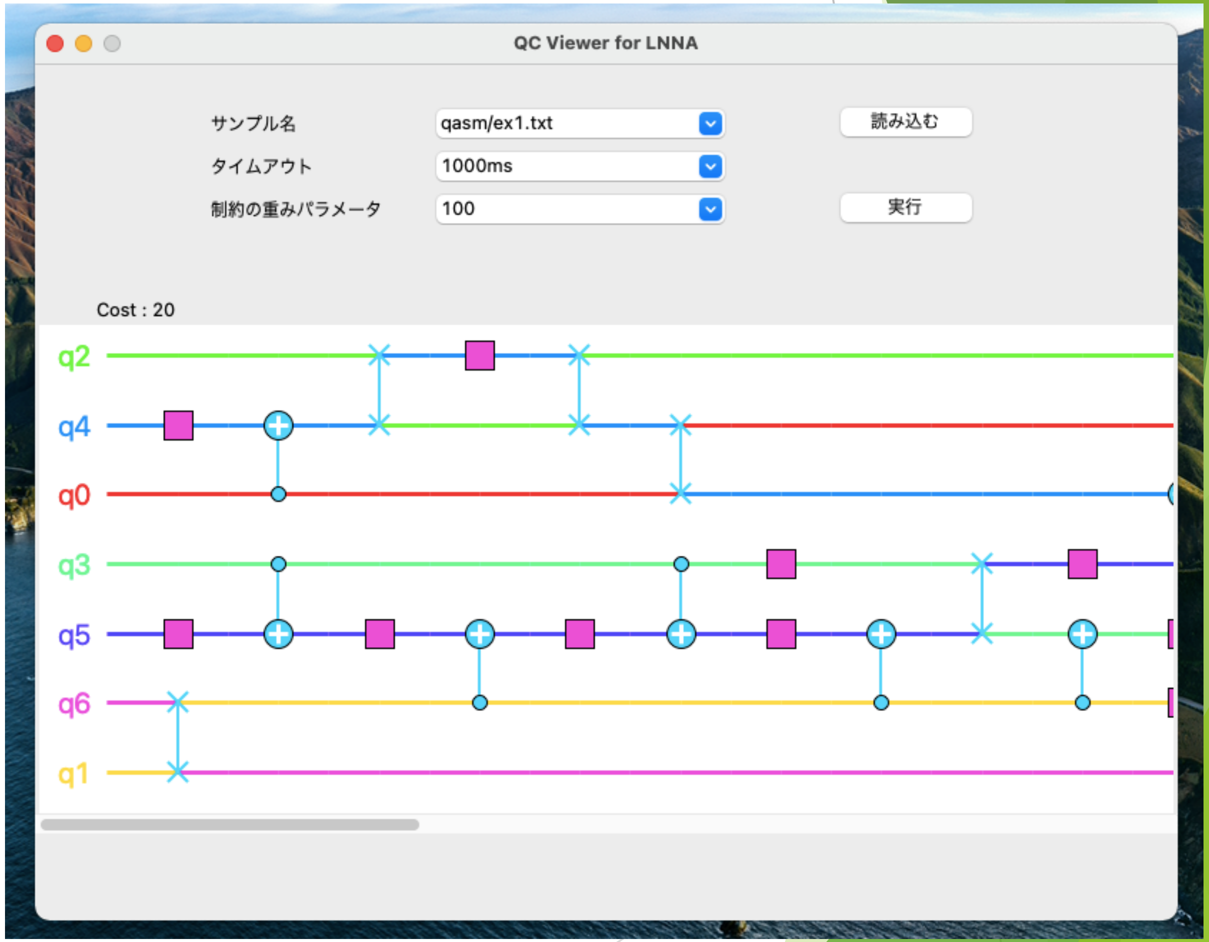
図：読み込んだ回路設計図の一部.
(IBM Q上で描画．)



図：ビジュアライザ上での描画結果.

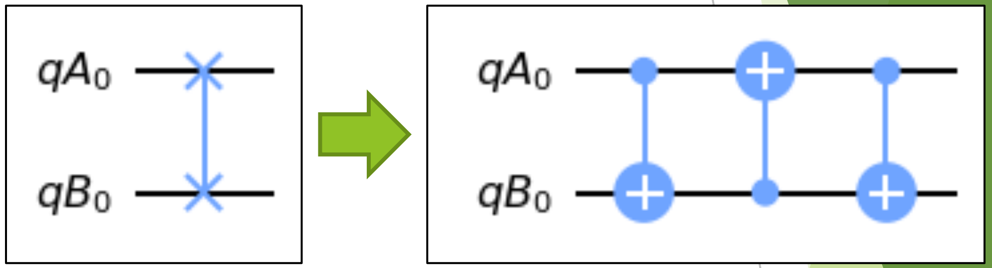
アプリ機能紹介：Amplifyの呼び出し

- 第三章で構成した解法を呼び出す
  - タイムアウト (探索時間) の調整可
  - 制約重みパラメータの調整可
- コスト (SWAPゲートの個数) の表示
  - 右の例ではコスト = 20 となった.
- 並び替え後の回路の描画
  - SWAPゲートは「X—X」で描画されている

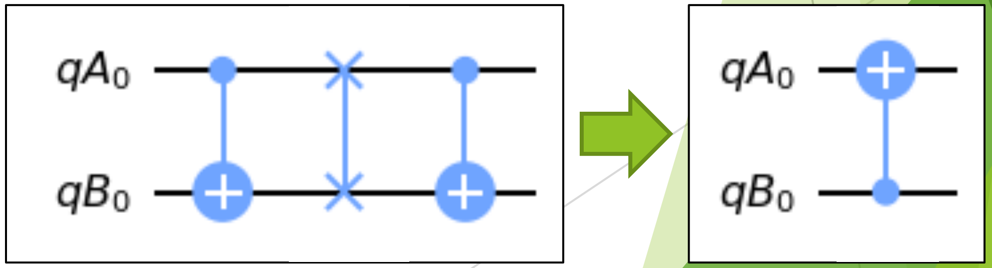


アプリ機能紹介：OpenQASM形式で出力

- SWAPゲートを3つのCXゲートに置き換えることでOpenQASM形式に書き換えることが可能
  - 設計図段階の回路を入力して，アプリでLNNA向けの回路に変換して，その結果を出力できるように



- CXゲートの個数が最も少なくなるように置き換えを実行
  - 上下反転を考えると，置き換えは2通りある
  - CXゲートが相殺する場合がある



アプリを使ってみる