

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

第一章 テーマ説明

量子計算のアプローチ

▶ 量子アニーリング型

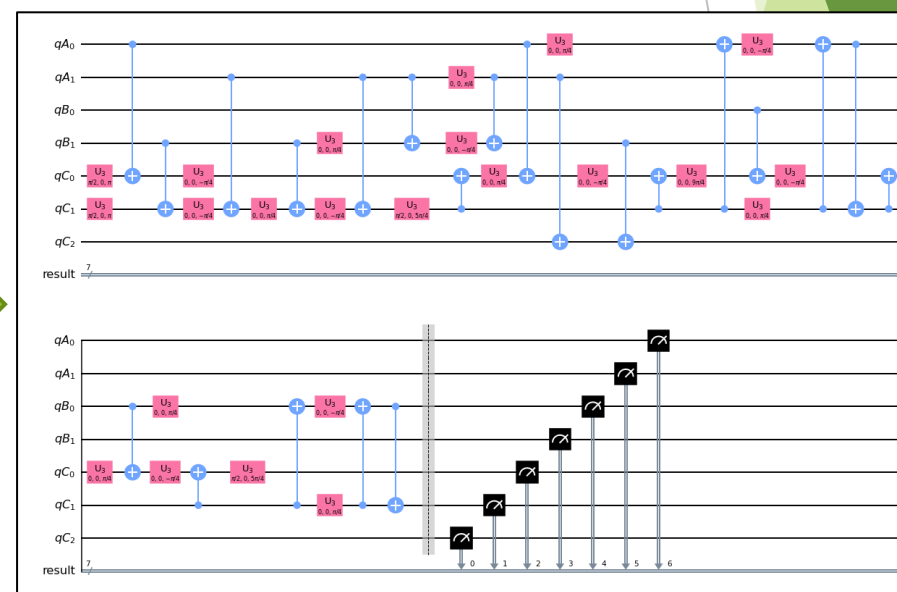
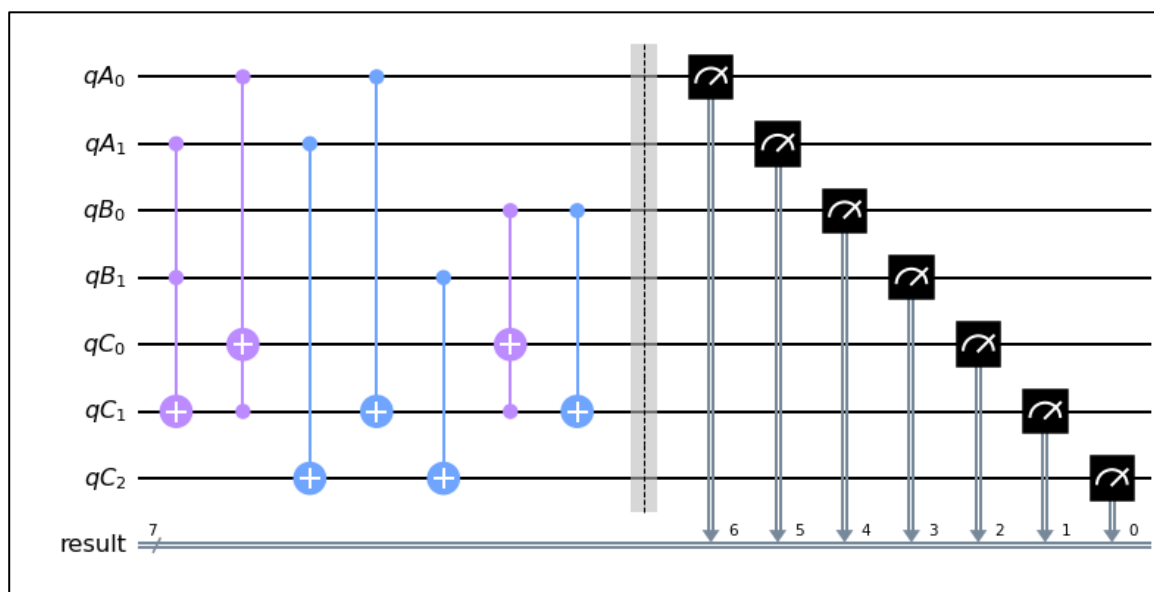
- ▶ 多変数二次関数の最小化問題(組み合わせ最適化)を解ける
 - ▶ それぞれの変数は0か1のどちらかを取る
- ▶ 量子ビットの個数は数千個オーダーと、小～中規模な問題なら実用化可能
- ▶ Fixstars Amplifyで使っているのはこっち

▶ 量子ゲート型

- ▶ 汎用計算機. 古典コンピュータも(理論的には)シミュレートできる
- ▶ 量子ビットの個数は非常に少ない(数十個のオーダー)
- ▶ GoogleやIBMが取り組んでいる計算機はこっち
- ▶ 今回扱うテーマはこの話

量子回路

- ▶ 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

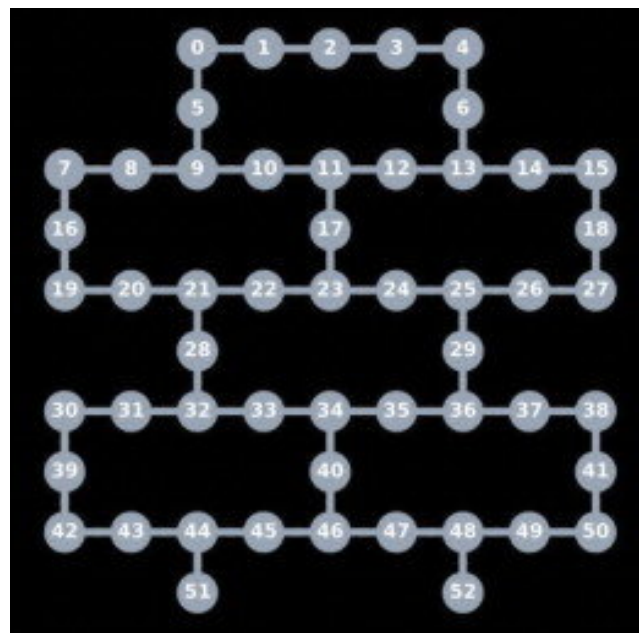


図：IBM Qにて作成した量子回路の例。
(左：回路設計段階における見た目。 右：U3ゲートとCXゲートに展開した結果。)

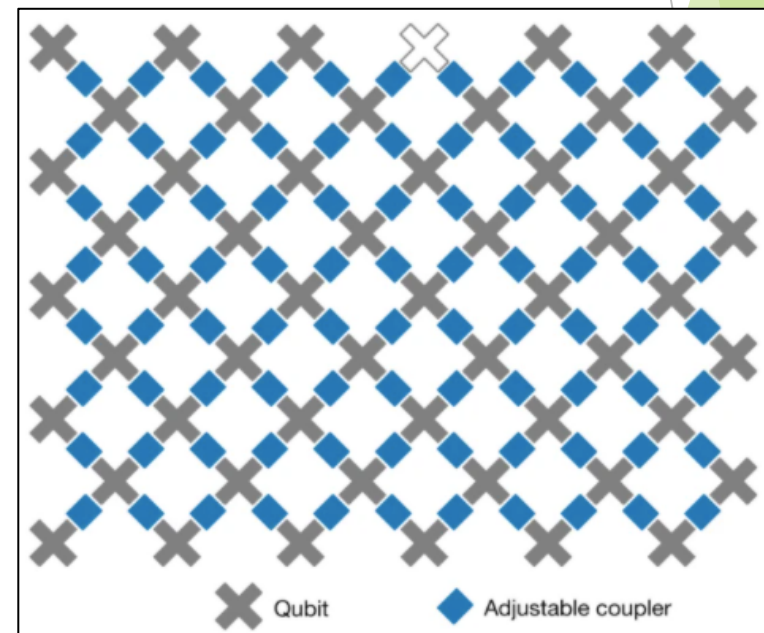
量子回路の実機搭載

- ▶ 「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ゲートは隣り合う2ビットにしか作用させられない
- ▶ 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。

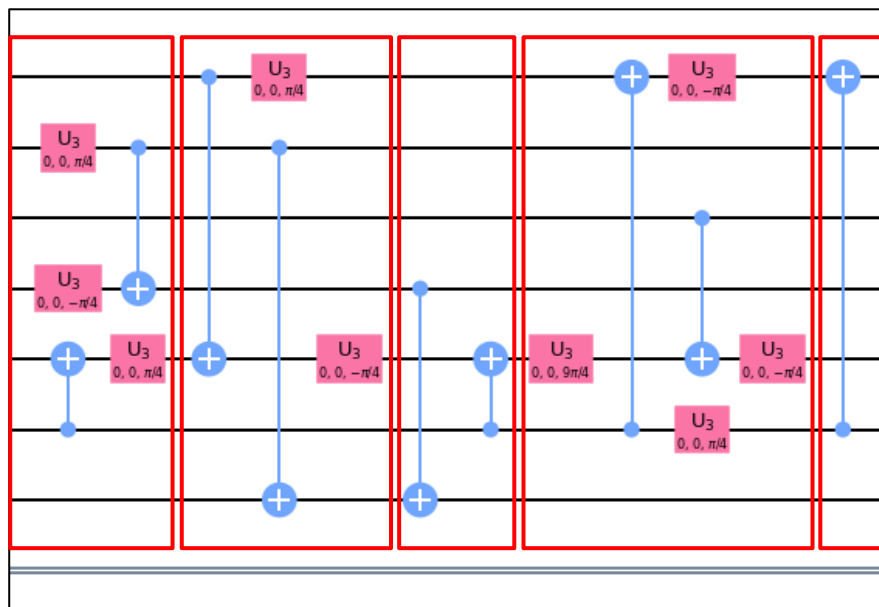


Linear Nearest Neighbor Architecture

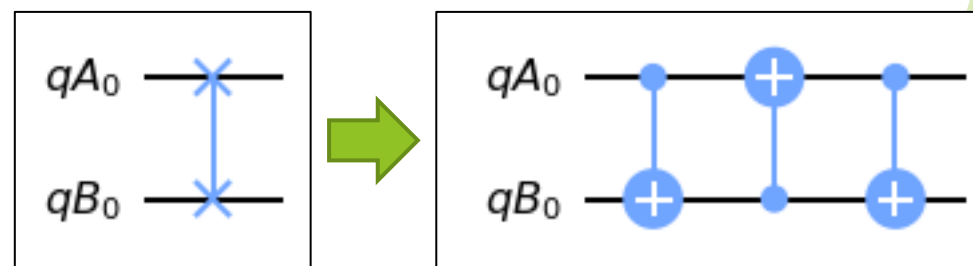
- ▶ 量子ビットが1次元状に並んでいるアーキテクチャのこと
- ▶ SWAPゲートによる並び替えはバブルソートと非常に似ている
 - ▶ 挿入する個数 = バブルソートの交換回数 = 転倒数 となるため, 扱いやすい問題に

扱う問題

- ▶ 量子ビットを共有しないCXゲートを一つの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.

第二章

とりあえず実装してみる

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 $[0, 1, \dots, N - 1]$ 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いなければならない
 - ▶ ペナルティ関数 : $\sum_{(a,b) \in [2\text{-input-gates}]} \sum_{(i,j), |i-j| \geq 2} Q_{mia} Q_{mjb}$

コスト関数の定式化

- ▶ SWAPゲートの個数 = バブルソートの交換回数 の総和を減らしたい
 - ▶ $A : [3, 0, 4, 1, 2] \rightarrow B : [2, 1, 3, 4, 0]$ の交換回数は？
- ▶ $B = [0, 1, 2, \dots, N - 1]$ なら, 初期配置 A における転倒数を求めれば良い
 - ▶ 転倒数 : $A_i > A_j$ and $i < j$ となる組の総数
 - ▶ 全ての組に対して足し合わせればいいので $O(N^2)$
- ▶ $B = [0, 1, 2, \dots, N - 1]$ とは限らない場合, どう定式化する... ?

コスト関数の定式化 (初期案)

- ▶ 置換の合成も置換なので, $(A \rightarrow B) = (C \rightarrow [0, 1, 2, \dots, N-1])$ となる C を見つけてあげれば良い?
 - ▶ $A: [3, 0, 4, 1, 2] \rightarrow B: [2, 1, 3, 4, 0]$ なら,
 $C: [2, 4, 3, 1, 0] \rightarrow [0, 1, 2, 3, 4]$ となる
 - ▶ 「 B の配置を $[0, 1, 2, 3, 4]$ とみなした時の A の配置」を求める問題
- ▶ $C_{ij} = 1 \Leftrightarrow C[i] = j \Leftrightarrow A[i] = B[j]$
 - ▶ $A[i]$ のシンボルと $B[j]$ のシンボルが等しいかどうか
 - ▶ ベクトルの内積で定式化できる?
 - ▶ $C_{ij} = \sum_{v=0}^{N-1} A_{iv} B_{jv}$

コスト関数の定式化 (初期案)

- ▶ いくら待っても実行可能解が見つからない.....
- ▶ 何がダメだったか
 - ▶ 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 $\frac{(C_{ij} - \sum_{v=0}^{N-1} A_{iv} B_{jv})^2}{2\text{次式の2乗} = 4\text{次式}} = 0$ となり, 2次以下の多項式で表せない
- ▶ 任意の A, B 間の転倒数は4次多項式で定式化されるので, この厳密な値を組み込むことがそもそも不可能だった

第三章

転倒数のフィッティング

転倒数のフィッティング

- ▶ A, B 間の転倒数をどうにかして2次以下で表したい
 - ▶ 厳密解は諦めて, 2次以下でフィッティングを試みる
- ▶ 使える変数
 - ▶ 「同じシンボルがどこからどこへ移動したか」
 - ▶ つまり, $C_{ij} = \sum_{v=0}^{N-1} A_{iv} B_{jv}$ のこと
 - ▶ C_{ij} の線形和で転倒数を表現したい, という話になる
 - ▶ $B = [0, 1, 2, \dots, N-1]$ とは限らないので, シンボル間の大小関係は使えない
 - ▶ 変数の個数は N^2 個
 - ▶ 対称性の除去により, N が奇数の時は $\frac{1}{4}(N+1)^2$ 個, 偶数の時は $\frac{1}{4}(N^2 + 2N)$ 個に

重回帰分析によるアプローチ

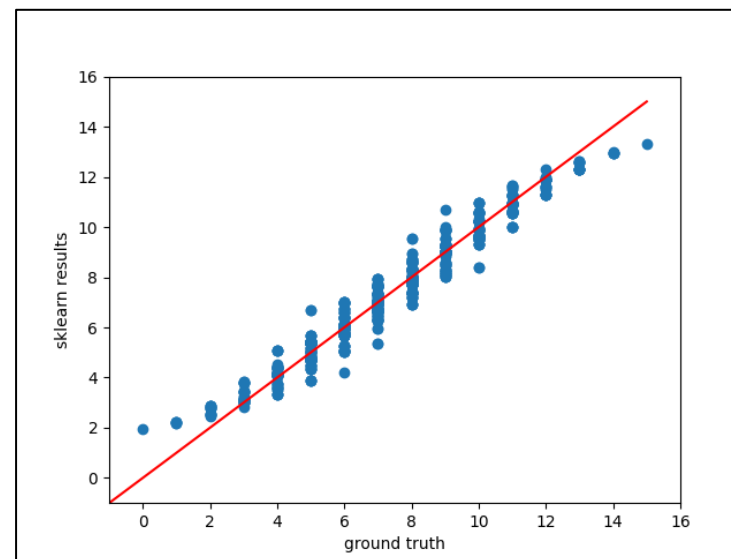
- ▶ 重回帰分析なら scikit-learn でできるので、これで転倒数を推定してみる
- ▶ 係数, 切片ともにすごい値になってしまった

```
coefficient  
[-4.71281238e+12 -1.46172282e+12 -5.40448318e+13 -5.40448318e+13  
 -1.46172282e+12 -4.71281238e+12  1.78936673e+12 -5.07937422e+13  
 -5.07937422e+13  1.78936673e+12 -1.03376851e+14 -1.03376851e+14]  
intercept  
212600593697132.16
```

図：係数, 切片の出力結果.
 10^{14} オーダーの数字が見られる.

- ▶ でも, プロットしてみると良い感じに見える

図： $N = 6$ の場合の推定結果.
(横軸：正解 縦軸：推定値)



期待値によるアプローチ

- ▶ ある並び替えにおいて、左から $i + 1$ 番目のシンボルが $j + 1$ 番目に動いた際、他のシンボルと入れ替わっている確率を求める
 - ▶ $A : [3, 0, 4, 1, 2] \rightarrow B : [2, 1, 3, 4, 0]$ なら, $(i, j) = (1, 4)$
 - ▶ この場合, 0 は3つのシンボル $(1, 2, 4)$ と入れ替わっている

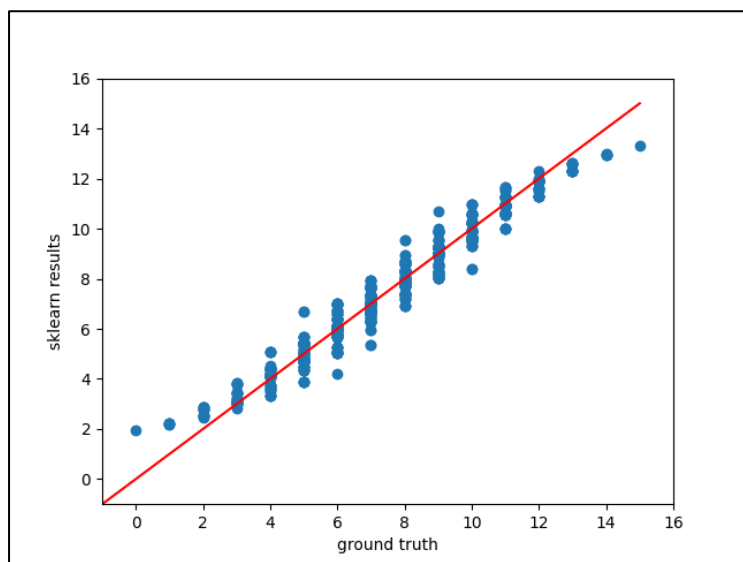
- ▶ 自分以外の $N - 1$ 個のシンボルに対し, 左側 \rightarrow 右側 or 右側 \rightarrow 左側 と動いた確率を足し合わせ, (重複を考えて) 2で割ると求められる

$$\frac{1}{2}(N-1) \left\{ \frac{i}{N-1} \cdot \frac{N-1-j}{N-1} + \frac{N-1-i}{N-1} \cdot \frac{j}{N-1} \right\} = \frac{i+j}{2} - \frac{ij}{N-1}$$

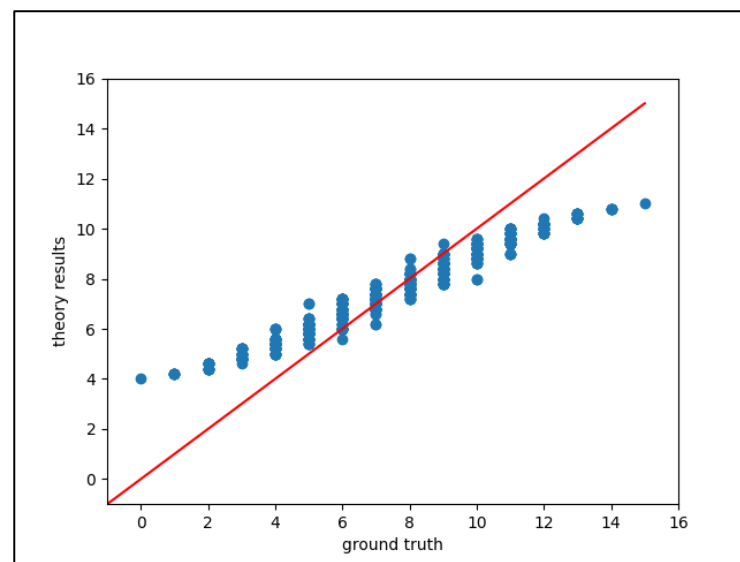
- ▶ 転倒数 $\approx \sum_{0 \leq i, j < N} \left(\frac{i+j}{2} - \frac{ij}{N-1} \right) \cdot c_{ij}$ として予測を行ってみた

期待値によるアプローチ

- ▶ のっぺりした分布になってしまった...
- ▶ 一方で、分布の形状は重回帰分析による推定結果に非常に似ている
 - ▶ 両者の間に何か対応が見えるのでは...？



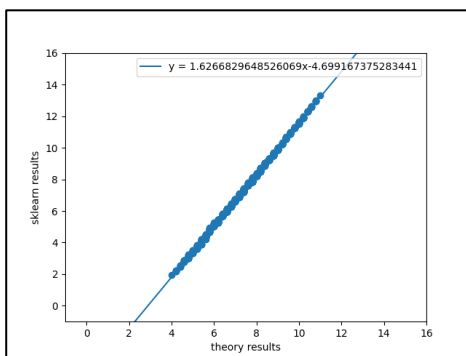
図：重回帰分析による推定結果



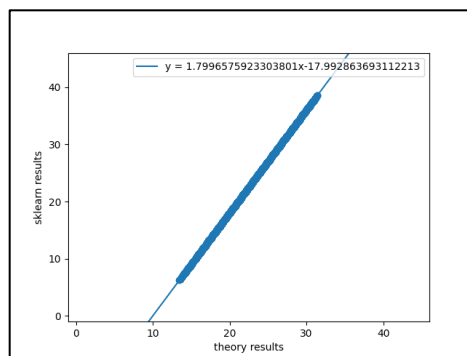
図：期待値による推定結果

予測結果の相関を調べる

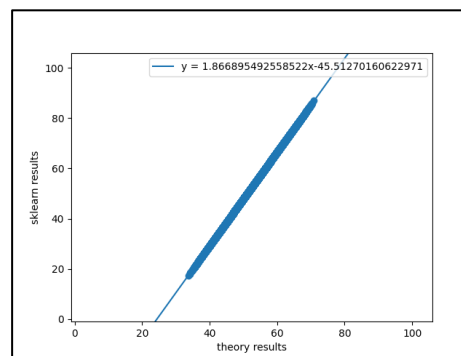
- ▶ 横軸を「期待値による推定」，縦軸を「重回帰分析による推定」としてプロット
- ▶ 下図は，左から $N = 6$, $N = 10$, $N = 15$, $N = 20$ の場合
 - ▶ $N \geq 10$ に対しては，500000 サンプルをランダムに(復元抽出で)取り出して分析した



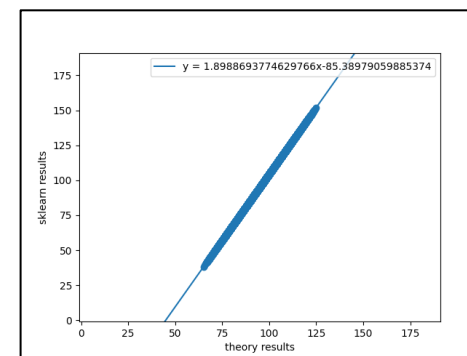
$$y = 1.627x - 4.699$$
$$R^2 = 0.999355$$



$$y = 1.800x - 17.993$$
$$R^2 = 0.999956$$



$$y = 1.867x - 45.513$$
$$R^2 = 0.999978$$



$$y = 1.899x - 85.390$$
$$R^2 = 0.999991$$

- ▶ 1次関数の関係になっていると言える ($1 - R^2$ は縦軸の計算誤差によるものか)

転倒数の推定

- ▶ 「重回帰分析による推定結果」は、転倒数を一番良く推定できると考えられる
- ▶ 「期待値による推定結果」に1次関数を作用させると、
「重回帰分析による推定結果」に一致させられることが分かった
- ▶ → 期待値による推定結果だけから「最も良いモデル」を構成できる！
 - ▶ 例えば $N = 6$ ($y = 1.627x - 4.699$) なら,
転倒数 $\approx 1.627 \left\{ \sum_{0 \leq i, j < N} \left(\frac{i+j}{2} - \frac{ij}{N-1} \right) \cdot c_{ij} \right\} - 4.699$ として表現できる

一次関数の係数についての考察

- ▶ 量子ビットの個数 N に対して, 一次関数は $y = \frac{2N-2}{N}x - \frac{(N-1)(N-2)}{4}$ と書けそう
 - ▶ しかし, 予想の証明にはまだ至らず...
- ▶ (誤差) = (予測値) - (実測値) として記録

N	3	4	5	6	7	8	9	10	15	20
傾き	1.3333	1.5000	1.5987	1.6267	1.6856	1.7425	1.7738	1.7990	1.8664	1.8992
切片	-0.500	-1.500	-2.974	-4.699	-7.196	-10.379	-13.933	-17.975	-45.490	-85.419
傾きの誤差	0	0	-0.0013	-0.0400	-0.0287	-0.0075	-0.0040	-0.0010	-0.0003	-0.0008
切片の誤差	0	0	0.026	0.301	0.304	0.121	0.067	0.025	0.010	0.081

表: N を動かした時の, 一次関数の傾きと切片の変化.
誤差は小さく, 上記の予想とおおよそ一致していると言える.

転倒数の推定モデルの実装

- ▶ 「同じシンボルがどこからどこへ移動したか」 : $C_{ij} = \sum_{v=0}^{N-1} A_{iv} B_{jv}$
- ▶ 転倒数 $\approx \left\{ \sum_{0 \leq i, j < N} \left(\frac{i+j}{2} - \frac{ij}{N-1} \right) \cdot C_{ij} \right\}$ に $y = \frac{2N-2}{N}x - \frac{(N-1)(N-2)}{4}$ を適用
- ▶ \rightarrow 転倒数 $\approx \frac{2N-2}{N} \left\{ \sum_{0 \leq i, j < N} \left(\frac{i+j}{2} - \frac{ij}{N-1} \right) \left(\sum_{v=0}^{N-1} A_{iv} B_{jv} \right) \right\} - \frac{(N-1)(N-2)}{4}$ と書ける
 - ▶ 複雑な見た目だけど、ちゃんと2次多項式で表現できている！
- ▶ あとは、この式をもとに目的関数を設計するだけ

コスト関数の定式化 (改良案)

- ▶ 目的関数：各レイヤー間の転倒数 (の推定値) の和

- ▶
$$\sum_{m=0}^{M-1} \left\{ \frac{2N-2}{N} \left\{ \sum_{0 \leq i, j < N} \left(\frac{i+j}{2} - \frac{ij}{N-1} \right) \left(\sum_{v=0}^{N-1} A_{iv} B_{jv} \right) \right\} - \frac{(N-1)(N-2)}{4} \right\}$$
 - ▶ レイヤー m , $m+1$ 間においては, $A_{iv} = Q_{miv}$, $B_{jv} = Q_{(m+1)jv}$ 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」

- ▶ 代入すると, 以下のように整理できる

- ▶
$$\underbrace{\sum_{m=0}^{M-1} \left\{ \sum_{0 \leq i, j < N} \frac{(N-1)(i+j)-2ij}{N} \left(\sum_{v=0}^{N-1} Q_{miv} Q_{(m+1)jv} \right) \right\}}_{\text{この部分を最小化したい.}} - \underbrace{(M-1) \frac{(N-1)(N-2)}{4}}_{\text{定数項. 最小化においては無視される.}}$$

この部分を最小化したい.

定数項.
最小化においては無視される.

第四章

改良案の実装とその評価

バイナリ変数を用いた定式化(再掲)

- ▶ 各レイヤーにおける量子ビットは $[0, 1, \dots, N - 1]$ 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いなければならない
 - ▶ ペナルティ関数 : $\sum_{(a,b) \in [2\text{-input-gates}]} \sum_{(i,j), |i-j| \geq 2} Q_{mia} Q_{mjb}$

QUBO形式への変換

- ▶ $cost = \sum_{m=0}^{M-1} \left\{ \sum_{0 \leq i, j < N} \frac{(N-1)(i+j)-2ij}{N} \left(\sum_{v=0}^{N-1} Q_{miv} Q_{(m+1)jv} \right) \right\}$
- ▶ $constraint = \sum_{m=0}^M \left\{ \sum_{v=0}^{N-1} (1 - \sum_{n=0}^{N-1} Q_{mnv})^2 + \sum_{n=0}^{N-1} (1 - \sum_{v=0}^{N-1} Q_{mnv})^2 + \sum_{(a,b) \in [2-input-gates]} \sum_{(i,j), |i-j| \geq 2} Q_{mia} Q_{mjb} \right\}$
- ▶ $model = constraint \times \lambda + cost$ 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
 - ▶ N, M が大きくなる = コストが大きくなるにつれて, λ も大きくするべきか

評価：用いたデータ

- ▶ CXゲートを含むレイヤーを M 枚生成した ($M = 5$)
 - ▶ レイヤー1枚あたり, $1 \sim \left\lfloor \frac{N}{2} \right\rfloor$ 個のCXゲートを含むように構成
 - ▶ CXゲートの個数, 作用先はランダムに決定
 - ▶ CXゲートどうして量子ビットの共有は起こらない
- ▶ 古典解法 ($N \leq 6$), Amplify解法 ($N \leq 26$) を10回ずつ試してコストや実行時間を比較した
 - ▶ 実験では, $timeout = 1\text{ s}$ として実行
 - ▶ コストの比較においては, $M = 20$ も検討した

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 \leq 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
- ▶ $M = 20$ 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 ($M = 5$)	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

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

評価：実行時間の内訳

- ▶ Amplify解法に対して、実行時間を以下の3つに分けて測定した
 - ▶ 「準備時間」：量子ビット, 制約条件, コスト関数の生成
 - ▶ 「探索時間」："result = solver.solve(model)" でかかる時間
 - ▶ 「解析時間」：出力結果のデコード, 結果に基づく厳密なコストの計算
- ▶ 準備時間が大きく増える傾向にあった

N	5	10	15	20
準備時間(s)	0.013 (0.004)	0.473 (0.194)	3.607 (1.232)	25.818 (13.732)
探索時間(s)	1.316 (0.377)	1.191 (0.053)	1.436 (0.248)	1.886 (0.497)
解析時間(s)	0.001 (0.000)	0.003 (0.000)	0.005 (0.001)	0.006 (0.000)

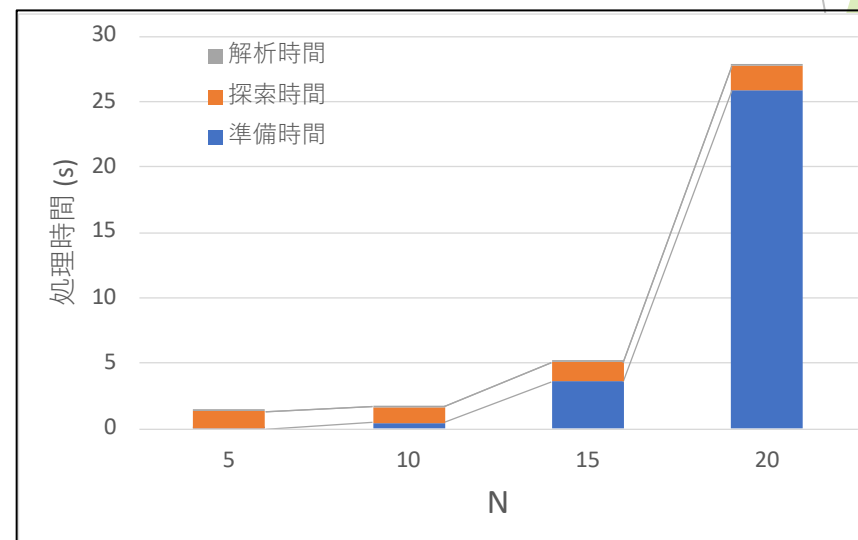


表 / 図：実行時間の内訳。N が大きくなるにつれて準備時間が増える傾向が見てとれる。
カッコ内は(不偏)標準偏差。

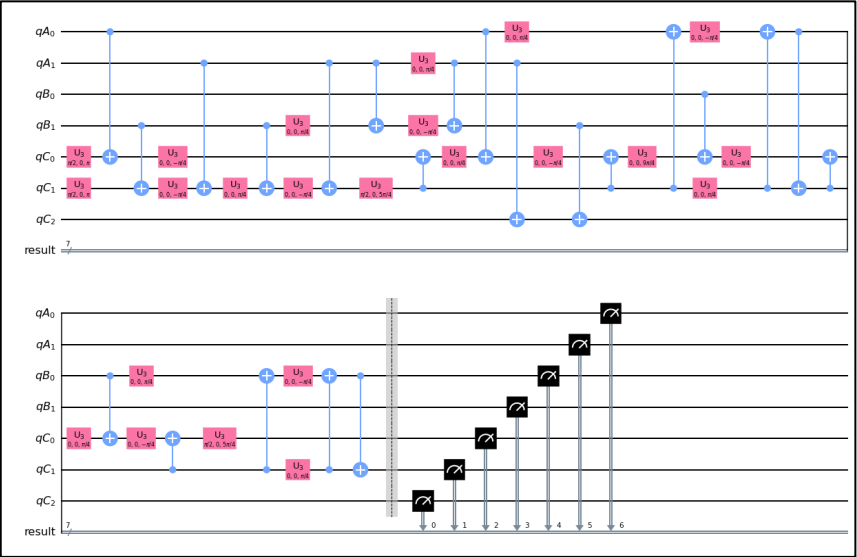
評価：実行時間の見積もり

- ▶ 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

第五章 実問題への応用

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.

<pre>... 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];</pre>	<pre>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(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]; ...</pre>
--	--

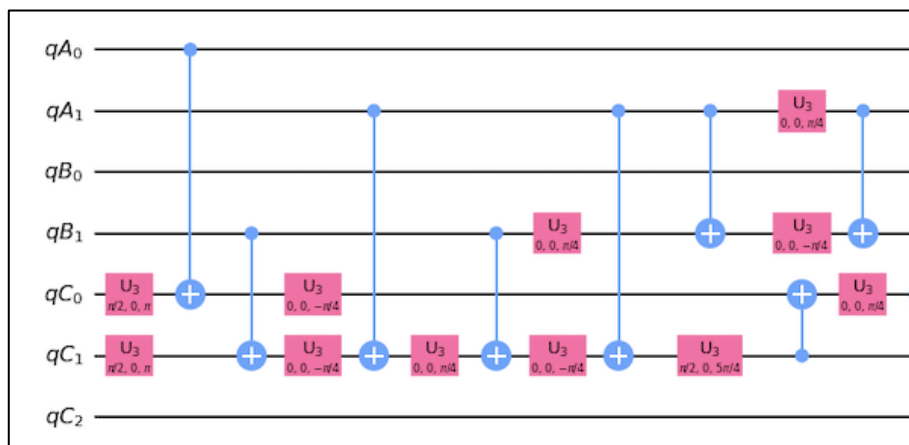
図：出力されるOpenQASMプログラム.

用いた量子回路の例

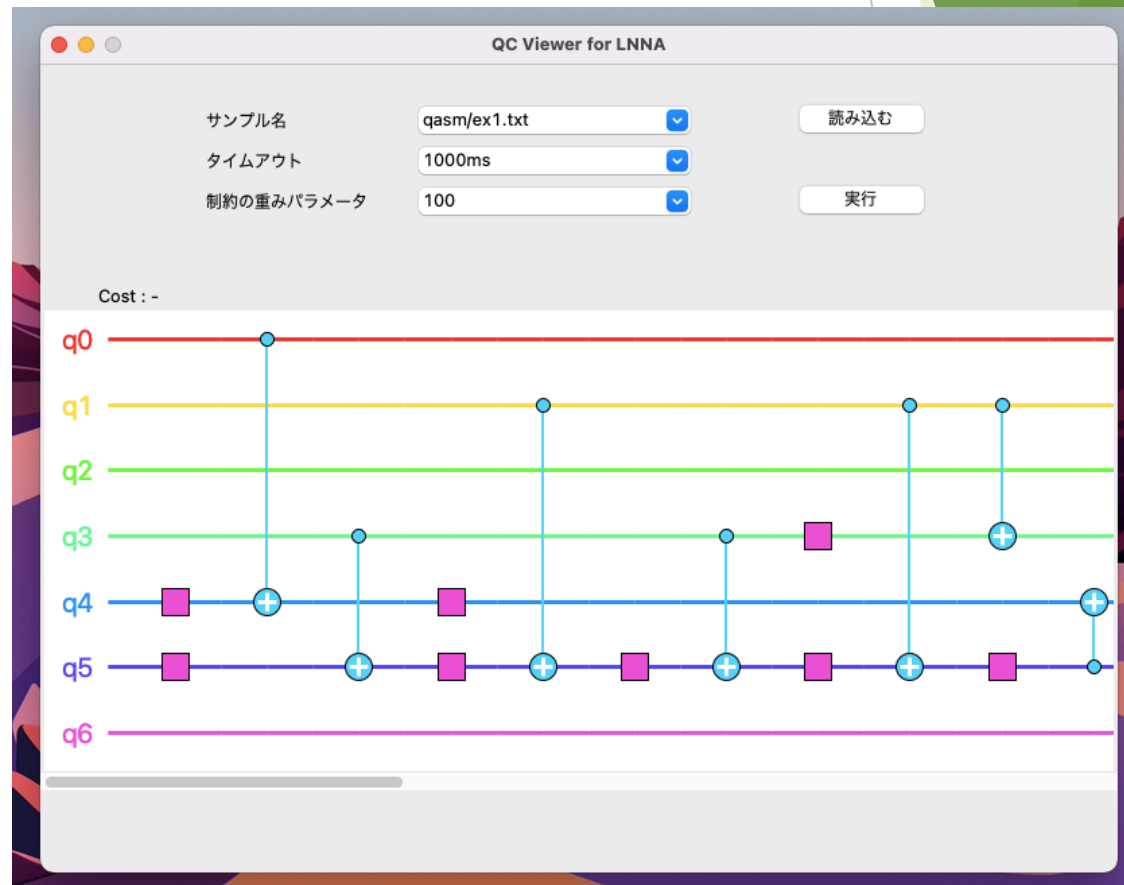
- ▶ 例1 : 2bit加算器 (7bit)
 - ▶ 入力 : $2+2 = 4$ bit
 - ▶ 出力 : 3bit
- ▶ 例2 : 8bit回文判定器 (10bit)
 - ▶ 入力 : 8bit
 - ▶ 出力 : なし
 - ▶ 補助 : 2bit

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



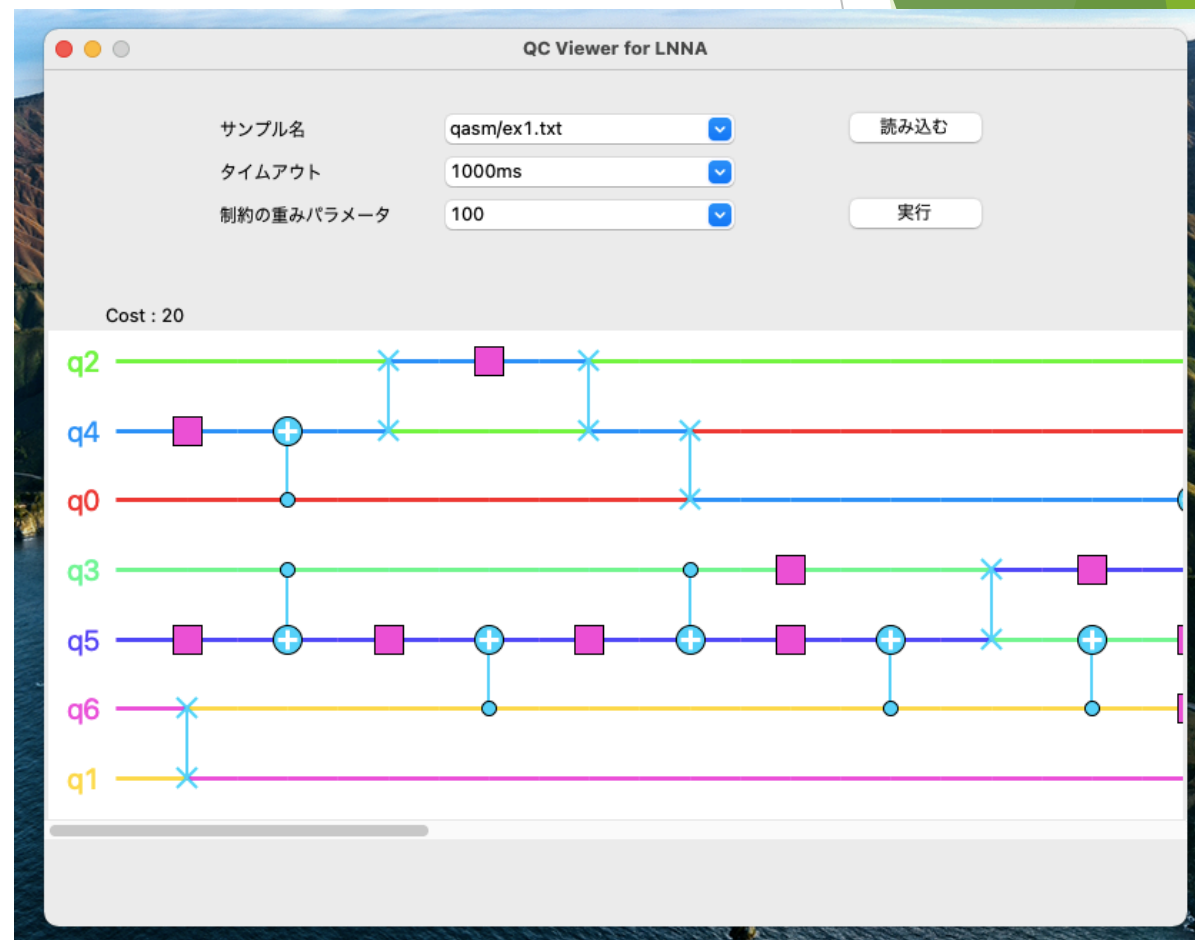
図：読み込んだ回路設計図の一部。
(IBM Q上で描画。)



図：ビジュアライザ上での描画結果。

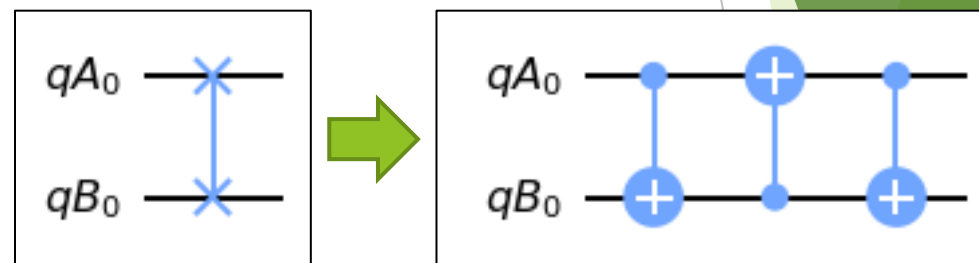
機能紹介：Amplifyの呼び出し

- ▶ 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

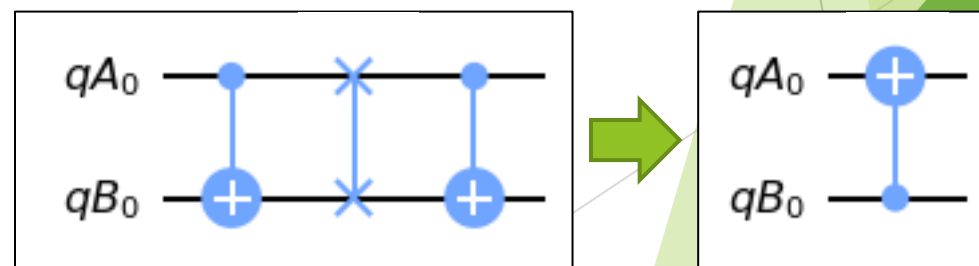


機能紹介：OpenQASM形式で回路を出力

- ▶ 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，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



第六章 アプリを使ってみる

