Vol.2022-QS-5 No.23
2022/3/25

ISAAQ: イジングマシンを活用した量子コンパイラ

内藤 壮俊[1]　長谷川 禎彦[1]　松田 佳希[2,3]　田中 宗[4,3]

概要：量子コンパイラは，論理回路として表現された量子プログラムを受け取り，デバイス上で実行可能かつ論理的に等価な回路を合成するソフトウェアである．近年主流となっている NISQ デバイスは，物理的に接続された量子ビット間でしか量子ゲートを作用させられない，操作によって生じたエラーが蓄積するといった特性を有している．そのため NISQ デバイスを対象とする量子コンパイラは，接続関係の制約を満たしながら，回路のコストすなわちゲート操作回数が少なくなるように回路を出力しなければならない．このコンパイル操作において最も重要なタスクは，論理回路中の量子ビットをデバイス上の量子ビットに割り当てるタスクである．これは NP 困難であり，出力回路のコストに大きく影響する問題となっている．私たちの提案する量子コンパイラ「ISAAQ (ISing mAchine Assisted Quantum compiler)」は，出力回路のコストを QUBO モデルとして表現し，イジングマシンを用いた解の探索，およびその解に基づいた回路合成を実行する．ISAAQ は，実行結果に基づいた QUBO モデルの更新，複数イジングマシンによる並列実行，デバイス上の経路を考慮したコスト削減といった，他にはない特徴を多く持っている．IBM QX5 および IBM QX20 を対象とした実験では，ISAAQ は既存の QUBO 手法やその他のアルゴリズムよりも低コストな回路を出力できていることが確認され，本提案手法の有効性が示された．

1. はじめに

近年，量子デバイスの急速な発展により，量子計算は現実のものとなった．現在主流の量子デバイスは NISQ(Noisy Intermediate-Scale Quantum) デバイスと呼ばれており，その名の通り，これらは誤り訂正を行わない中規模な量子デバイスである．このようなデバイス上で量子計算を行うためには，量子プログラムを表現する論理回路に対して，全ての演算をデバイス上で実行可能な形式に変換しなければならない．この操作は量子回路コンパイル (quantum circuit compilation) と呼ばれている．

コンパイル処理では，NISQ デバイス特有の性質を考慮しなければならない．NISQ デバイスでは量子ビット間の接続関係が制限されており，2 入力ゲートは物理的に隣り合った量子ビット間でしか作用できない．また，NISQ デバイスではエラー訂正を行えないため，量子ゲートの引き起こすエラーは蓄積してしまう．したがって，量子コンパイラは接続関係の制約を満たしながら，できるだけ少ない量子ゲートで回路を合成しなければならない．

本研究では，論理回路は 1 入力ゲートと CNOT ゲートの組み合わせとして表現されていると仮定する．このフォーマットは任意の論理回路を表現することが可能である [1] ため，この仮定は入力可能な論理回路を限定しない．CNOT ゲートは制御ビットと標的ビットを持つ 2 入力の量子ゲートであり，制御ビットの状態に応じて標的ビットを反転する機能を持っている．

量子コンパイラでは, 論理回路中の 1 入力ゲートや CNOT ゲートを，デバイス上の 1 入力ゲートや CNOT ゲートに変換する．デバイス上の CNOT ゲートとしては，隣接した量子ビット間の CNOT ゲート (隣接 CNOT ゲート) と，離れた量子ビット間の CNOT ゲート (遠隔 CNOT ゲート) の 2 種類が考えられるが，遠隔 CNOT ゲートはデバイス上でそのまま実行することはできない．そのため，量子コンパイラは遠隔 CNOT ゲートを複数の隣接 CNOT ゲートに分解し，実行可能な形に変換している．本研究では，隣接 CNOT ゲートの個数をコストして定義する．これは，隣接 CNOT ゲートは 1 入力ゲートよりもエラー率が高く，

[1] 東京大学大学院情報理工学系研究科
Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
[2] 株式会社フィックスターズ
Fixstars Corporation, 3-1-1 Shibaura, Minato-ku, Tokyo 108-0023, Japan
[3] 早稲田大学グリーン・コンピューティング・システム研究機構
Green Computing System Research Organization, Waseda University, 27 Wasedacho, Shinjuku-ku, Tokyo 162-0042, Japan
[4] 慶應義塾大学理工学部物理情報工学科
Department of Applied Physics and Physico-Informatics, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223-8522, Japan

計算の精度を大きく左右するためである．

コンパイル処理において最も難しい問題は，プログラム中の論理量子ビットをデバイス上の物理量子ビットに割り当てる問題であり，これは量子ビットマッピング (qubit mapping) と呼ばれている．マッピング処理では，2 入力ゲートを作用させる量子ビットのペアが物理的に近くなるように割り当てを行う．しかし厄介なことに，回路の各部分で作用する量子ビットのペアは変化するため，固定された割り当てでは最適とはならない．そのため量子ビットマッピングでは，回路を数枚のレイヤーに分割し，それぞれのレイヤーにおける配置を決定する．

マッピング処理を厳密に解く方法としては，整数計画法 [2–4]，SAT [5,6]，SMT [7–9] のようなソルバー向けの手法や，動的計画法 [10] や探索 [11,12] といった手法が提案されている．しかし，この問題は多項式時間で解けないことが証明されているため [10,13]，厳密な解法は非常に小さなプログラムにしか適用できない．中規模デバイス向けの手法としてはヒューリスティックスによる解法が有力であり，非常に多くの手法が提案されている [9,10,14–24]．

近年の研究では，量子コンパイラへの新しいアプローチとして，マッピングにおけるコスト関数を QUBO(Quadratic Unconstrained Binary Optimization) 問題として表現する試みが行われている．QUBO 問題はバイナリ変数による二次最適化問題であり，巡回セールスパーソン問題やグラフ分割問題といった，多くの組合せ最適化問題を表現することができる [25–27]．また，QUBO モデルはイジングマシンや量子アニーリングマシンが扱える形式でもあるため，QUBO 問題に特化したこれらのデバイスを活用することにより，コストを小さくする解を高速にサンプリングできると期待できる．

QUBO 定式化を用いた先行研究としては，Dury らの研究や Butko らの研究がある [28,29]．Dury らは量子ビットの初期配置を決定する問題の QUBO 定式化に取り組んでおり，物理量子ビット間の距離や各ゲートのエラー率に基づいたヒューリスティックな QUBO モデルを提案していた．また，Butko らは量子ゲートを 1 つのタスクとみなし，タスクスケジューリング問題として QUBO 定式化を行った．しかし，Dury らの手法では量子ビットの移動経路が考慮されていないため，ある程度深い回路に対しては改善の効果が見込めない．加えて，彼らの用いた QUBO モデルに対して最適性は保証されていない．また，Butko らの手法は多くのバイナリ変数を必要としており，小さく浅い回路にしか適用できない．

本研究で提案する「ISAAQ(ISing mAchine Assisted Quantum compiler，“アイザック”と読む)」は，以下のような特徴を持つ量子コンパイラである．

- 量子ビットの移動経路の QUBO 定式化．
- コストを最も良く近似する QUBO モデル．

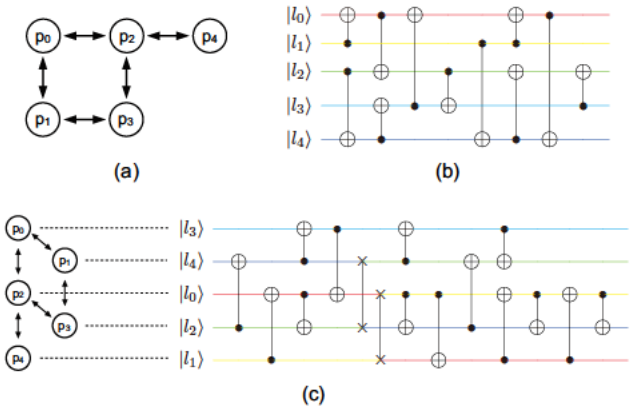

図 1 (a) コンパイル先の量子デバイス．(b) コンパイル前の論理回路．各色は論理量子ビットに対応しており，“o-⊕” の記号は CNOT ゲートを表している．(c) コンパイル結果．“x-x” の記号は SWAP ゲートを表している．この回路は 20 個の隣接 CNOT ゲートを使って構成可能であるため，コストは 20 となる．

- 自身の出力を利用した近似の高精度化．
- 複数イジングマシンを使った並列実行．
- デバイス上の経路を考慮したコスト削減．

ISAAQ は既存手法の弱点を克服できるだけでなく，他の手法には無かった新しい特徴も併せ持つ，大変興味深い手法といえる．Fixstars Amplify [30] を用いた評価実験では，ISAAQ は Dury らの手法および既存のヒューリスティックアルゴリズムよりもコストの小さな回路を出力できており，本手法が有効に機能することが確かめられた．

2. NISQ デバイスに対する量子回路コンパイル

この章では，図 1 に示された例を使いながら，コンパイル処理について説明する．図 1(a) はデバイス形状を表しており，5 つの量子ビット $p_0,\cdots,p_4$ を含んでいる．このデバイスでは，物理的に繋がったペアは $(p_0 \leftrightarrow p_1),(p_0 \leftrightarrow p_2),(p_1 \leftrightarrow p_3),(p_2 \leftrightarrow p_3),(p_2 \leftrightarrow p_4)$ の 5 つであり，隣接 CNOT ゲートはこのペアにのみ作用させることが可能である．

コンパイル処理では，まず回路をレイヤーに分割する処理が行われる．図 1(b) に示される回路は，5 つの CNOT ゲートを含む前半のレイヤーと，6 つの CNOT ゲートを含む後半のレイヤーの 2 つに分割される．次に，各レイヤーに対して量子ビットの割り当てが行われる．この例では，前半のレイヤーに対しては $(l_0,l_1,l_2,l_3,l_4) \rightarrow (p_2,p_4,p_3,p_0,p_1)$ と，後半のレイヤーに対しては $(l_0,l_1,l_2,l_3,l_4) \rightarrow (p_4,p_2,p_1,p_0,p_3)$ と割り当てられている．

このとき，論理回路に含まれていた 11 個の CNOT ゲートは，10 個の隣接 CNOT ゲートと 1 個の遠隔 CNOT ゲートによって実装される．この遠隔 CNOT ゲートは隣接

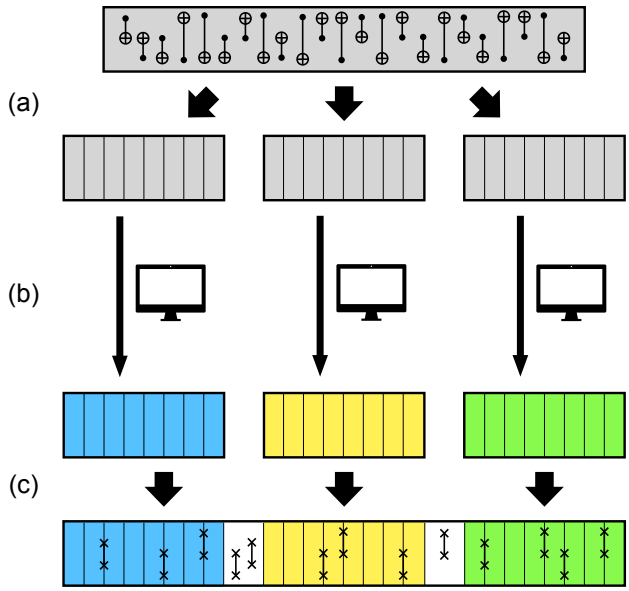

図 2 (a) 巨大な論理回路を複数のチャンクに分割した上で，各チャンクを数枚のレイヤーに分割する．(b) それぞれのチャンクに対して QUBO モデルを生成し，イジングマシンを使って量子ビットマッピングを行う．(c)(b) で得られた割り当てをもとに論理回路中の CNOT ゲートを実装し，レイヤー間を繋ぎ合わせるために SWAP ゲートを挿入する．

CNOT ゲートを 4 つ使うことで実装可能なため，CNOT ゲート実装にかかるコストは 14 となる．また，前半と後半のレイヤーでは量子ビットの配置が異なっているため，それらを繋ぎ合わせるために 2 つの SWAP ゲートがレイヤーの間に挿入される．1 つの SWAP ゲートは 3 つの隣接 CNOT ゲートによって実装可能なため，SWAP ゲート実装にかかるコスト 6 を加えて，回路合成全体のコストは 20 となる．

3. ISAAQ

本稿では，変数および関数を表 1 のように定義し用いている．

3.1 処理の流れ

ISAAQ で行われる処理の流れを図 2 に示す．ISAAQ では，初めに論理回路を複数のチャンク (短く切られた回路) に分割する処理が行われ，その後各チャンクは数枚のレイヤー (薄い回路の断片) に分割される (図 2(a))．図の例では，論理回路は左，中央，右の 3 つのチャンクに分割され，それぞれのチャンクは 8 枚のレイヤーに分割されている．ISAAQ では，それぞれのレイヤーが高々 20 個の CNOT ゲートを含むように，それぞれのチャンクがイジングマシンで扱える上限を超えないように分割が実行される．これにより，ISAAQ は任意の深さの回路をコンパイルすることが可能となっている．

図 2(b) に示す量子ビットマッピングでは，ISAAQ はそれぞれのチャンクに対して QUBO モデルを生成し，イジングマシンを使って解を探索する．レイヤーにおいて量子ビットの配置が決定した後は，論理回路中の CNOT ゲートの実装と，レイヤー間への SWAP ゲートの挿入が行われる (図 2(c))．これにより，デバイスで実行可能な回路へのコンパイルが実現される．

3.2 QUBO 定式化

この節では，量子ビットマッピングにおけるコスト関数を，以下の式で表される QUBO モデルとして定式化する手法について説明する．

$$\begin{aligned} &\text{minimize} \quad \sum_i \sum_j J_{i,j} x_i x_j + \sum_i h_i x_i, \\ &\text{subject to} \quad x_i \in \{0,1\}. \end{aligned} \tag{1}$$

3.2.1 変数

QUBO 定式化において，変数は各レイヤーにおける量子ビットの配置である．m 番目のレイヤーにおける割り当て f^m および論理量子ビットの行き先 $\boldsymbol{p^m}$ は

$$\begin{aligned} &f^m : \{l_0, \cdots, l_{N-1}\} \mapsto \{p_0, \cdots, p_{N-1}\} \\ &\boldsymbol{p^m} = \begin{pmatrix} p_0^m & \cdots & p_{N-1}^m \end{pmatrix} \coloneqq \begin{pmatrix} f^m(l_0) & \cdots & f^m(l_{N-1}) \end{pmatrix} \end{aligned} \tag{2}$$

のように表現できる (ここで N は量子ビットの個数である)．QUBO 定式化するにあたり，ISAAQ では $\boldsymbol{p^m}$ を以下で定義されるバイナリ変数 $x_{i,\mu}^m$ を使って表現する．

$$x_{i,\mu}^m \coloneqq \mathbb{1}\left(p_i^m = p_\mu\right). \tag{3}$$

このとき，$x_{i,\mu}^m$ は「m 番目のレイヤーにおいて，l_i が p_μ に割り当てられているかどうか」という意味を持っている．

ISAAQ で用いる QUBO モデルは 1 枚のレイヤーにつき N^2 個のバイナリ変数を用いている．しかし，2^{N^2} 通りの状態空間は $\boldsymbol{p^m}$ を表現するには広すぎるため，全単射写像に制限するように，以下の式 (4) で示される (one-hot 制約と呼ばれる) 制約を l_i, p_μ ごとに課す必要がある．

$$\sum_{i=0}^{N-1} x_{i,\mu}^m = \sum_{\mu=0}^{N-1} x_{i,\mu}^m = 1. \tag{4}$$

QUBO モデルでは制約条件をそのまま扱うことができないため，以下の式 (5) に示すペナルティ項として表現し，コスト関数に足し合わせることで埋め込みが可能になる．ISAAQ では，制約の重みパラメータ λ は有効な解をサンプリングするのに十分大きな値となるように調整されている．

$$\text{penalty} = \lambda \left(\sum_{i=0}^{N-1} x_{i,\mu}^m - 1 \right)^2 + \lambda \left(\sum_{\mu=0}^{N-1} x_{i,\mu}^m - 1 \right)^2. \tag{5}$$

表 1 変数，関数の表記およびその定義

表記	定義
N	量子ビットの個数.
l_i	論理量子ビット．$(i \in \{0,\cdots,N-1\})$
p_μ	物理量子ビット．$(\mu \in \{0,\cdots,N-1\})$
$d(p_\mu, p_\nu)$	物理量子ビット間の距離.
M	レイヤーの枚数．$(m \in \{0,\cdots,M-1\})$
CNOT^m	レイヤーに含まれる CNOT ゲートの集合．$((l_a,l_b) \in \mathrm{CNOT}^m)$
$\mathrm{CNOT}(l_a \to l_b)$	l_a から l_b に作用する CNOT ゲート.
$f^m(\cdot)$	論理量子ビットを物理量子ビットに割り当てる関数．$f^m(l_i)$ は l_i の行き先を表している.
$\boldsymbol{p^m} = (p_0^m,\cdots,p_{N-1}^m)$	論理量子ビット $(l_0\cdots l_{N-1})$ それぞれの行き先.
$x_{i,\mu}^m$	f^m によって l_i が p_μ に割り当てられているかどうか.
$y_{\mu,\nu}^m$	m 番目のレイヤーで p_μ にあった論理量子ビットが，$(m+1)$ 番目のレイヤーで p_ν にあるかどうか.
$\boldsymbol{\sigma^m} = \begin{pmatrix} 0 & \cdots & N-1 \\ \sigma_0^m & \cdots & \sigma_{N-1}^m \end{pmatrix}$	$\boldsymbol{p^m}$ から $\boldsymbol{p^{m+1}}$ への置換．もし $\sigma_\mu^m = \nu$ なら $y_{\mu,\nu}^m = 1$ が従う.
$\mathrm{N_s}(\boldsymbol{\sigma^m})$	置換 $\boldsymbol{\sigma^m}$ を実装するのに必要となる SWAP ゲートの最小個数.
λ	QUBO モデルで使われる制約の重みパラメータ.

3.2.2 コスト関数

量子ビットマッピングにおけるコスト関数，すなわち隣接 CNOT ゲートの個数は，論理回路中の CNOT ゲートの実装に費やされた個数と，レイヤー間に挿入される SWAP ゲートの費やされた個数の和となる．これは，

$$\mathrm{COST_{total}} = \mathrm{COST_{cnot}} + \mathrm{COST_{swap}} \tag{6}$$

と形式的に書くことができる．

$\mathrm{COST_{cnot}}$ に対しては，ISAAQ は CNOT ゲート一つ一つのコストの和としてこれを定式化する．つまり，

$$\begin{aligned}\mathrm{COST_{cnot}} &= \sum_{m=0}^{M-1} \sum_{\substack{(l_a,l_b)\in \\ \mathrm{CNOT}^m}} c\left(f^m(l_a), f^m(l_b)\right) \\ &= \sum_{m=0}^{M-1} \sum_{\substack{(l_a,l_b)\in \\ \mathrm{CNOT}^m}} c\left(p_a^m, p_b^m\right)\end{aligned} \tag{7}$$

と定義する．ここで，$c(\cdot,\cdot)$ とは CNOT ゲート単体のコストを表す関数であり，実装に必要な隣接 CNOT ゲートの個数に等しい．このコストはデバイス上の距離 $d(p_a^m, p_b^m)$ に依存しており，以下のように変化する．

$$c\left(p_a^m, p_b^m\right) = \max\left(1, 4(d(p_a^m, p_b^m) - 1)\right). \tag{8}$$

これは，距離が 1 の場合は隣接 CNOT ゲート 1 つで実装できるが，2 以上離れている場合は図 3 のように組み合わせなければいけないからである．

$c(p_\mu, p_\nu)$ は，デバイス形状だけから計算可能な値である．そのため，QUBO モデルの作成において $c(p_\mu, p_\nu)$ は定数として扱うことができる．これを用いると，$c\left(p_a^m, p_b^m\right)$ は次のように $x_{i,\mu}^m$ の二次式として書き表すことができる．

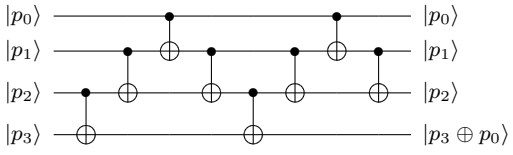

図 3 遠隔 CNOT ゲートの実装．この例では，量子ビットのペアは 3 離れている．

$$\begin{aligned}c\left(p_a^m, p_b^m\right) &= \sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} c(p_\mu,p_\nu)\mathbb{1}\left(p_a^m = p_\mu\right)\mathbb{1}\left(p_b^m = p_\nu\right) \\ &= \sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} c(p_\mu,p_\nu) x_{a,\mu}^m x_{b,\nu}^m.\end{aligned} \tag{9}$$

したがって，$\mathrm{COST_{cnot}}$ は

$$\mathrm{COST_{cnot}} = \sum_{m=0}^{M-1} \sum_{\substack{(l_a,l_b)\in \\ \mathrm{CNOT}^m}} \sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} c(p_\mu,p_\nu) x_{a,\mu}^m x_{b,\nu}^m \tag{10}$$

と定式化できる．

次に，$\mathrm{COST_{swap}}$ の QUBO 定式化を行う．これは SWAP ゲートの個数の 3 倍に一致し，数式では以下のように表現できる．

$$\mathrm{COST_{swap}} = 3\sum_{m=0}^{M-2} \mathrm{N_s}(\boldsymbol{\sigma^m}). \tag{11}$$

ここで $\boldsymbol{\sigma^m}$ は以下のように定義される置換である．

$$\boldsymbol{\sigma^m} = \begin{pmatrix} 0 & \cdots & N-1 \\ \sigma_0^m & \cdots & \sigma_{N-1}^m \end{pmatrix} := \begin{pmatrix} p_0^m & \cdots & p_{N-1}^m \\ p_0^{m+1} & \cdots & p_{N-1}^{m+1} \end{pmatrix}. \tag{12}$$

$\mathrm{N_s}(\boldsymbol{\sigma^m})$ は，並び替え $\boldsymbol{\sigma^m}$ を構成するために必要な SWAP ゲートの最小個数である．残念なことに，$\mathrm{COST_{swap}}$ の厳密な QUBO 定式化は $\mathrm{COST_{cnot}}$ とは違って非常に難しい．その理由は，$\mathrm{N_s}(\boldsymbol{\sigma^m})$ を求める問題が NP 困難であると証

明されているからである [13].

この問題に対処するため, ISAAQ では $\mathrm{N_s}(\boldsymbol{\sigma^m})$ を式 (13) のように近似して扱っている.

$$\mathrm{N_s}(\boldsymbol{\sigma^m}) \approx \sum_{\mu=0}^{N-1} a_{\mu,\sigma_\mu^m} = \sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} a_{\mu,\nu} y_{\mu,\nu}^m. \tag{13}$$

ここで $a_{\mu,\nu}$ は定数であり，$y_{\mu,\nu}^m$ は「p_μ にあった論理量子ビットが p_ν に移ったかどうか」を表す補助変数である．このとき，$y_{\mu,\nu}^m$ は

$$y_{\mu,\nu}^m = \mathbb{1}\left(\sigma_\mu^m = \nu\right) = \sum_{i=0}^{N-1} x_{i,\mu}^m x_{i,\nu}^{m+1} \tag{14}$$

と書け，これを式 (13) に代入することで以下の関係式を満たす.

$$\mathrm{N_s}(\boldsymbol{\sigma^m}) \approx \sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} a_{\mu,\nu}\left(\sum_{i=0}^{N-1} x_{i,\mu}^m x_{i,\nu}^{m+1}\right). \tag{15}$$

よって，式 (15) を代入することにより

$$\mathrm{COST_{swap}} \approx 3\sum_{m=0}^{M-2}\sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} a_{\mu,\nu}\left(\sum_{i=0}^{N-1} x_{i,\mu}^m x_{i,\nu}^{m+1}\right) \tag{16}$$

と，近似された $\mathrm{COST_{swap}}$ を表現することが可能となる．これによって，コスト関数全体を QUBO モデルとして表現することができた.

3.2.3 SWAP コストを近似する QUBO 係数の決定

式 (13) で用いられている係数 $a_{\mu,\nu}$ は，近似における誤差を最小とするように，QUBO モデルを作成する前に決定されなければならない．ISAAQ では，近似における誤差として以下を用いている.

$$E \coloneqq \frac{1}{2}\sum_{\left(\boldsymbol{\sigma},\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\right)\in D}\left(\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma}) - \sum_{\mu=0}^{N-1} a_{\mu,\sigma_\mu}\right)^2. \tag{17}$$

ここで，D は layer 間に現れた置換とその際用いられた SWAP ゲート数のデータセットであり，これは ISAAQ を実行することで蓄積することができる.

E は凸関数であるため，$a_{\mu,\nu}$ の最適なパラメータを決定することは，以下に示す極小性の条件を全ての μ,ν に対して満たすことと同値となる.

$$\frac{\partial E}{\partial a_{\mu,\nu}} = \sum_{\substack{\left(\boldsymbol{\sigma},\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\right)\in D \\ \sigma_\mu=\nu}}\left(\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma}) - \sum_{\mu'=0}^{N-1} a_{\mu',\sigma_{\mu'}}\right) = 0. \tag{18}$$

この方程式は

$$\sum_{\substack{\left(\boldsymbol{\sigma},\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\right)\in D \\ \sigma_\mu=\nu}} \hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma}) = \sum_{\mu'=0}^{N-1}\sum_{\nu'=0}^{N-1} a_{\mu',\nu'} \sum_{\substack{\left(\boldsymbol{\sigma},\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\right)\in D \\ \sigma_\mu=\nu\wedge\sigma_{\mu'}=\nu'}} 1 \tag{19}$$

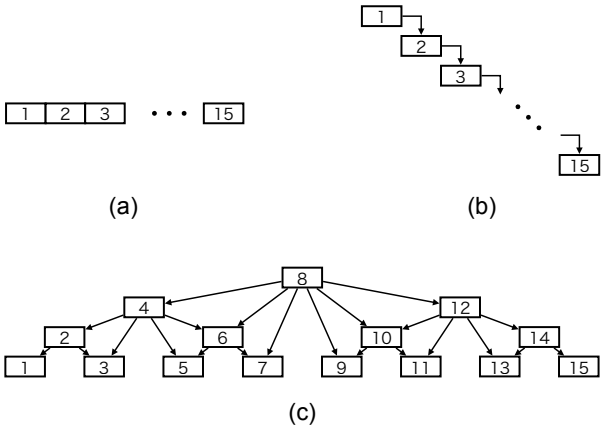

図 4 (a) チャンクが独立な場合．(b) 前のチャンクに依存している場合．(c) 二分木の親ノードに依存している場合.
(a)(b)(c) 全てにおいて，矢印はチャンク間の依存関係を示している.

と線形方程式として表現することができるため，行列演算により最適な $a_{\mu,\nu}$ を求めることができる．実際には，$a_{\mu,\nu}$ の解空間は自由度を持っており一意には定まらないため，ISAAQ では二乗ノルム $|\boldsymbol{a}|^2 = \sum_{\mu=0}^{N-1}\sum_{\nu=0}^{N-1} a_{\mu,\nu}^2$ が最小となるような解を $a_{\mu,\nu}$ に採用している.

ISAAQ を最初に実行する場合は，D が一様分布であると仮定することで $a_{\mu,\nu}$ を次のように決定できる.

$$a_{\mu,\nu} = \frac{N-1}{N}\langle\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\rangle_{\sigma_\mu=\nu} - \frac{N-2}{N}\langle\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\rangle. \tag{20}$$

ここで，$\langle\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})\rangle_{\sigma_\mu=\nu}$ は $\sigma_\mu=\nu$ となる場合の $\hat{\mathrm{N}}_\mathrm{s}(\boldsymbol{\sigma})$ の期待値である．なお，紙面の都合上，詳細な導出過程は省略する.

3.3 チャンク間の依存関係の工夫

ISAAQ において，チャンク間の依存関係は SWAP ゲートの削減や複数イジングマシンを使った並列処理の可否を左右し，ISAAQ の性能に大きく影響する．チャンク間の依存関係として最も単純な例は，チャンクが独立している (すなわち依存関係を持たない) ケースである (図 4(a))．この場合，QUBO モデルはチャンク内のコストだけを表すように生成されるため，チャンク間の SWAP コストは考慮されない．したがって，チャンク間の置換はほとんどランダムになり，これの解消に多数の SWAP ゲートを必要としてしまう．一方で，それぞれのチャンクにおけるマッピング結果は他の QUBO モデルに影響を与えないため，複数のイジングマシンによる並列実行が可能となる．その他の代表的な例としては，QUBO モデルが前のチャンクとの間のコストを考慮して生成されるケースである (図 4(b))．このとき，全てのチャンクは一つ前のチャンクに依存しているため並列実行はできないが，イジングマシンはチャンク間の SWAP ゲートも考慮しながら QUBO 問題を解くた

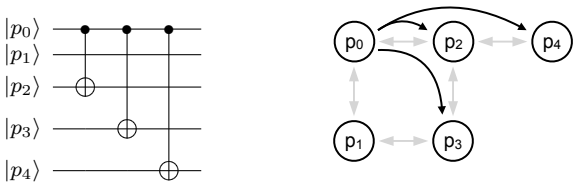

図 5 $(p_0 \leftrightarrow p_3)$ と $(p_0 \leftrightarrow p_4)$ のペアは隣接していないため，1 つずつ実装した場合 9 個の隣接 CNOT ゲートが必要となる．

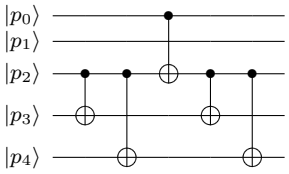

図 6 p_2 を中継地点として利用することで，4 個の隣接 CNOT ゲートを削減することができる．

め，チャンク間の SWAP コストは低減される．

これらのケースは，SWAP コスト削減と並列実行のどちらか一方は達成できるが，両方を達成することはできない．そこで，ISAAQ では図 4(c) のようにチャンクを二分木上に配置し，各チャンクが親ノードにのみ依存するような処理順を採用することとした．二分木において，それぞれのチャンクの高さは番号のバイナリ表現に基づいて決定され，奇数なら高さ 0，2 の倍数なら高さ 1，4 の倍数なら高さ 2，$\cdots$ となる．このとき，それぞれのチャンクで生成される QUBO モデルはより上位のチャンクとの SWAP コストを含めるように生成され，同じ高さのチャンクとは依存関係を持たない．そのため，この手法は SWAP コスト低減と並列実行を同時に実現することができている．

3.4 回路合成

各レイヤーにおける量子ビットの割り当てが決まったら，ISAAQ はこれをもとにして，できるだけ少ない隣接 CNOT ゲートを使って回路を合成する．

3.4.1 デバイス上の経路を考慮したコスト削減

QUBO 定式化で用いられていた $\mathrm{COST}_{\mathrm{cnot}}$ は，レイヤー内で使われる隣接 CNOT ゲートの個数を表していた．このコストは，それぞれの CNOT ゲートを 1 つずつ実装した場合のコストに等しいが，最適ではない．なぜなら，複数の CNOT ゲートが同じ量子ビットを共有している場合，より少ない隣接 CNOT ゲートで実装できる場合があるからである．

ISAAQ は，図 5 のように複数の CNOT ゲートが同じ量子ビットを共有しているケースに対し「中継地点」を使ったコスト低減手法を適用する．直感的には，これはデバイス上で重複した経路を削減する手法である．図 6 の例では，p_2 を中継地点とすることで $p_0 \to p_2$ の経路が削減され，隣接 CNOT ゲートを 9 個から 5 個にまで減らすことができている．

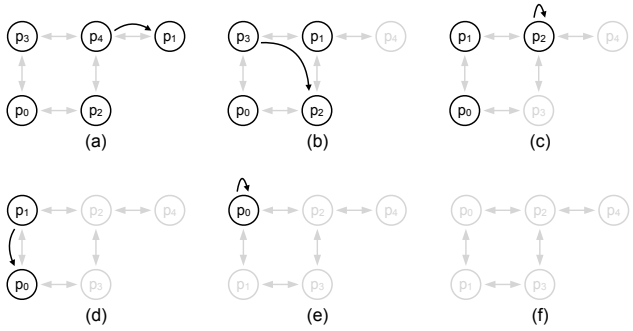

図 7 $p_4, \cdots, p_0$ の順番で量子ビットを移動させるアルゴリズム．この例では 4 個の SWAP ゲートで実現でき，最小となっている．(b) では 2 通りの経路が考えられるが，下を通る経路は p_2 を目的地から遠ざけてしまうため採用されない．

ISAAQ はこのような CNOT ゲートの集合に対し，中継地点を使うか否か，使うとしたらどこを中継するかを決定する．中継地点を使わない場合，ISAAQ は

$$\sum_{p_t \in T} c(p_c, p_t) = \sum_{p_t \in T} \min(1, 4(d(p_c, p_t) - 1)) \qquad (21)$$

個の隣接 CNOT ゲートを用いる (ここで T は標的ビットの集合である)．一方で，中継地点を使う場合，隣接 CNOT ゲートの個数は以下のように表現できる．

$$\begin{aligned} &c(p_c, p_h) + \sum_{p_t \in T} \min(2c(p_h, p_t), c(p_c, p_t)) \quad (\text{if } p_h \in T) \\ &2c(p_c, p_h) + \sum_{p_t \in T} \min(2c(p_h, p_t), c(p_c, p_t)) \quad (\text{otherwise}). \end{aligned} \qquad (22)$$

3.4.2 レイヤー間への SWAP ゲート挿入

置換 $\boldsymbol{\sigma^m}$ を実現するために，ISAAQ ではグラフ上のソートアルゴリズムを用いている．ソートする関数としては，ISAAQ は厳密解法とヒューリスティック解法を組み合わせている．厳密解法では，ISAAQ は事前に求めた答えをメモリに格納しておき，$\boldsymbol{\sigma^m}$ に対応する答えを返す．しかし，この解法は $O(N!)$ の計算量を要してしまうため，N が大きい場合は答えが小さい場合にのみ適用される．

ヒューリスティック解法では，ISAAQ は量子ビットを 1 つずつ動かしては固定していく．図 7 はその例であり，ここでは $p_4, \cdots, p_0$ の順番で量子ビットを動かしている．ルートを決定する際は，ISAAQ はそれぞれ量子ビットに対して目的地への距離を考慮し，できるだけ多くの量子ビットを目的地に近づけるルートを選択する．図 7(b) では 2 通りの経路が考えられるが，$p_3 \to p_1 \to p_2$ は $p_3 \to p_0 \to p_2$ より好ましいと判断される．なぜなら，このルートは p_1, p_2 を目的地に近づけるが，もう一方のルートは p_2 を目的地から遠ざけてしまうからである．このアルゴリズムは幅優先探索を用いて実装されており，$O(N)$ で最適なルートを

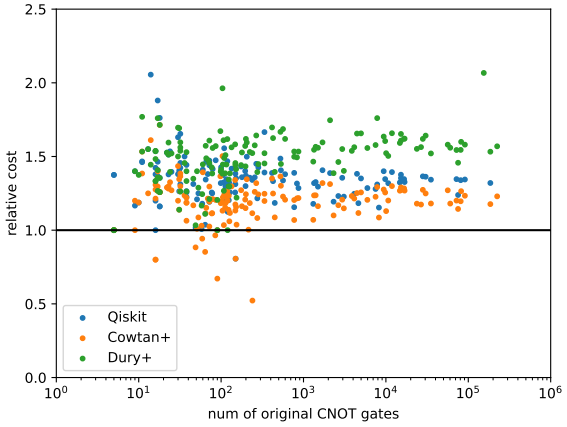

図 8 IBM QX5 に対するコンパイル結果．それぞれの点は ISAAQ のコストに対する倍率を表している．黒線より上にある点は ISAAQ よりもコストが大きかったことを意味している．

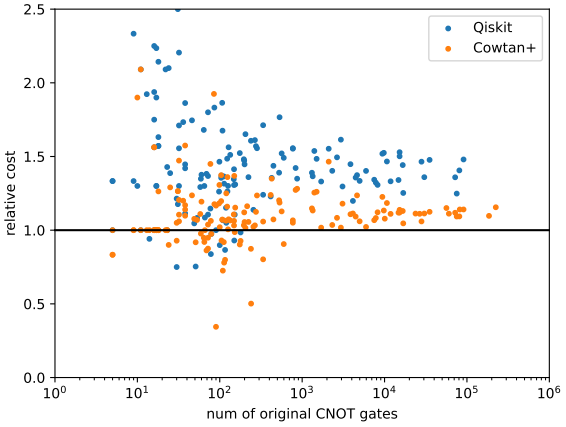

図 9 IBM QX20 に対するコンパイル結果．

探索できる．したがって，このアルゴリズムは $O(N^2)$ の計算量で動作する．ISAAQ では，動かす量子ビットの順番をランダムに 100 通り生成し，その中で最も良かった解を答えとして出力している．

4. 評価実験

4.1 実験環境

本研究で提案する ISAAQ は，Fixstars Amplify [30] を使って実装されている．Fixstars Amplify はイジングマシンの統合開発環境であり，Python を用いて様々なイジングマシンを共通のインターフェースで使用できるという特徴がある．ISAAQ のバックエンドで動作するイジングマシンとしては Fixstars Amplify Annealing Engine が用いられており，これは GPU を使って実装されている．ISAAQ では最適解に近い解を高速にサンプリングできるように，QUBO モデルに含まれるバイナリ変数を 1200 個以下に抑え，各 QUBO モデルに対し探索処理を 1000ms ずつ実行した．

4.2 他の量子コンパイラとの比較

ISAAQ と他の量子コンパイラを比較するにあたり，本研究では過去の研究 [15, 24, 28] で用いられている回路のデータセットが用いられている．このデータセットは 158 個の回路サンプルからなり，量子ビットは 3 個から 16 個，CNOT ゲートは 5 個から 224028 個と，幅広く分布している．また，コンパイル先のデバイスとしては IBM QX5(IBM Melbourne) と IBM QX20(IBM Tokyo) を用いており，それぞれ 16 個，20 個の量子ビットを含んでいる．

ISAAQ の比較相手としては，Li らの手法 (SABRE) [22]，Cowtan らの手法 [15]，Dury らの手法 [28] の 3 つが選ばれている．Li ら，および Cowtan らの手法はヒューリスティックなアルゴリズムであり，それぞれ Qiskit [31] と TKET [32] という量子プログラミング環境で用いられている手法である．また，Dury らの手法は QUBO を使って初期配置を決定する手法であり，ISAAQ と似たアプローチだと言えよう．この実験では，SABRE によるコストは Qiskit 上でコンパイルを実行することによって取得し，その他の手法によるコストは彼らの論文中で示されている値を参照した．なお，Dury らの手法ではコンパイルが IBM QX5 にのみ行われていたため，IBM QX20 に対するデータは無かった．

表 2 論理回路中の CNOT ゲート 1 つに対する平均コスト．ISAAQ は 4 つの手法のうち最も小さな平均コストを達成している．

	Li+	Cowtan+	Dury+	ISAAQ
IBM QX5	2.8217	2.5888	3.2388	**2.0366**
IBM QX20	2.1145	1.7091	-	**1.4832**

図 8 および図 9 は，それぞれ IBM QX5，IBM QX20 に対する結果である．これらの図においては，ISAAQ との大小関係を明確にするため，ISAAQ のコストに対する倍率をプロットしている．これは，他手法によるコストを ISAAQ のコストで割った値として定義される．

図 8 では，ほとんど全ての点が黒線 (ISAAQ と等倍のライン) より上にプロットされていた．特に CNOT ゲートが 1000 個を超えるような論理回路に対しては，全てのサンプルにおいて ISAAQ が既存手法に勝つ結果となり，ISAAQ が大きな回路に対し特に有効であることが分かった．図 9 においても，その差は小さいものの，大きい回路に対して ISAAQ は依然として既存手法を上回る性能を発揮していることが分かった．

表 2 は，論理回路中の CNOT ゲート 1 つに対する平均コストを示している．ISAAQ は既存手法のうち最も優れていた Cowtan らの手法と比べても，IBM QX5 では平均 21.3%, IBM QX20 では平均 13.2%のコスト削減に成功しており，極めて性能の高い量子コンパイラだと言える．

5. まとめ

本研究では，イジングマシンを活用した量子コンパイラ「ISAAQ」を提案した．ISAAQ は量子ビットマッピングの QUBO 定式化だけでなく，自身の出力を使った高精度化，複数イジングマシンを使った並列実行，回路合成におけるコスト削減など，他の手法には無い特徴を多く備えている手法となっている．Fixstars Amplify を用いた評価実験では，ISAAQ は既存の QUBO 解法やヒューリスティックアルゴリズムよりも低コストな回路を出力できており，ISAAQ が量子コンパイラとして極めて優れていることを確認した．

謝辞 この研究は，一部，総合科学技術・イノベーション会議 SIP(戦略的イノベーション創造プログラム)「光・量子を 活用した Society5.0 実現化技術」，JSPS 科研費 19H01553，21K03391 の支援を受けて実施された．また，本研究を進めるにあたり，株式会社フィックスターズから開発サポートおよびイジングマシンの提供を受けた．

参考文献

[1] Nielsen, M. A. and Chuang, I. L.: *Quantum Computation and Quantum Information: 10th Anniversary Edition*, Cambridge University Press (2010).
[2] Shafaei, A., Saeedi, M. and Pedram, M.: Qubit placement to minimize communication overhead in 2D quantum architectures, *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, pp. 495–500 (2014).
[3] De Almeida, A. A., Dueck, G. W. and Da Silva, A. C.: Finding optimal qubit permutations for IBM's quantum computer architectures, *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, pp. 1–6 (2019).
[4] Nannicini, G., Bishop, L. S., Gunluk, O. and Jurcevic, P.: Optimal qubit assignment and routing via integer programming, *arXiv preprint arXiv:2106.06446* (2021).
[5] Lye, A., Wille, R. and Drechsler, R.: Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits, *The 20th Asia and South Pacific Design Automation Conference*, IEEE, pp. 178–183 (2015).
[6] Wille, R., Lye, A. and Drechsler, R.: Optimal SWAP gate insertion for nearest neighbor quantum circuits, *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, pp. 489–494 (2014).
[7] Tan, B. and Cong, J.: Optimal Qubit Mapping with Simultaneous Gate Absorption, *arXiv preprint arXiv:2109.06445* (2021).
[8] Wille, R., Burgholzer, L. and Zulehner, A.: Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations, *2019 56th ACM/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6 (2019).
[9] Murali, P., Baker, J. M., Javadi-Abhari, A., Chong, F. T. and Martonosi, M.: Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1015–1029 (2019).
[10] Siraichi, M. Y., Santos, V. F. d., Collange, S. and Pereira, F. M. Q.: Qubit allocation, *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pp. 113–125 (2018).
[11] Zhu, P., Cheng, X. and Guan, Z.: An exact qubit allocation approach for NISQ architectures, *Quantum Information Processing*, Vol. 19, No. 11, pp. 1–21 (2020).
[12] Burgholzer, L., Schneider, S. and Wille, R.: Limiting the Search Space in Optimal Quantum Circuit Mapping (2021).
[13] Aichholzer, O., Demaine, E. D., Korman, M., Lynch, J., Lubiw, A., Mas, Z., Rudoy, M., Williams, V. V. and Wein, N.: Hardness of Token Swapping on Trees, *arXiv preprint arXiv:2103.06707* (2021).
[14] Zhu, P., Guan, Z. and Cheng, X.: A dynamic look-ahead heuristic for the qubit mapping problem of nisq computers, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 39, No. 12, pp. 4721–4735 (2020).
[15] Cowtan, A., Dilkes, S., Duncan, R., Krajenbrink, A., Simmons, W. and Sivarajah, S.: On the qubit routing problem, *arXiv preprint arXiv:1902.08091* (2019).
[16] Itoko, T., Raymond, R., Imamichi, T. and Matsuo, A.: Optimization of quantum circuit mapping using gate transformation and commutation, *Integration*, Vol. 70, pp. 43–50 (2020).
[17] Li, Z.-T., Meng, F.-X., Zhang, Z.-C. and Yu, X.-T.: Qubits' mapping and routing for NISQ on variability of quantum gates, *Quantum Information Processing*, Vol. 19, No. 10, pp. 1–25 (2020).
[18] Nishio, S., Pan, Y., Satoh, T., Amano, H. and Meter, R. V.: Extracting success from ibm's 20-qubit machines using error-aware compilation, *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, Vol. 16, No. 3, pp. 1–25 (2020).
[19] Zhu, P., Feng, S. and Guan, Z.: An Iterated Local Search Methodology for the Qubit Mapping Problem, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021).
[20] Ren, S., Chen, K., Ghadermarzy, N., Nguyen, B., Huang, Y. and Ronagh, P.: Nuwa: A Quantum Circuit Transpiler Based on a Finite-Horizon Heuristic for Placement and Routing, *arXiv preprint arXiv:2110.00592* (2021).
[21] Li, S., Zhou, X. and Feng, Y.: Qubit mapping based on subgraph isomorphism and filtered depth-limited search, *IEEE Transactions on Computers* (2020).
[22] Li, G., Ding, Y. and Xie, Y.: Tackling the qubit mapping problem for NISQ-era quantum devices, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1001–1014 (2019).
[23] Wille, R., Keszocze, O., Walter, M., Rohrs, P., Chattopadhyay, A. and Drechsler, R.: Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits, *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*, IEEE, pp. 292–297 (2016).
[24] Zulehner, A., Paler, A. and Wille, R.: An efficient methodology for mapping quantum circuits to the IBM QX architectures, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 38, No. 7, pp. 1226–1236 (2018).
[25] Tanahashi, K., Takayanagi, S., Motohashi, T. and

Tanaka, S.: Application of Ising Machines and a Software Development for Ising Machines, *Journal of the Physical Society of Japan*, Vol. 88, No. 6, p. 061010 (online), DOI: 10.7566/JPSJ.88.061010 (2019).

[26] Tanana, S., Tamura, R. and Chakrabarti, B. K.: *Quantum Spin Glasses, Annealing and Computation*, Cambridge University Press (2017).

[27] Lucas, A.: Ising formulations of many NP problems, *Frontiers in physics*, Vol. 2, p. 5 (2014).

[28] Dury, B. and Di Matteo, O.: A QUBO Formulation for Qubit Allocation, *arXiv preprint arXiv:2009.00140* (2020).

[29] Butko, A., Turimbetov, I., Michelogiannakis, G., Donofrio, D., Unat, D. and Shalf, J.: TIGER: Topology-aware Assignment using Ising machines Application to Classical Algorithm Tasks and Quantum Circuit Gates, *arXiv preprint arXiv:2009.10151* (2020).

[30] Fixstars: Fixstars Amplify, `https://amplify.fixstars.com/`.

[31] IBM: Qiskit, `https://qiskit.org/`.

[32] Cambridge Quantum: TKET, `https://cambridgequantum.com/`.