Quantum Circuit Transformation: A Monte Carlo Tree Search Framework

XIANGZHEN ZHOU, Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia and State Key Lab of Millimeter Waves, Southeast University, China
YUAN FENG, Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia
SANJIANG LI, Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia

In Noisy Intermediate-Scale Quantum (NISQ) era, quantum processing units (QPUs) suffer from, among others, highly limited connectivity between physical qubits. To make a quantum circuit *effectively* executable, a circuit transformation process is necessary to transform it, with overhead cost the smaller the better, into a functionally equivalent one so that the connectivity constraints imposed by the QPU are satisfied. While several algorithms have been proposed for this goal, the overhead costs are often very high, which degenerates the fidelity of the obtained circuits sharply. One major reason for this lies in that, due to the high branching factor and vast search space, almost all these algorithms only search very *shallowly* and thus, very often, only (at most) locally optimal solutions can be reached. In this paper, we propose a Monte Carlo Tree Search (MCTS) framework to tackle the circuit transformation problem, which enables the search process to go much deeper. The general framework supports implementations aiming to reduce either the size or depth of the output circuit through introducing SWAP or remote CNOT gates. The algorithms, called MCTS-Size and MCTS-Depth, are polynomial in all relevant parameters. Empirical results on extensive realistic circuits and IBM Q Tokyo show that the MCTS-based algorithms can reduce the size (depth, resp.) overhead by, on average, 66% (84%, resp.) when compared with t|ket⟩, an industrial level compiler.

CCS Concepts: • **Hardware → Quantum computation**;

Additional Key Words and Phrases: Quantum computing, qubit mapping, quantum circuit transformation, QPU, Monte Carlo Tree Search

ACM Reference Format:
Xiangzhen Zhou, Yuan Feng, and Sanjiang Li. 2022. Quantum Circuit Transformation: A Monte Carlo Tree Search Framework. 1, 1 (March 2022), 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With Google's recent conspicuous, though arguable, success in demonstrating quantum supremacy in a 53-qubit quantum processor [Arute et al. 2019], NISQ (Noisy Intermediate-Scale Quantum) devices have attracted rapidly increasing interests from researchers in both academic and industrial

Authors' addresses: Xiangzhen Zhou, Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia and State Key Lab of Millimeter Waves, Southeast University, China; Yuan Feng, Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia, yuan.feng@uts.edu.au; Sanjiang Li, Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia, sanjiang.li@uts.edu.au.

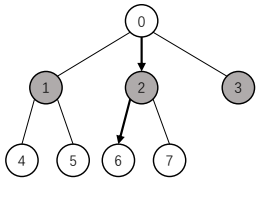
XXXX-XXXX/2022/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

communities. *Quantum processing units* (QPUs) in the NISQ era only support a limited set of basic operations (elementary quantum gates) and often suffer from high gate errors, short coherence time, and limited connectivity between physical qubits. In order to run a quantum algorithm, described as a quantum circuit, we need to *compile* the circuit (referred to as *logical circuit* henceforth) into a functionally equivalent *physical circuit* executable on the QPU. The compilation includes two basic processes. In the *decomposition* process, gates in the logical circuit are decomposed, or transformed, into elementary gates supported by the QPU [et al. 2019; Häner et al. 2018; Sivarajah et al. 2020]. The *transformation* process, initiated in [Cheung et al. 2007; Maslov et al. 2007] and also known as *quantum circuit transformation* (QCT) [Childs et al. 2019] or *qubit mapping* [Li et al. 2019], is then performed on the generated circuit, which further consists of two steps: *initial mapping construction* and *qubit routing*. The former process constructs a mapping that maps qubits in a logical circuit, called *logical qubits*, to the ones in the QPU, called *physical qubits*; while the latter transforms a circuit through adding ancillary operations like SWAP gates to 'route' physical qubits in order to make all multi-qubits gates executable.

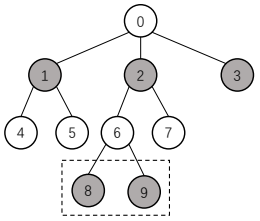
Both the decomposition and the transformation processes have been studied extensively in the literature. As there are now standard decomposition processes (see, e.g., [Nielsen and Chuang 2002, Chapter 4]), in this paper, we focus on the transformation process, and assume that gates in the input logical circuit have been well decomposed into elementary gates that are supported by the QPU. Furthermore, we assume that an initial mapping is given, which can be obtained by employing, say, the greedy strategy [Cowtan et al. 2019; Paler 2019; Zulehner et al. 2018], the reverse traversal technique [Li et al. 2019], the simulated annealing based algorithm [Zhou et al. 2020b], or the subgraph isomorphism based methods [Li et al. 2021; Maslov et al. 2007; Siraichi et al. 2019b].

To reduce the gate overheads in the qubit routing step, many algorithms have been proposed aiming at minimising gate counts [Li et al. 2021; Lye et al. 2015; Zhou et al. 2020b; Zulehner et al. 2018], circuit depths [Booth et al. 2018; Lao et al. 2019; Venturelli et al. 2017; Zhang et al. 2020] or circuit error [Murali et al. 2019; Nishio et al. 2020]. These algorithms can be roughly classified into two broad categories (see also [Kusyk et al. 2021] for a similar classification). The first category consists of algorithms that try to reformulate QCT as a planning or optimisation problem and solve it by applying off-the-shelf tools [Booth et al. 2018; de Almeida et al. 2019; Murali et al. 2019; Rasconi and Oddi 2019; Saeedi et al. 2011; Siraichi et al. 2018; Venturelli et al. 2018, 2017; Wille et al. 2019; Zhang et al. 2021; Zhu et al. 2020]. However, as shown in [Childs et al. 2019; Siraichi et al. 2018], QCT is NP-complete in general. Algorithms in this category are usually highly unscalable when the size of input circuits becomes large.

In contrast, algorithms in the second category use heuristic search to construct the output quantum circuit step by step from the original input quantum circuit [Finigan et al. 2018; Li et al. 2019; Oddi and Rasconi 2018; Paler 2019; Siraichi et al. 2018; Zhou et al. 2020b; Zulehner et al. 2018]. Experimental results show that customised heuristic search algorithms are more promising in transforming large-scale circuits, but usually there is still a considerable gap between the output circuit and an optimal one. The reason partially lies in the limited search depth in most of these algorithms. To achieve efficiency, one either divides the circuits into layers and tries to execute the gates layer-wise [Zulehner et al. 2018], or simply considers only the direct effect of a single move (i.e., SWAP) (see e.g., [Childs et al. 2019; Cowtan et al. 2019; Li et al. 2019]). This leads to a very shallow search depth. The Simulated Annealing and Heuristic Search algorithm (SAHS) [Zhou et al. 2020b] and the Filtered and Depth-Limited Search approach (FiDLS) [Li et al. 2021] can go one or two steps further, but exploring even more seems impractical as the searching process will become very slow if many qubit connections are present in the QPU. Recently, machine learning techniques have also been exploited to provide a more precise evaluation tool for QCT algorithms



(a) Selection



(b) Expansion

Node	EBBP	EABP1	EABP2
0	6/14	8/16	9/18
1	3/6	3/6	3/6
2	4/6	4/8	5/10
3	1/2	1/2	1/2
4	1/2	1/2	1/2
5	2/2	2/2	2/2
6	2/2	4/4	5/6
7	0/2	0/2	0/2
8	unopen	0/2	0/2
9	unopen	0/0	1/2

(c) Simulation and Backpropagation

Fig. 1. Example for one playout in MCTS. Each grey or white node in (a) and (b) represents a legal move made by the corresponding player. The last three columns in (c) represent, respectively, evaluations in each node before, after the first, and after the second *Backpropagation*. The evaluation is defined as $\#wins/\#simulations$ obtained by *Simulation* and Backpropagation.

[Pozzi et al. 2020; Sinha et al. 2021; Zhou et al. 2021]. Whereas, those algorithms often suffer from a poor scalability in terms of the number of qubits of the NISQ device.

Inspired by the recent spectacular success of Monte Carlo Tree Search (MCTS) in Computer Go play [Silver et al. 2016, 2017], in this paper, we propose an MCTS framework for the QCT problem. Although first designed for solving computer games, MCTS has found applications in many domains which can be represented as trees of sequential decisions [Browne et al. 2012]. MCTS is a flexible statistical anytime algorithm, which can be used with little or no domain knowledge [Browne et al. 2012]. The basic idea behind MCTS is to explore and exploit, in a balanced way, a search tree in which each node represents a game state and each branch a legal move starting from that state. Given the current game state, the aim is to select the most promising move by exploring a search tree rooted with this state, based on random sampling of the search space. This is achieved through the following five steps: (1) *Selection*. Starting from the root, we first select successively a child node until a leaf node is reached; (2) *Expansion*. Expand the selected leaf node with one or more child nodes each of which corresponding to a legal move; (3) *Simulation*. Play out the task to completion by selecting subsequent moves randomly; (4) *Backpropagation*. Backpropagate the simulation result (wining, losing, or the reward points collected) towards the root node to update the values of nodes along the way; (5) *Decision*. After repeated a sufficient number of times, we then select the best move (with the largest value) and move to the next game state.

Example 1. *We show how to conduct a full playout based on a search tree as shown in Fig. 1(a). Suppose a simple strategy only choosing child with maximum winning rate*[1] *is used in* Selection. *Then starting from root node 0, nodes 2 and 6 with maximum #wins/#simulations values* $4/6$ *and* $2/2$ *among their peers according to the data in column 'EBBP', Evaluation Before BackPropagation, in Fig. 1(c) will be chosen successively. Because node 6 is a leaf, it will be expanded and its child nodes 8 and 9, as shown in the dashed box of Fig. 1(b), will be opened. After* Expansion*, one or more newly opened nodes will be chosen to perform simulations. In this example, both nodes 8 and 9 are chosen to execute 2 random simulations and the results are assumed to be* $0/2$ *and* $1/2$ *respectively. After all simulations in node 8 are done, the result will be back propagated to root node 0 through nodes 6 and 2, and their values will be updated and are marked red in the 'EABP1', Evaluation After BackPropagation, column of Fig. 1(c). To be specific, the denominator values of nodes 6, 2 and 0 along the backpropagation path will be increased by 2 because the same number of new simulations are done in node 8; only the numerator values of white nodes 6 and 0 are increased because the black player lost both simulations. The same*

[1]The strategy for *Selection* in practice is much more complex than this and should take both evaluations and time of visits into account. Interested readers can refer to [Chaslot et al. 2008; Kocsis and Szepesvári 2006] for further details.

operation applies after the simulations in node 9 are finished and the updated values can be found in the 'EABP2' column.

Our MCTS framework for the QCT problem also consists of these five major modules. In the framework, we adopt a fast random strategy for simulation and carefully design a scoring mechanism which takes both short and long-term rewards into consideration. Based on the five modules and the scoring mechanism, an algorithm, abbreviated as MCTS-Size, is proposed to optimise the size of the output circuit. The algorithm is polynomial in all relevant parameters and experiments on an extensive set of realistic benchmark circuits show that the search depth can easily exceed most, if not all, existing algorithms. The search depth in the proposed algorithm is defined as the depth of the selected leaf node to the root during each invoking of the Selection module. In Example 1, node 6 is chosen in Selection and its search depth is 2. This deep search method can reduce the gate overhead of the output physical circuits by a large margin when compared with the state-of-the-art algorithms [Cowtan et al. 2019; Li et al. 2021; Zhou et al. 2020b] on IBM Q20.

Although aiming to optimise the circuit size in terms of gate numbers, MCTS-Size also reduces the depth of the output circuit significantly. Depth is perhaps a more significant criterion for quantum circuits due to the highly limited coherence time in NISQ devices. When compared with t|ket⟩ introduced in [Cowtan et al. 2019], a state-of-the-art and industrial level algorithm aiming at depth optimisation, MCTS-Size reduces the circuit size and depth overheads by, respectively, 66% and 75% on IBM Q20 (cf. Table 1). More importantly, as our MCTS framework is flexible, it can be easily adapted to accommodate various optimisation criteria. To exemplify this feature, we design MCTS-Depth by introducing two very simple modifications to MCTS-Size. Experimental results on IBM Q20 show that, compared to t|ket⟩ again, MCTS-Depth is able to reduce the depth overhead up to 84%.

This paper is a significant extension of the conference paper [Zhou et al. 2020a] presented at ICCAD'20. Among others, we have made the following major extensions: (a) aiming to optimise the output circuit depth, we design the MCTS-Depth algorithm (cf. Sec. 4) (note that [Zhou et al. 2020a] only considered optimisation of the output circuit size); (b) to further demonstrate the flexibility of our framework, we incorporate remote CNOT gates into the MCTS-based algorithms (cf. Sec. 5), which are also known as bridge gates and can execute CNOT gates whose two qubits are not neighbours without changing the current mapping; (c) we describe in detail the parameter selection process and empirically compare the search depth of the MCTS-Size algorithm with that of SAHS [Zhou et al. 2020b] (cf. Sec. 6); (d) we present detailed and additional empirical evaluation results (considering depth reduction as well as the effect of remote CNOT gates) on IBM Q20, a hypothetical grid-like QPU called Grid 4×5, and IBM Rochester and Google Sycamore, and with two other state-of-the-art algorithms, viz., Qiskit and SABRE [Li et al. 2019] (cf. Sec. 6).

The remainder of this paper is organised as follows: Sec. 2 provides some background knowledge about quantum computation and summarises the state-of-the-art of the quantum circuit transformation problem. Sec. 3 then presents a detailed description of the MCTS framework as well as a theoretical analysis. The adapted depth-optimisation algorithm is presented in Sec. 4. After that, we show how to incorporate remote CNOT in the MCTS-based algorithms in Sec. 5. Empirical evaluations of both MCTS-based algorithms on an extensive set of realistic benchmark circuits and on various QPUs are presented in Sec. 6. The last section concludes the paper with an outlook.

2 QUANTUM CIRCUIT TRANSFORMATION

In classical computing, data are stored in the form of bits which can take one of two states, 0 and 1. In contrast, data in quantum computing are stored in *qubits*, which also have two basis states represented by $|0\rangle$ and $|1\rangle$, respectively. However, unlike a classical bit, a qubit can be in

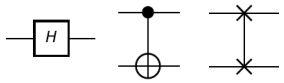


Fig. 2. Hadamard, CNOT and SWAP gates (from left to right).

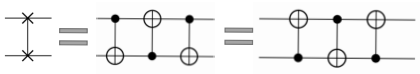


Fig. 3. The decomposition of a SWAP into three CNOT gates.

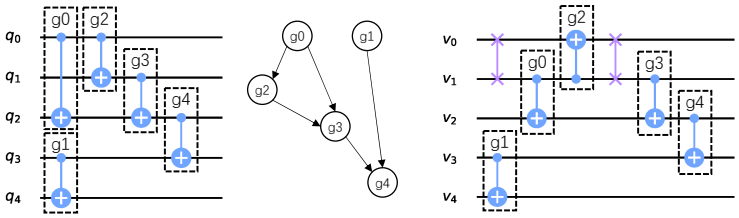


Fig. 4. A quantum circuit (left), its dependency graph (middle) and the circuit after transformation (right).

the superposition $\alpha\,|0\rangle + \beta\,|1\rangle$ of basis states, where α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$.

The state of a qubit can be changed by quantum gates, which are mathematically represented by unitary matrices. Fig. 2 depicts three important quantum gates used in this paper: Hadamard, CNOT and SWAP gates. Hadamard is a single-qubit gate that has the ability to generate superposition: it maps $|0\rangle$ to $(|0\rangle+|1\rangle)/\sqrt{2}$ and $|1\rangle$ to $(|0\rangle-|1\rangle)/\sqrt{2}$. CNOT and SWAP are both two-qubit gates. CNOT flips the target qubit depending on the state of the control qubit; that is, CNOT: $|c\rangle\,|t\rangle \longrightarrow |c\rangle\,|c \oplus t\rangle$, where $c, t \in \{0, 1\}$ and $\oplus$ denotes exclusive-or. SWAP exchanges the states of its operand qubits: it maps $|a\rangle\,|b\rangle$ to $|b\rangle\,|a\rangle$ for all $a, b \in \{0, 1\}$. Note that a SWAP gate can be decomposed into three CNOT gates as shown in Fig. 3.

Quantum gates can be concatenated to form complex *circuits* which, together with measurements, are used to describe quantum algorithms. A circuit is usually denoted by a pair (Q, C), where Q is a set of qubits and C a sequence of quantum gates on Q. Sometimes we also call C a circuit when Q is clear from the context. Fig. 4 shows a circuit where $Q = \{q_0, \ldots, q_4\}$, $C = (g_0, \ldots, g_4)$, $g_0 = \text{CNOT}(q_0, q_2)$, $g_1 = \text{CNOT}(q_3, q_4)$, etc. Here each CNOT is annotated with the qubits on which they are applied.

2.1 Quantum Circuit Transformation: Problem Formulation

As mentioned in the introduction, to run a quantum circuit on a given QPU in the NISQ era, we need to transform it so that the connectivity constraints imposed by the QPU are all satisfied. Such connectivity constraints are typically described as an undirected and connected graph $AG = (V, E)$, called the *architecture graph* [Childs et al. 2019], where V denotes the set of physical qubits of the QPU and E the pairs of physical qubits on which a two-qubit gate can be applied.

Note that by a standard process [Nielsen and Chuang 2002], any quantum circuit can be decomposed into a functionally equivalent one which consists of only CNOT and single-qubit gates. Furthermore, as single-qubit gates can be executed directly on a QPU (connectivity constraints only prevent two-qubit gates from applying on certain pairs of physical qubits), if not otherwise stated,

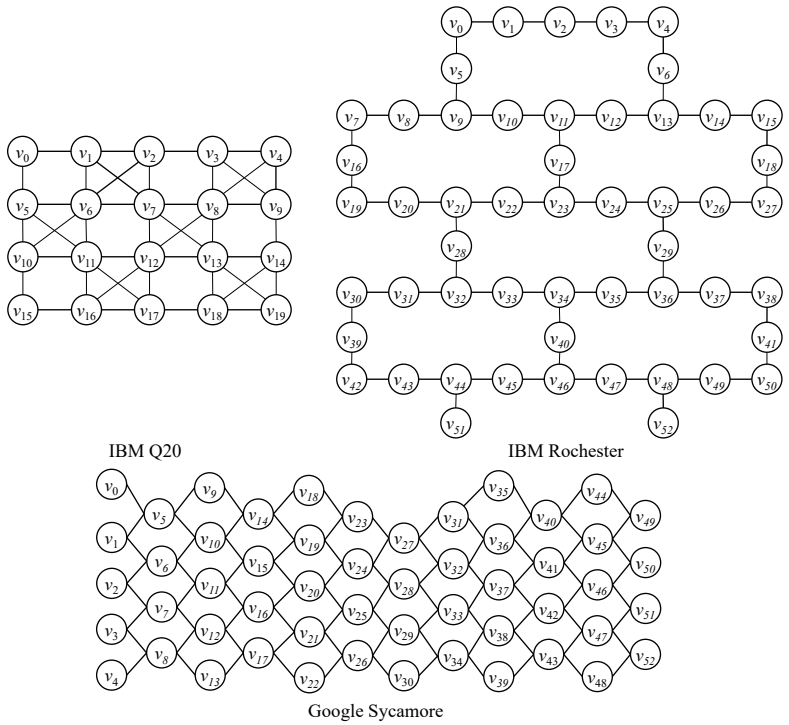


Fig. 5. The architecture graphs for IBM Q20, IBM Rochester and Google Sycamore.

we assume that single-qubit gates have been removed and the circuit to be transformed consists solely of CNOT gates.[2] Note that this is only a technical assumption and, whenever necessary, we can always add back the corresponding single-qubit gates (cf. Sec. 4). Besides that, the SWAP gates added during the transformation process will be decomposed into CNOTs in the end.

An important notion related to quantum circuits which plays a key role in QCT is the *dependency graph*. Let $C = (g_0, g_1, \ldots)$ be a quantum circuit. We say gate g_i in C *depends* on g_j if $j < i$ and they share at least one common qubit. The dependence is *direct* if there is no gate g_k with $j < k < i$ such that g_i depends on g_k and g_k depends on g_j. In general, we can construct a directed acyclic graph (DAG), called the dependency graph [Itoko et al. 2019], to characterise the dependency between gates in a circuit. Specifically, each node of the dependency graph represents a gate and each directed edge the direct dependency relationship between the gates involved. Let's say a gate g_i directly depends on g_j, then the corresponding edge (g_j, g_i) should be added to the dependency graph. With the help of dependency graph, any quantum circuit C can be divided into different *layers* such that gates in the same layer can be executed in parallel. The first or front layer, denoted by $\mathcal{L}_0(C)$, consists of the gates which have no parents in the DAG. The second layer, $\mathcal{L}_1(C)$, is then the front layer of the DAG obtained by deleting all gates in $\mathcal{L}_0(C)$. Analogously, we can define the i-th layer of a circuit for any $i \geq 0$.

[2]This implies that we cannot simplify the circuits by, say, cancelling two consecutive CNOT gates acting on the same pair of qubits.

Example 2. *Fig. 4 shows an example of a quantum circuit (left) and its dependency graph (right), from which we can see that the front layer of the circuit consists of g_0 and g_1, the second g_2, the third g_3, and the fourth g_4.*

Another key notion for QCT is a qubit mapping τ which allocates logical qubits Q to physical qubits V so that for any $q_i, q_j \in Q$, $\tau(q_i) = \tau(q_j)$ if and only if $i = j$. Given a logical circuit (Q, C) and an architecture graph AG, a two-qubit gate $g = \text{CNOT}(q_i, q_j)$ in C is called *executable* by τ if $\tau(q_i)$ and $\tau(q_j)$ are adjacent in AG, and g is either in the front layer of C or all the gates it depends on are executable. Note that in general it is impossible that all two-qubit gates in a circuit are executable by a single mapping. Once no gates are executable by the current mapping τ, a QCT algorithm seeks to insert into the circuit some ancillary SWAP gates to change τ into a new one so that more gates are executable. This insertion-execution process is iterated until all gates from the input circuits are executed. To illustrate the basic ideas, we revisit the circuit on the left side of Fig. 4.

Example 3. *We transform the logical circuit $LC = (Q, C^l)$ shown in Fig. 4 into a physical one $PC = (V, C^p)$ satisfying the architecture graph AG in Fig. 5. Suppose the initial qubit mapping τ is given as a naive one which maps q_i to v_i, $0 \le i \le 4$.*

(1) *Since $\tau(q_3) = v_3$, $\tau(q_4) = v_4$, and v_3 and v_4 are adjacent in AG, gate g_1 in C^l is already executable by τ. Thus we initialise PC as a physical circuit with $V = \{v_0, \ldots, v_{19}\}$ containing only a single CNOT gate acting on v_3 and v_4, and delete g_1 from C^l. Thus now, $C^l = (g_0, g_2, g_3, g_4)$ and $C^p = (\text{CNOT}(v_3, v_4))$.*

(2) *As no gates in C^l is executable by τ, we have to insert a SWAP (or a sequence of them) to get a new mapping which admits more CNOT gates from C^l executable. In this example, we choose to add $\text{SWAP}(v_0, v_1)$ to C^p, which in effect converts τ into τ' that maps q_0 to v_1 and q_1 to v_0. Now g_0, which acts on q_0 and q_2, is executable (since v_1 and v_2 are adjacent in AG). Similarly, g_2 is executable as well. Thus they can be deleted from C^l and added into C^p (with the operand qubits changed accordingly). Consequently, now $C^l = (g_3, g_4)$ and*

$$C^p = (\text{CNOT}(v_3, v_4), \text{SWAP}(v_0, v_1), \text{CNOT}(v_1, v_2), \text{CNOT}(v_1, v_0)).$$

(3) *Proceeding in a similar way, we add another $\text{SWAP}(v_0, v_1)$ to C^p to converts τ' back to τ so that g_3 and g_4 are executable. After deleting them from C^l and adding them into C^p, we have $C^l = \emptyset$ and the finial physical circuit becomes*

$$C^p = \big(\text{CNOT}(v_3, v_4), \text{SWAP}(v_0, v_1), \text{CNOT}(v_1, v_2), \text{CNOT}(v_1, v_0),$$
$$\text{SWAP}(v_0, v_1), \text{CNOT}(v_1, v_2), \text{CNOT}(v_2, v_3)\big),$$

which satisfies all the connectivity constraints of AG. The final physical circuit is shown in Fig. 4 (right).

2.2 Heuristic Search Algorithms

Recall that given a logical circuit LC_0, an architecture graph AG, and an initial qubit mapping τ_{ini}, the QCT process aims to output a physical circuit which respects all the connectivity constraints in AG. To present this process as a search problem, we need to first define the notion of states. Naturally, a *state* of the QCT process is a triple $\mathbf{s} = (\tau, PC, LC)$, where τ is a qubit mapping describing the current allocation of logical qubits, PC is the physical circuit that consists of all gates that have been executed so far and the auxiliary SWAP gates inserted and the logical circuit LC consists of the remaining gates to be executed. Sometimes we denote by $LC(\mathbf{s})$ and $PC(\mathbf{s})$ the logical and the physical circuits of $\mathbf{s}$, respectively.

A *legal action* in the QCT process can be either a SWAP operation (corresponding to an edge in AG) or a sequence of SWAP operations.[3] Let $\mathbf{s} = (\tau, PC, LC)$ be the current state, and suppose an action $\mathrm{SWAP}(v_i, v_j)$ is taken on $\mathbf{s}$. Then a new state $\mathbf{s}' = (\tau', PC', LC')$ is reached where τ' is the same as τ except that it maps $\tau^{-1}(v_i)$ to v_j and $\tau^{-1}(v_j)$ to v_i, where $\tau^{-1}(v_i)$ and $\tau^{-1}(v_j)$ are, respectively, the preimages of v_i and v_j under τ. Furthermore, LC' is obtained from LC by deleting all gates which are executable by τ', and PC' is obtained from PC by adding first $\mathrm{SWAP}(v_i, v_j)$ and then all the gates just deleted from LC, with the operand qubits changed according to τ'. While most algorithms select one SWAP each time, the A^* algorithm [Zulehner et al. 2018] and FiDLS [Li et al. 2021] select a sequence of SWAPs. Note that when regarding sequences of SWAPs as legal actions, usually we execute a gate only after the last SWAP is applied.

The *initial state* $\mathbf{s}_0$ of the QCT process is taken as $(\tau_{ini}, PC_0, LC_0')$ where PC_0 is the physical circuit consisting of all gates from LC_0 which are executable by τ_{ini}, and LC_0' the logic circuit obtained by deleting all gates in PC_0 from LC_0. The *goal states* are those with the associated logical circuit being empty. Note that the associated physical circuit of any goal state respects the connectivity restraints in AG. The *cost* of a state $\mathbf{s}$ depends on the optimisation objective. In this paper, it can be either the total number of auxiliary gates inserted or the depth overhead of the stored physical circuit of $\mathbf{s}$. The aim of QCT is to find a goal state with the minimal cost w.r.t. the particular objective.

Many QCT algorithms in the literature adopt a divide-and-conquer approach in the search process. Starting from the current state $\mathbf{s} = (\tau, PC, LC)$, each subtask consists of executing the front layer, the first two layers, or a front section of the circuit. For example, in the A^* algorithm, a shortest path in AG (which corresponds to a sequence of SWAPs) is found which converts τ to a new mapping so that all gates in the first two layers of LC are executable. In [Cowtan et al. 2019], Cowtan et al. partition LC into layers and then select the SWAP which can maximally reduce the diameter of the subgraph composed of all pairs of qubits in the current layer. Siraichi et al. [Siraichi et al. 2019a] decompose LC into sub-circuits each of which leads to an isomorphic subgraph of AG and thus the corresponding embedding can act as a mapping τ' that executes all gates in the sub-circuit. Their algorithm then tries to find a minimal sequence of SWAPs which converts τ to τ'. A similar approach is also adopted in Childs et al. [Childs et al. 2019].

Unlike the above algorithms, SAHS [Zhou et al. 2020b] and FiDLS [Li et al. 2021] do not divide the problem into sub-problems. Whenever a mapping is generated, they try to execute as many as possible gates from the logical circuit, no matter which level they are in. SAHS regards each SWAP as a valid action, but when selecting the best SWAP to enforce, it simulates the search process one step further and select the SWAP which has the best consecutive SWAP to apply. In principle, SAHS can go deeper but this will make the algorithm much slower (cf. Fig. 11 for an example). FiDLS regards any sequence with up to k SWAPs as a legal action and selects the sequence which executes the most number of gates per SWAP. In a sense, this means that its search depth can reach k. To ensure the running time is acceptable, in the experiments on Q20, FiDLS chooses k as 3 and introduces various filters to filter out unlike SWAPs.

3 THE PROPOSED MCTS FRAMEWORK

In this section, we describe an MCTS framework for quantum circuit transformation and present a detailed algorithm implementation. The algorithm, called MCTS-Size, aims at finding a goal state which has the minimal number of SWAPs inserted. Shortly in Sec. 4 we shall see this can be easily adapted to address other optimisation objectives.

[3] In Sec. 5 we will relax this restriction and allow remote CNOTs to be legal actions.

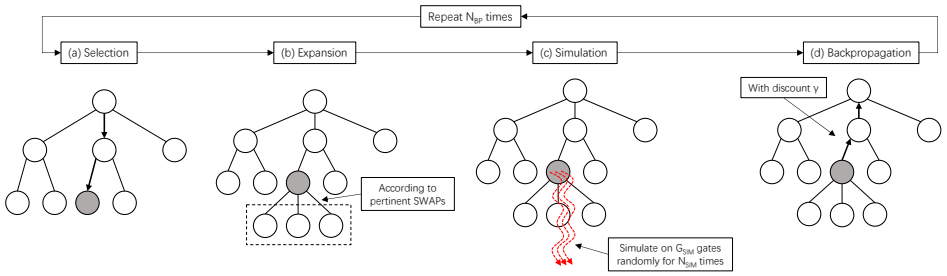


Fig. 6. Overview of the Monte Carlo Tree Search Framework.

Like general MCTS algorithms, our framework also consists of five major parts: *Selection*, *Expansion*, *Simulation*, *Backpropagation* and *Decision*. However, some significant modifications have been made to cater to the unique characteristics of QCT.

The Monte Carlo search tree for QCT, which is initialised immediately after the algorithm starts, stores all states having been explored during the transformation process. In practical implementation, it is not necessary to store the full physical and logical circuits in a state; instead, only the incremental information, i.e., the gates added to and removed from, respectively, the physical and logical circuits of its parent state, are stored. When necessary, the circuits of a state can be restored from the incremental information in itself and its ancestor states. As stated in the previous section, an edge $(\mathbf{s}', \mathbf{s})$ connecting node $\mathbf{s}'$ and its child $\mathbf{s}$ indicates that a SWAP is applied to convert $\mathbf{s}'$ to $\mathbf{s}$. With the aim to minimise the number of inserted gates, we define an immediate, short-term *reward* for each edge and a long-term *value* for each node of the search tree as follows.

The short-term reward $\textsc{rwd}(\mathbf{s}', \mathbf{s})$ is the reward collected from the parent node $\mathbf{s}'$ to the child $\mathbf{s}$, in terms of the number of gates executed by the newly inserted SWAP when this transition is made:

$$\textsc{rwd}(\mathbf{s}', \mathbf{s}) = \#\text{gates\_in\_}LC(\mathbf{s}') - \#\text{gates\_in\_}LC(\mathbf{s}). \tag{1}$$

The long-term value $\textsc{val}(\mathbf{s})$. To determine the value of a state $\mathbf{s}$, the following two factors are taken into account: (i) the (inverse of the) number of inserted SWAPs when transformation of the remaining logical circuit is simulated at $\mathbf{s}$. For efficiency, the simulation is performed on, instead of $LC(\mathbf{s})$ itself, a fixed-size sub-circuit of $LC(\mathbf{s})$. It is expected that the larger the sub-circuit is for simulation, the better simulated value will be obtained. (ii) the (simulated) value of its best child node and the reward to it collected from $\mathbf{s}$. To be specific,

$$\textsc{val}(\mathbf{s}) = \max\{\textsc{sim}, \gamma \cdot [\textsc{rwd}(\mathbf{s}, \mathbf{s}'') + \textsc{val}(\mathbf{s}'')]\},$$

where $\textsc{sim}$ is the simulated value obtained from (i), $\mathbf{s}''$ is the child of $\mathbf{s}$ with the maximal value, and γ is a predefined discount factor satisfying $\gamma < 1$. In our later implementation, $\textsc{val}(\mathbf{s})$ is initially assigned $\textsc{sim}$ in the *Simulation* module, and then updated in *Backpropagation*, whenever simulations are performed at a descendant of $\mathbf{s}$. Intuitively, $\textsc{val}(\mathbf{s})$ describes the efficiency of introducing SWAPs (in terms of the average number of executed gates per SWAP) from $\mathbf{s}$, considering both the simulation at itself and the backpropagated one from this child nodes. Obviously, the larger $\textsc{val}(\mathbf{s})$ is, the smaller the number of SWAPs needed to lead $\mathbf{s}$ to a goal node, and the 'better' $\mathbf{s}$ is (compared with its siblings).

In addition to the above definitions, as shown in Fig. 6, our framework differs from traditional MCTS algorithms for game playing in the following ways:

(1) The simulation is performed on the leaf node selected in the *Selection* module, instead of the child nodes opened in the *Expansion* one. Experimental results on real benchmarks indicate that this achieves a better performance for the QCT problem.
(2) In game playing, the simulation result can be obtained only when the game is decided. In contrast, the reward of a move in our setting is collected during the execution of CNOT gates from the logic circuit. Consequently, in the *Simulation* module, we simulate only on a sub-circuit of the current logic circuit to improve efficiency.
(3) We introduce a discount factor, which can be adjusted to better suit the problem setting, when backpropagating the simulated values.

3.1 Main modules

We now elaborate the five major modules one by one.

Selection. *Selection* is the iterated process to find an appropriate leaf node in the search tree to expand and simulate. It starts from the root node and, in each iteration, evaluates and picks one of the child nodes until a leaf node is reached.

The way we evaluate child nodes during *Selection* is critical to the performance of the whole algorithm. On one hand, if we only consider their values, the chance for exploring unpromising nodes will be too low and we can easily get stuck in a local minimum. On the other hand, if we always select nodes with a smaller visit count, the search will be too shallow and thus a large amount of time will be wasted in exploring inferior nodes. To get a balance between these two aspects, the following evaluation formula, similar to the well-known UCT (Upper Confidence Bound 1 applied to trees) [Kocsis and Szepesvári 2006], is introduced in our implementation to make a balanced evaluation among all child nodes $\mathbf{s}'$ of $\mathbf{s}$:

$$\textsc{rwd}(\mathbf{s},\mathbf{s}') + \textsc{val}(\mathbf{s}') + c\sqrt{\frac{\log \textsc{visit}(\mathbf{s})}{\textsc{visit}(\mathbf{s}')}} \tag{2}$$

where c is a pre-defined parameter, and $\textsc{visit}(\mathbf{s})$ is the number of times that $\mathbf{s}$ has been visited. Intuitively, the first two terms in Eq. (2) correspond to the exploitation rate and the third the exploration rate in UCT. In each iteration of the *Selection* module, the node which maximises Eq. (2) is selected. The *Selection* module is presented in Alg. 1.

Algorithm 1: Select($\mathcal{T}$)

input : A Monte Carlo search tree $\mathcal{T}$.
output: A leaf node to expand and simulate.

```
s ← root(T);
visit(s) ← visit(s) + 1;
while s is not a leaf node do
    s ← the child node s' of s that maximises Eq. (2);
    visit(s) ← visit(s) + 1;
return s;
```

Expansion. The goal of *Expansion* is to open all child nodes of a given leaf node by applying all relevant SWAP operations. Given a logic circuit C and a qubit mapping τ, the set of *pertinent* SWAPs, denoted $\mathcal{SWAP}_{C,\tau}$, is the set of gates $\mathrm{SWAP}(v_i, v_j)$ such that either $\tau^{-1}(v_i)$ or $\tau^{-1}(v_j)$ appears in a gate in the current front layer of C, i.e.,

$$(v_i, v_j) \in E \text{ and } (\tau^{-1}(v_i) \in Q_0 \text{ or } \tau^{-1}(v_j) \in Q_0),$$

where Q_0 is the set of logical qubits that are involved in the gates in $\mathcal{L}_0(C)$. To expand a selected node $\mathbf{s} = (\tau, PC, LC)$, only gates in $\mathcal{SWAP}_{LC,\tau}$ will be applied to generate child nodes. This strategy has been widely used in quantum circuit transformation, see, e.g., [Li et al. 2019; Zhou et al. 2020b; Zulehner et al. 2018]. In particular, several variants are introduced in FiDLS [Li et al. 2021].

For each pertinent SWAP of $\mathbf{s}$, a new child node $\mathbf{s}'$ will be generated. Furthermore, the reward $\textsc{rwd}(\mathbf{s}, \mathbf{s}')$ is as defined in Eq. (1) and both $\textsc{val}(\mathbf{s}')$ and $\textsc{visit}(\mathbf{s}')$ are set as 0. The details can be found in Alg. 2.

Algorithm 2: Expand($\mathcal{T}, \mathbf{s}$)

input : A Monte Carlo search tree $\mathcal{T}$ and node $\mathbf{s} = (\tau, PC, LC)$.

for *all* SWAP(v_i, v_j) *in* $\mathcal{SWAP}_{LC,\tau}$ **do**

- $\tau' \leftarrow \tau[\tau^{-1}(v_i) \mapsto v_j, \tau^{-1}(v_j) \mapsto v_i]$;
- $C \leftarrow$ the set of all τ'-executable gates in LC;
- $LC' \leftarrow LC$ with all gates in C deleted;
- $PC' \leftarrow PC$ by adding SWAP(u_i, u_j) and all gates in C;
- $\mathbf{s}' \leftarrow (\tau', PC', LC')$;
- $\textsc{val}(\mathbf{s}'),\ \textsc{visit}(\mathbf{s}') \leftarrow 0$;
- Add $\mathbf{s}'$ as a child node of $\mathbf{s}$;
- $\textsc{rwd}(\mathbf{s}, \mathbf{s}') \leftarrow$ number of gates in C;

Simulation. The objective here is to obtain a simulated score, serving as the initial long-term value $\textsc{val}(\mathbf{s})$, of the current state $\mathbf{s}$ by simulation. In our implementation, we perform simulation on the first G_{SIM}, a predefined number, gates in the current logical circuit. While almost all existing QCT algorithms can be used for this purpose, for the sake of efficiency, a fast random simulation is designed in Alg. 3. Related to this, in Sec. 6.6, we shall see an MCTS algorithm with a deterministic simulation module.

Given the current state $\mathbf{s}$, let N be, among all N_{SIM} (a predefined number) iterations, the minimal number of SWAP gates we have inserted until all the first G_{SIM} CNOT gates of $LC(\mathbf{s})$ have been executed. Then the initial long-term value of $\mathbf{s}$ is defined as

$$\textsc{val}(\mathbf{s}) = \gamma^{N/2} \cdot G_{\text{SIM}}, \tag{3}$$

where $\gamma < 1$ is a predefined discount factor. What deserves explanation is the way we compute the simulated score (or, the initial value) for state $\mathbf{s}$ in Eq. (3). In particular, one may wonder why we take as the exponent $N/2$ instead of N? The intuitive meaning of this definition is as follows. Although these G_{SIM} gates are executed in different steps during the simulation, for simplicity, we suppose they are all executed right at the middle point $\mathbf{s}'$ which is the $N/2$-generation child of $\mathbf{s}$. Then the reward collected at the transition to $\mathbf{s}'$ from its parent is exactly G_{SIM}. Note that every edge along the path from $\mathbf{s}$ to the parent of $\mathbf{s}'$ has zero reward. Thus, we need only backpropagate the reward collected at $\mathbf{s}'$ upwards with discount factor γ. This gives the simulated score $\gamma^{N/2} \cdot G_{\text{SIM}}$ for $\mathbf{s}$ as specified in Eq. (3). Real benchmark experiments also confirm that the current choice performs better than simply letting $\textsc{val}(\mathbf{s})$ be the sum of all the (discounted) rewards collected during the actual execution of these G_{SIM} CNOT gates.

We next show how to do random simulation. Let C be a sub-circuit of $LC(\mathbf{s})$ and τ the current mapping. We write $\mathcal{SWAP}_{C,\tau}$ for the set of pertinent SWAPs for C under τ. For any $h \in \mathcal{SWAP}_{C,\tau}$,

its *impact factor* is defined as

$$\mathrm{IF}(h) := f\Big(\sum_{g\in\mathcal{L}_0(C)} \mathrm{SCOST}(g,\tau) - \sum_{g\in\mathcal{L}_0(C)} \mathrm{SCOST}(g,\tau')\Big) \tag{4}$$

where τ' is the mapping obtained from τ after applying h; $\mathrm{SCOST}(g,\delta)$ for $\delta=\tau$ or $\delta=\tau'$ is the *swap cost* of $g=\mathrm{CNOT}(q_j,q_k)$ with respect to mapping δ, defined as the shortest distance between the physical qubits $\delta(q_j)$ and $\delta(q_k)$ in the architecture graph, in which the edges have an uniform weight of 1 and thus the distance is the summed edge weights; and f the scaling function defined as

$$f(x)=\begin{cases} 0, & \text{if } x<0\\ 0.001, & \text{if } x=0\\ x, & \text{if } x>0\end{cases}$$

which is slightly different from the Relu function in that it returns a tiny positive value (0.001 in our case) instead of 0 when $x=0$. We make this change to ensure that SWAPs which increase the cost will not be selected when no SWAP can decrease the cost.

Then, a probability distribution is obtained as follows

$$P(X=h)=\frac{\mathrm{IF}(h)}{\sum\{\mathrm{IF}(h')\mid h'\in\mathcal{SWAP}_{C,\tau}\}}, \tag{5}$$

through which a SWAP operation can be sampled from $\mathcal{SWAP}_{C,\tau}$ and used to execute gates from LC. Note that this simulation process will be repeated for N_{SIM}, also a predefined parameter, times to obtain the best score.

Algorithm 3: Simulate($\mathcal{T}$, $\mathbf{s}$)

input : A Monte Carlo search tree $\mathcal{T}$ and node $\mathbf{s}=(\tau,PC,LC)$.

```
N ← ∞;
do
    C ← circuit with the first G_SIM gates in LC;
    n ← 0; τ' ← τ;
    while C is not empty do
        Sample h from SWAP_{C,τ'} according to the probability distribution in Eq. (5);
        τ' ← τ' by applying h;
        C ← C with all τ'-executable gates deleted;
        n ← n + 1;
    if n < N then
        N ← n;
for N_SIM times;
VAL(s) ← γ^{N/2} · G_SIM;
```

Backpropagation. The *Backpropagation* module updates the values of ancestors of the just simulated node in the search tree. More precisely, the value of node $\mathbf{s}$ in the propagated path will be updated as

$$\mathrm{VAL}(\mathbf{s}) \leftarrow \max\big\{\mathrm{VAL}(\mathbf{s}),\ \gamma\cdot[\mathrm{RWD}(\mathbf{s},\mathbf{s}')+\mathrm{VAL}(\mathbf{s}')]\big\}, \tag{6}$$

in which $\mathbf{s}'$ is the child node of $\mathbf{s}$ on the path. This reflects the intuitive meaning of $\mathrm{VAL}(\mathbf{s})$ discussed at the beginning of this section. The implementation is shown in Alg. 4.

Algorithm 4: Backpropagate($\mathcal{T}$, $\mathbf{s}$)

input : A Monte Carlo search tree $\mathcal{T}$ and node $\mathbf{s}$.

while $\mathbf{s} \neq root(\mathcal{T})$ **do**

  $\mathbf{s}' \leftarrow$ parent node of $\mathbf{s}$;

  $\textsc{val}(\mathbf{s}') \leftarrow \max\{\textsc{val}(\mathbf{s}'), \gamma \cdot [\textsc{rwd}(\mathbf{s}', \mathbf{s}) + \textsc{val}(\mathbf{s})]\}$;

  $\mathbf{s} \leftarrow \mathbf{s}'$;

Decision. This module, depicted in Alg. 5, decides the best move from the root node rt and updates the search tree with the subtree rooted at the best child node of rt.

Algorithm 5: Decide($\mathcal{T}$)

input : A Monte Carlo search tree $\mathcal{T}$.

$rt \leftarrow root(\mathcal{T})$;

$\mathbf{s} \leftarrow$ child node of rt with the highest $\textsc{rwd}(rt, \mathbf{s}) + \textsc{val}(\mathbf{s})$;

$\mathcal{T} \leftarrow$ the subtree of $\mathcal{T}$ rooted at $\mathbf{s}$;

3.2 Combine Everything Together

Finally, we combine all modules together as in Alg. 6 to form the MCTS framework for QCT. Note that, to ensure the reliability of the *Decision* module, a sufficiently large number (N_{BP}, a predefined parameter) of *Selection*, *Expansion*, *Simulation*, and *Backpropagation*, should be performed to get a good estimation of the values of relevant states.

Due to the stochastic nature of our algorithm, there is a negligible but still positive possibility that at certain iteration of the while loop in Alg. 6, even the best child node derived from the *Decision* module cannot execute any new gate. To guarantee termination in this extreme case, a *fallback* mechanism, which has been widely used in the literature (cf. [Childs et al. 2019]), is adopted. Specifically, if no CNOTs have been executed after $|V|$ consecutive *Decisions* and the current root node is (τ, PC, LC), then we choose a CNOT from $\mathcal{L}_0(LC)$ with minimum swap cost with respect to τ, and insert the corresponding SWAP gates to PC so that progress will be made by executing this chosen CNOT. For the sake of readability, the fallback module is omitted in Alg. 6.

3.3 Complexity Analysis

This subsection is devoted to a rough analysis of the complexity of our algorithm. Suppose $AG = (V, E)$ and the input logical circuit $LC = (Q, C)$. Among the five main modules presented in subsection 3.1, the most expensive ones are *Selection*, *Expansion*, and *Simulation*. We analyse their complexity separately as follows.

Selection. The complexity of this module depends on the depth of the search tree. In the worst case, each of the N_{BP} iteration in the **do** loop of Alg. 6 increases the depth by 1. Taking into account the fallback introduced in the last subsection, the depth of the search tree is at most $N_{\text{BP}} \cdot |V|$. As each node has at most $|E|$ children, the overall complexity for this module is $O(N_{\text{BP}} \cdot |V| \cdot |E|)$.

Expansion. There are at most $|E|$ pertinent SWAP gates available to create new nodes, and for each new one, at most $|C|$ gates need to be checked to see whether they are executable. Thus the time complexity is $O(|E| \cdot |C|)$. Here $|C|$ denotes the number of gates in C.

Simulation. Computing the probability distribution in Eq. (5) takes time $O(|E| \cdot |V|)$. To guarantee termination, the while loop will be aborted if no gates have been executed after $|V|$ consecutive iterations. Hence, the complexity of this module is $O(|E| \cdot |V|^2 \cdot G_{\text{SIM}} \cdot N_{\text{SIM}})$.

Algorithm 6: Quantum circuit transformation based on Monte Carlo tree search

input : An architecture graph AG, a logical circuit LC, and an initial mapping τ_{ini}.
output : A physical circuit satisfying the connectivity constraints in AG.

```
PC ← the circuit consisting of all executable gates in LC under τ_ini;
LC ← LC with gates in PC deleted;
s ← (τ_ini, PC, LC);
VAL(s), VISIT(s) ← 0;
T ← a search tree with a single (root) node s;
while LC(s) ≠ ∅ do
    do
        s ← Select(T);
        Expand(T, s);
        Simulate(T, s);
        Backpropagate(T, s);
    for N_BP times;
    Decide(T);
    // T is updated in Decide and Expand modules
    s ← root(T);
return PC(s)
```

Finally, note that in the worst case, all gates from C are executed by the fallback mechanism which is invoked after every $|V|$ iterations. Hence, the *Sel-Exp-Sim-BP* modules will be run for at most $|C| \cdot |V| \cdot N_{\text{BP}}$ times, and the overall time complexity of our algorithm is

$$O\left(|C|\cdot|V|\cdot N_{\text{BP}}\cdot|E|\cdot[N_{\text{BP}}\cdot|V| + |C| + |V|^2\cdot G_{\text{SIM}}\cdot N_{\text{SIM}}]\right),$$

or $O(|C|\cdot|V|\cdot|E|\cdot(|C| + |V|^2))$ when the parameters are regarded as constants.

4 DEPTH OPTIMISATION

QPUs in the NISQ era also suffer from limited coherence time, meaning that the depth of the output physical circuit is also an important criterion for optimising the circuit transformation process. In this section, we propose MCTS-Depth, which is adapted from the MCTS-Size algorithm presented in the previous section by introducing two minor changes, to further reduce the depth of the output circuit.

Recall that in MCTS-Size we have removed all single-qubit gates because they have no effect when the QCT objective is to minimise the number of inserted SWAP gates. However, as shown in Example 4, this is not the case as far as circuit depth is concerned. In this paper, we adopt a simple strategy to deal with these gates: whenever an executable CNOT g is removed from the logical circuit and added to the physical circuit in *Expansion*, all single-qubit gates after g and before any other CNOT that directly depends on g will be greedily added to the physical circuit.

Example 4. *Suppose the quantum (logical) circuit to be transformed by MCTS-Depth is specified as in Fig. 7(a). Assume that the target QPU is IBM Q20 and we take the initial mapping to be the naive one. As the CNOT g_0 is directly executable, g_0 and the single-qubit gate g_1 are immediately added to the physical circuit. To make g_2 executable, we can insert a SWAP either between physical qubits v_0 and v_1 (cf. Fig. 7(b)) or between v_1 and v_2 (cf. Fig. 7(c)). The depth overhead brought by adding* $\text{SWAP}(v_0, v_1)$ *and* $\text{SWAP}(v_1, v_2)$ *are, respectively, 1 and 3, after decomposing each SWAP into 3 CNOTs.*

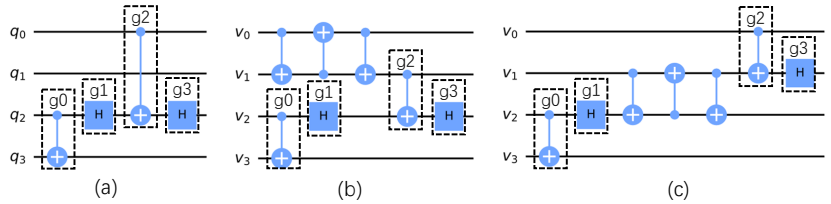


Fig. 7. An input quantum circuit (a) and two functionally equivalent physical circuits, (b) and (c), that are executable on IBM Q20 with the naive initial mapping.

As shown in Example 4, different SWAP gates may incur different depth overheads. Let $\mathbf{s} = (\tau, PC, LC)$ be the current state and $\mathbf{s}' = (\tau', PC', LC')$ the child state corresponding to some SWAP. As each SWAP is implemented as three consecutive CNOTs, the depth overhead, written $\mathrm{I}_{\mathbf{s}'}$, is an integer between 0 and 3. That is, a SWAP may incur 0, 1, 2, or 3 extra layers. The precise value of $\mathrm{I}_{\mathbf{s}'}$ is calculated as the depth difference of PC' and PC, where the executed single-qubit gates are properly added back. Again, we note that, in practical implementation, the depth information is stored in the form of a tuple with $|V|$ elements all initiated in 0. When a gate, say $\mathrm{CNOT}(v_i, v_j)$, is added to the circuit, the tuple will be updated by changing both of its i- and j-th elements to $x+1$, where x is the larger original value between those two elements. In addition, the depth of the corresponding circuit is exactly the maximum value in the tuple.

Apparently, we prefer SWAPs with smaller $\mathrm{I}_{\mathbf{s}'}$. This motivates us to replace the discount factor γ in Eq. (6) for MCTS-Size with $\gamma^{\mathrm{I}_{\mathbf{s}'}}$ and obtain the following value-update rule for MCTS-Depth:

$$\textsc{val}(\mathbf{s}) \leftarrow \max\{\textsc{val}(\mathbf{s}),\ \gamma^{\mathrm{I}_{\mathbf{s}'}} \cdot [\textsc{rwd}(\mathbf{s}, \mathbf{s}') + \textsc{val}(\mathbf{s}')]\} \tag{7}$$

Another modification is applied to the definition of the initial long-term value of a state $\mathbf{s}$ in the simulation process, given in Eq. (3), where it uses N, the minimal number of SWAP gates required during all N_{SIM} simulations, as an important index. Apparently, in order to reduce depth, it is more meaningful to replace N with M, the minimal depth overhead of all N_{SIM} simulations. That is, in MCTS-Depth, Eq. (3) is replaced with

$$\textsc{val}(\mathbf{s}) = \gamma^{M/2} \cdot G_{\mathrm{SIM}} \tag{8}$$

and the second last line of Alg. 3 is replaced with

$$\textsc{val}(\mathbf{s}) \leftarrow \gamma^{M/2} \cdot G_{\mathrm{SIM}},$$

where for the same reason as that in Eq. (3) the exponent is taken as $M/2$ instead of M.

It is clear that these modifications do not affect the complexity analysis given in Sec. 3.3.

5 INCORPORATING REMOTE CNOT

In above, we have seen how a circuit can be transformed by inserting SWAPs. This is sometimes not desirable as the mapping will change with the inserted SWAPs (cf. Example 5 below). Several transformers (including the current version of t|ket⟩) suggest using remote CNOT operations (also known as bridge gates) to execute CNOT gates whose two qubits in the current mapping are not neighbours (i.e., remote). In this section, we show how remote CNOTs can be incorporated into our MCTS-based algorithms.

Let τ be the current mapping and $g = \mathrm{CNOT}(q, q')$. If the two physical qubits $\tau(q)$ and $\tau(q')$ are not neighbours in the target AG, we may replace g with a sequence of CNOT gates, written $\mathcal{R}_\tau(g)$,

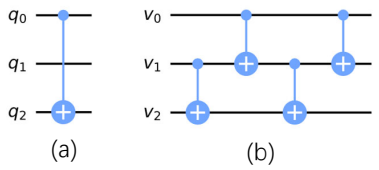


Fig. 8. Take AG to be that of IBM Q20 and the initial mapping τ be naive. (a) A logical circuit with only one gate CNOT$(q_0, q_)$. (b) A remote CNOT implementation of CNOT(q_0, q_2).

which are executable and functionally equivalent to g. Fig. 8(b) shows the special case when the distance of $\tau(q)$ and $\tau(q')$ in AG is 2. More general construction can be found in [Nash et al. 2020].

Example 5. *Consider the circuit shown in Fig. 4 (left). Except g_0, every CNOT in the circuit can be executed by the naive mapping. If only SWAPs are allowed, we need to insert a SWAP to execute g_0. As a consequence, the mapping is changed and at least one of the other CNOTs are not executable and we need to insert another SWAP, which results in a size overhead of at least six! However, g_0 can be executed by implementing it as a remote CNOT depicted in Fig. 8(b) and, after that, the other CNOTs can be immediately executed, which gives an overhead of three!*

To extend our MCTS algorithms with remote CNOT, we need only modify *Expansion* and *Backpropagation*. Starting from a state/node $\mathbf{s} = (\tau, PC, LC)$, besides all relevant SWAPs as used in Alg. 2, we also consider all $g = \text{CNOT}(q, q')$ if g is in the first layer of LC and the distance of $\tau(q)$ and $\tau(q')$ in AG is between 2 and some fixed integer d. We then may replace g with a CNOT sequence $\mathcal{R}_\tau(g)$ along the shortest path connecting $\tau(q)$ and $\tau(q')$ if desirable. As remote CNOTs and SWAPs may incur different size and depth overheads, a modified value-update rule like Eq. (7) is used during *Backpropagation* if $\mathbf{s}' = (\tau', PC', LC')$ is the child node of $\mathbf{s}$ derived by a remote CNOT implementation of g. More precisely, for MCTS-Size, $\mathrm{I}_{\mathbf{s}'}$ is defined as $(|\mathcal{R}_{\tau'}(g)| - 1)/3$. The intuition behind this is that the sub-circuit $\mathcal{R}_{\tau'}(g)$ we used to replace g brings a size overhead of $|\mathcal{R}_{\tau'}(g)| - 1$ in terms of CNOTs, which translates to $(|\mathcal{R}_{\tau'}(g)| - 1)/3$ in terms of SWAPs. For MCTS-Depth, $\mathrm{I}_{\mathbf{s}'}$ is set as the depth overhead brought by adding gates in $\mathcal{R}_{\tau'}(g)$ to the physical circuit in state $\mathbf{s}'$.

To conclude this section, we point out that the remote CNOT approach does not always give better result than the SWAP-based approach. This is because inserting SWAPs changes the mapping, which is sometimes desirable as the new mapping may execute more later CNOTs. Consider again the circuit in Fig. 4 (left). If the CNOT gate g_3 were applied on q_0 and q_2, then inserting SWAP(v_0, v_1) (i.e. three CNOTs) suffices to solve all gates in the circuit, while remote implementation of both g_0 and g_3 would introduce an overhead of six CNOTs. In practice, however, it might not be easy to decide which approach is preferable. Thus we provide both of them as possible choices: The user may decide if s/he wants to use remote CNOT together with SWAPs as legal MCTS actions when calling our QCT algorithms. In Sec. 6 we will evaluate its impact for two AGs.

6 IMPLEMENTATION AND EVALUATION

To evaluate our approach, we compare it with five state-of-the-art algorithms (cf. Sec. 6.1). As the choice of initial mappings may sometimes influence the performance of QCT algorithms, to make a fair comparison, we always take the same initial mappings in their original design if available. We use Python as our main programming language and IBM Qiskit [et al. 2019] as the auxiliary environment to implement our algorithms. For efficiency, the *Simulation* module is implemented in C++. All experimental results reported here are obtained by choosing the best one from five trials.

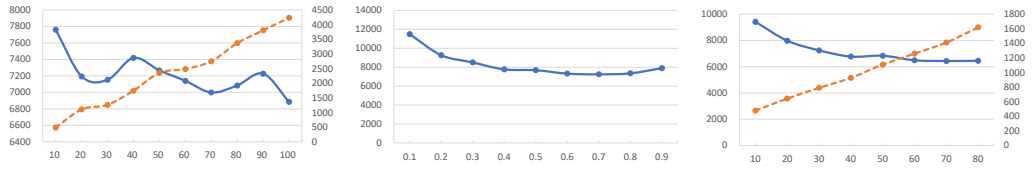


Fig. 9. Evaluation of the performance of MCTS-Size on IBM Q20 for different parameter settings: (a) N_{BP}, (b) γ, and (c) G_{SIM}, where the vertical axes on the left and right in each sub-figure represent the aggregated number of added CNOTs in output circuits (blue lines) and running time (seconds, orange dashed lines), respectively. All data is aggregated from the small benchmark set $\mathcal{B}_{11}$ and initial mappings are the same as those used in SAHS [Zhou et al. 2020b].

Note that we only provide summarised results here. Readers are referred to the GitHub repository[4] for detailed empirical results together with source code of our algorithms and benchmarks used in the experiments.

6.1 Benchmarks and Compared State-of-the-Art QCT Algorithms

In our evaluation, we selected a set of 114 benchmark circuits, with a sum of 554,497 gates (including 248,553 CNOTs) and a sum of 303,469 depths, which, taken from [Zulehner et al. 2018], were published by IBM as part of the 2018 QISKit Developer Challenge[5] and have been widely used in evaluating circuit transformation algorithms by, e.g., [Cowtan et al. 2019; Li et al. 2019, 2021; Zhou et al. 2020b]. In the following, we write $\mathcal{B}_{114}$ for this benchmark set.

Although circuits in $\mathcal{B}_{114}$ are widely used, not all of them are directly relevant to quantum computing. To evaluate the proposed algorithms on 'real' quantum circuits, we also extracted a set of 173 quantum circuits, written $\mathcal{B}_{real}$, from the quantum algorithm library in Qiskit. Circuits in $\mathcal{B}_{real}$ have a sum of 603,654 gates (including 260,589 CNOTs) and a sum of 413,734 depths.

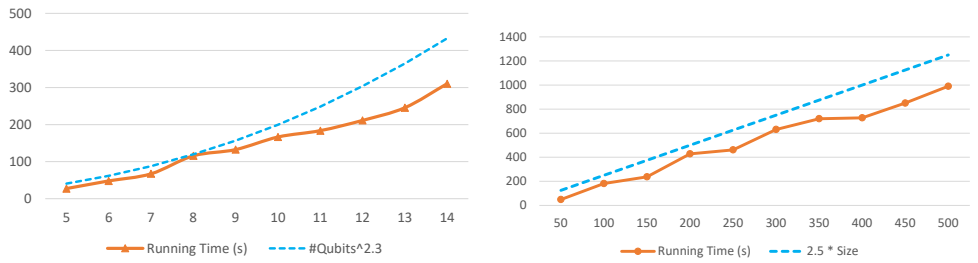
The QCT algorithms to be compared with our MCTS algorithms include t|ket⟩ (version 0.17.0[6]) [Cowtan et al. 2019], SAHS [Zhou et al. 2020b], FiDLS [Li et al. 2021], Qiskit (version 0.33.0) [et al. 2019] and SABRE [Li et al. 2019], which are state-of-the-art algorithms for quantum circuit transformation. While we didn't include Cirq[7] in our comparison, it was found in [Tan and Cong 2021] that Cirq is less efficient than t|ket⟩ and Qiskit. For fair and pure comparison of the routing abilities, we also disabled the postmapping optimisation of t|ket⟩.

6.2 Parameter Determination

Our MCTS-based QCT algorithms have a couple of parameters to be determined before actual running:

- N_{BP} (repeated times for the *Sel-Exp-Sim-BP* modules before each *Decision*),
- c (the exploration parameter used in Eq. (2)),
- G_{SIM} (the size of sub-circuit used in simulation),
- N_{SIM} (the number of simulations),
- γ (the discount ratio), and
- d (the maximum distance allowed for remote CNOT).

[4]https://github.com/BensonZhou1991/Circuit-Transformation-via-Monte-Carlo-Tree-Search
[5]https://www.ibm.com/blogs/research/2018/08/winners-qiskit-developer-challenge/
[6]https://cqcl.github.io/pytket/build/html/index.html
[7]https://quantumai.google/cirq



(a) (b)

Fig. 10. Evaluations of the performance of MCTS-Size on a hypothetical AG Grid 4×5 by comparing the average running time (seconds, orange line) with (a): number of logical qubits, and (b): number of CNOTs of the input circuits, where each circuit in (a) contains 500 CNOTs and each circuit in (b) has 20 qubits, and each data point denotes the average value of 10 randomly generated quantum circuits with naive initial mappings.

To help determine these parameters for IBM Q20, we selected a small subset of 11 typical circuits from $\mathcal{B}_{114}$ each of which involves 10-15 qubits and has 1000-6000 CNOTs. This benchmark set, written as $\mathcal{B}_{11}$, contains in total 26,676 CNOTs. Fig. 9 depicts the dependency of the size of the final physical circuits (left vertical axis) and the running time (right vertical axis) on different parameter settings. One may note that the performance in Fig. 9(a) does not improve steadily with the increasing of N_{BP}. This is perhaps due to that those data are derived by running our algorithm for five times and keeping only the best results.

To get a good balance between performance and running time, we empirically set

$$N_{\text{BP}} = c = 20,\ G_{\text{SIM}} = 30,\ N_{\text{SIM}} = 500, \text{ and } \gamma = 0.7.$$

We adopt the same parameter setting for the other proposed algorithms, and set $d = 2$ for MCTS-Size and MCTS-Depth.

6.3 Running Time and Search Depth

We have shown in Sec. 3.3 that our algorithm runs in time polynomial in all relevant parameters. To further demonstrate the running time in practice, we randomly generate two sets of 10 quantum circuits. In one set, each circuit has 500 CNOTs, and the number of logical qubits ranges from 5 to 14. In the other, each circuit has 20 qubits, and the number of CNOTs ranges from 50 to 500. We transform all these circuits via MCTS-Size on a hypothetical AG Grid 4×5, and record the average running time for each circuit set. As shown in Fig. 10, the real time cost is roughly the 2.3th power in the number of qubits and linear (with slope being about 2.5) in the number of CNOTs, indicating that our algorithm is practically scalable.

For MCTS-Size, we also record the search depth for each use of the *Selection* module and calculate the minimum, average, and maximum depth before each *Decision* process. In SAHS [Zhou et al. 2020b], the search depth is an adjustable parameter determining how deeply the heuristic evaluation process will look into. As shown in Fig. 12, the maximum search depth of MCTS-Size can easily exceed that of SAHS, which is set to 2 by default in its original implementation. Actually, in most of the time it is more than 3, meaning that our algorithm has better ability of exploring the unknown state.

Note that it is claimed in [Zhou et al. 2020b] that the size of output physical circuits can be further decreased by increasing the search depth, with the cost of more time consumption. Fig. 11 depicts a comparison of the output circuit size as well as the running time of MCTS-Size and SAHS on the example circuit 'misex1_241' with 4,813 gates, where the search depth of SAHS varies from

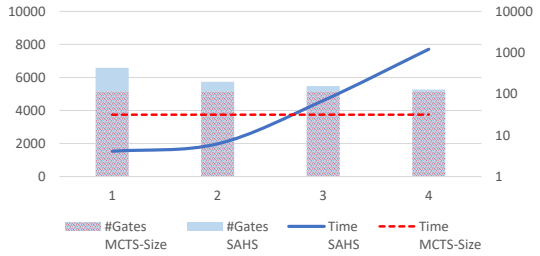


Fig. 11. The benefit brought by increasing the search depth in SAHS [Zhou et al. 2020b] for circuit 'misex1_241'. The horizontal axis and vertical axes on the left and right represent search depth, number of gates in the output circuit and running time (seconds), respectively. Note that here the search depth is only applied to SAHS and it is not adjustable in MCTS.

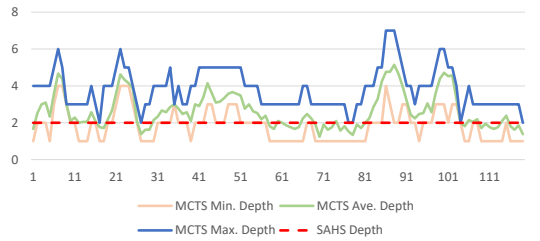


Fig. 12. Search depth (for *Selection* before each *Decision*) of MCTS-Size for circuit 'misex1_241'. The horizontal and vertical axis represent rounds for *Decision* and search depth, respectively. Note that here the search depth of SAHS is fixed as 2.

1 to 4. It shows that MCTS-Size outperforms SAHS in the output circuit size even when the search depth of SAHS is set to 4. However, in this case, the running time of SAHS is over 20 minutes, while MCTS-Size only needs 31 seconds.

6.4 Evaluation of the Performance of MCTS-Size

Now we compare MCTS-Size with SAHS [Zhou et al. 2020b], FiDLS [Li et al. 2021], and t|ket⟩[Cowtan et al. 2019] on IBM Q20 over the 114 benchmark circuits in $\mathcal{B}_{114}$. The results are summarised in Table 1, where columns 2 & 3 represent aggregated numbers of added CNOTs (each SWAP is decomposed into 3 CNOTs) obtained from other methods and MCTS-Size when using their initial mappings, respectively. Besides, the 'improvement' is defined as $(n_{comp} - n_{ours})/n_{comp}$, with n_{comp} and n_{ours} being the total numbers of CNOT gates added by the compared algorithm and ours, respectively, in transforming the 114 circuits. A similar definition for 'Improvement' is used in the rest of the paper.

From Table 1, we can see that MCTS-Size achieves a conspicuous improvement of 36.68% on average when compared with SAHS (by using the same initial mappings as SAHS). In [Li et al. 2021], two techniques for initial mappings, topgraph (topg.) and weighted graph (wgt.), are proposed with FiDLS. Our algorithm has a consistent improvement, 30.39% for the topgraph initial mappings and 28.89% for the weighted graph ones. As the initial mappings of t|ket⟩ are not directly available, we use naive mappings as the initial mappings in the experiments. For fair and pure comparison of the routing abilities, we also disabled the postmapping optimisation of t|ket⟩. As can be seen from the last row of Table 1, the gate overhead of t|ket⟩ is above 3 times of ours (also with naive initial

Table 1. Comparison of MCTS-Size with state-of-the-art algorithms using their initial mappings on the large benchmark set $\mathcal{B}_{114}$, where Columns 2 & 3 represent aggregated numbers of added CNOTs obtained from the algorithm under comparison and MCTS-Size when using their initial mappings, respectively.

Compared Algorithm	gate overhead	gate overhead MCTS-Size	Improvement
SAHS	116487	73758	36.68%
Topg. FiDLS	107406	74763	30.39%
Wgt. FiDLS	105645	75126	28.89%
t|ket⟩	238170	*77544*	59.24%

mappings). It is worth noting that, on this benchmark set, MCTS-Size still performs well even with naive initial mappings; the overhead compared with its best result is only 5.13% (77,544 vs. 73,758).

For each compared algorithm, MCTS-Size performs at least as good as the compared algorithm on most circuits. Particularly, the performance of MCTS is consistently better than or the same as SAHS in all circuits. Furthermore, an improvement less than 5% occurs only when the circuit is small and can be transformed by the compared algorithm with no more than 30 ancillary SWAP (or, equivalently, 90 CNOT) gates. Hence, it can be concluded that the performance of the MCTS algorithm is stable in various input circuits and initial mappings.

6.5 Evaluation of the Performance of MCTS-Depth

Now we compare MCTS-Depth with MCTS-Size, t|ket⟩, Qiskit, and SABRE on the two benchmark sets in terms of the total depth overhead. As the initial mappings used by t|ket⟩ are not directly available, we adopt the naive mapping as the initial mapping for all these algorithms. It is surprise that on IBM Q20 the initial mappings selected by t|ket⟩ are, on average, not better than the naive mappings (238,170 vs. 237,471 added CNOTs and 238,051 vs. 236,312 added depths).

To evaluate the impact of the underlying topology of the architecture structure, we also run the experiments on a hypothetical grid-like QPU, called Grid 4×5, which has fewer edges (31 vs. 43) than IBM Q20. We also empirically evaluated the impact of remote CNOT gates in our MCTS algorithms and t|ket⟩. As can be seen from Table 2, MCTS-Depth is able to improve the depth performance steadily for both tested AGs when compared to MCTS-Size, which confirms the utility of our modifications in Sec. 4. We note that the improvements of Alg. X against Alg. Y in Table 2 are calculated as $(n_Y - n_X)/n_Y$ with n_Y (n_X, resp.) being either the sum of the added CNOTs or the sum of the added depths by Alg. Y (Alg. X, resp.).

6.5.1 Impact of the Topology of the Architecture Structure. When compared with t|ket⟩, for IBM Q20, both MCTS-Size and MCTS-Depth have a great advantage, with about 75% and 84% improvement; for Grid 4×5, however, their advantages are not so remarkable. The main reason is perhaps due to the fact that IBM Q20 supports more qubit connections (and has a smaller diameter) than Grid 4×5, which enables the MCTS-based algorithms to find a good solution without going much deeper for IBM Q20. Another reason may be that t|ket⟩ performs really well on grid-like AGs.

6.5.2 Impact of Remote CNOT. Next, we discuss the improvement brought by introducing remote CNOT. We empirically evaluated the impact of remote CNOT in our MCTS-based algorithms and t|ket⟩. The results are summarised in Table 2, where for an algorithm A, A+r denotes the algorithm with remote CNOT enabled. For IBM Q20, the improvements brought by introducing remote CNOT are almost negligible, and sometimes even degraded. For Grid 4×5, however, the improvements are quite significant. Compared to t|ket⟩, our algorithms have gained an improvement of 28% in

size (MCTS-Size+r) and an improvement of 42% in depth (MCTS-Depth+r), while t|ket⟩+r has a 4% improvement in size and a 14% improvement in depth.

6.5.3 Evaluation on Real Benchmark Circuits. We also evaluated these algorithms on IBM Q20 and the real benchmark circuits in $\mathcal{B}_{real}$. The results are summarised in Table 3, where we can see that (i) the impact of remote CNOT is significantly negative; (ii) the improvements of both MCTS algorithms against t|ket⟩ are still above 50%.

In addition, we also compared MCTS-Depth with the new algorithm proposed in [Zhang et al. 2021], which aims at minimising the depth of the output circuit. Evaluation on IBM Q20 and 26 benchmark circuits used in [Zhang et al. 2021] shows that MCTS-Depth performs consistently better and can reduce on average the depth overhead by 67% when compared with the algorithm in [Zhang et al. 2021].

Table 2. Comparison of the proposed MCTS algorithms with t|ket⟩, Qiskit and SABRE on the benchmark set $\mathcal{B}_{114}$ with naive initial mappings, where A+r denotes the algorithm A with remote CNOT enabled. Columns 5 & 6 represent, respectively, the size and depth improvement compared to t|ket⟩.

AG	Method	CNOT Added	Depth Added	Size Imp.	Depth Imp.
IBM Q20	t|ket⟩	237471	236312	-	-
	t|ket⟩+r	273420	238661	-15.14%	-0.99%
	Qiskit	575400	335133	-142.30%	-41.82%
	SABRE	406365	382442	-71.12%	-61.84%
	MCTS-Size	79743	58602	66.42%	75.20%
	MCTS-Size+r	79275	59857	66.62%	74.67%
	MCTS-Depth	151530	37794	36.19%	84.01%
	MCTS-Depth+r	147972	37138	37.69%	84.28%
Grid 4×5	t|ket⟩	383409	368109	-	-
	t|ket⟩+r	367815	314148	4.07%	14.66%
	Qiskit	641616	456261	-67.35%	-23.95%
	SABRE	596679	517354	-55.62%	-40.54%
	MCTS-Size	356091	359240	7.13%	2.41%
	MCTS-Size+r	275274	263811	28.20%	28.33%
	MCTS-Depth	514311	292050	-34.14%	20.66%
	MCTS-Depth+r	392349	211421	-2.33%	42.57%

6.6 Evaluation on Large Architectural Graphs

IBM Rochester and Google Sycamore, both having 53 qubits (cf. Fig. 5), are two state-of-the-art QPUs. It is natural to ask if our MCTS-based algorithms still have superior performance on these QPUs. Empirical evaluations show that the MCTS algorithms as presented above do not perform significantly better than t|ket⟩. However, if we replace the original simulation module as described in Alg. 3 with a simple deterministic heuristic strategy, then the MCTS algorithms also demonstrate superior performance on Sycamore and Rochester.

6.6.1 A Deterministic Strategy for Simulation. Recall that the goal of Simulation is to obtain a simulated score as the initial long-term value for a state. We introduce a deterministic heuristic strategy to replace the original Simulation module described in Alg. 3. Specifically, when simulation is requested in a state $\mathbf{s}$ with a parent state, say $\mathbf{s}'$, we extract the first L_{SIM} (a predefined parameter,

Table 3. Comparison of the proposed MCTS algorithms with t|ket⟩, Qiskit, and SABRE over quantum circuits in the benchmark set $\mathcal{B}_{real}$ on IBM Q20.

AG	Method	CNOT Added	Depth Added	Size Imp.	Depth Imp.
IBM Q20	t|ket⟩	101304	91730	-	-
	t|ket⟩+r	156774	121734	-54.76%	-32.71%
	Qiskit	799818	514119	-93.64%	-9.44%
	SABRE	802737	597933	-96.52%	-100.81%
	MCTS-Size	41010	41826	59.52%	54.40%
	MCTS-Size+r	38607	49012	61.89%	46.57%
	MCTS-Depth	64992	27237	35.84%	70.31%
	MCTS-Depth+r	60333	31300	40.44%	65.88%

Table 4. Comparison of the proposed MCTS algorithms with t|ket⟩, Qiskit and SABRE on two benchmark sets $\mathcal{B}_{114}$ and $\mathcal{B}_{ran}$ with naive initial mappings on IBM Rochester.

Benchmark	Method	CNOT Added	Depth Added	Size Imp.	Depth Imp.
$\mathcal{B}_{114}$	t|ket⟩	575418	560479	-	-
	t|ket⟩+r	495687	429980	13.86%	23.28%
	Qiskit	882606	590607	-53.39%	-5.38%
	SABRE	844923	681685	-46.84%	-21.63%
	MCTS-Size	563643	560205	2.05%	0.05%
	MCTS-Size+r	970678	375384	16.04%	33.02%
	MCTS-Depth	416181	467698	-37.13%	16.55%
	MCTS-Depth+r	513771	310220	10.71%	44.65%
$\mathcal{B}_{ran}$	t|ket⟩	29526	7321	-	-
	t|ket⟩+r	29301	6905	0.76%	5.68%
	Qiskit	36296	5551	-16.16%	24.18%
	SABRE	30921	5456	-4.7%	25.47%
	MCTS-Size	26427	7050	10.50%	3.70%
	MCTS-Size+r	26097	7167	11.61%	2.10%
	MCTS-Depth	30378	3445	-2.89%	52.94%
	MCTS-Depth+r	29481	3491	0.15%	52.32%

fixed as 4 in our implementation) layers of gates in the logical circuit in the parent state $\mathbf{s}'$, and then calculate the initial long-term value of $\mathbf{s}$ as

$$\textsc{val}(\mathbf{s}) = \sum_{i=0}^{L_{\text{SIM}}} \left[\left(\sum_{g \in \mathcal{L}_i(LC(\mathbf{s}'))} \textsc{scost}(g, \tau') - \sum_{g \in \mathcal{L}_i(LC(\mathbf{s}'))} \textsc{scost}(g, \tau) \right) \cdot \gamma^{i+1} \right] \tag{9}$$

where the notations are identical to that in Eq. (4). This heuristic first calculates, for each layer in the extracted logical circuit, its score defined as the total distance reduction among all CNOTs brought by the inserted SWAP corresponding to $\mathbf{s}$ and then aggregates these scores with a discount factor γ (predefined and set as 0.7 in the experiments).

Table 5. Comparison of the proposed MCTS algorithms with t|ket⟩, Qiskit, and SABRE on two benchmark sets $\mathcal{B}_{114}$ and $\mathcal{B}_{ran}$ with naive initial mappings on Google Sycamore.

Benchmark	Method	CNOT Added	Depth Added	Size Imp.	Depth Imp.
$\mathcal{B}_{114}$	t|ket⟩	391923	378233	-	-
	t|ket⟩+r	370068	358354	5.58%	5.26%
	Qiskit	625014	446666	-59.47%	-18.09%
	SABRE	582933	509737	-48.74%	-34.77%
	MCTS-Size	365823	670279	6.66%	3.02%
	MCTS-Size+r	291732	366810	21.17%	26.24%
	MCTS-Depth	471309	304140	-20.26%	19.59%
	MCTS-Depth+r	365940	213782	6.63%	43.48%
$\mathcal{B}_{ran}$	t|ket⟩	15837	4146	-	-
	t|ket⟩+r	15813	4072	0.15%	1.78%
	Qiskit	17967	4062	-13.45%	2.03%
	SABRE	18030	3877	-13.85%	6.49%
	MCTS-Size	14328	4585	9.53%	-10.59%
	MCTS-Size+r	14148	4558	10.66%	-9.94%
	MCTS-Depth	16155	2280	-2.01%	45.01%
	MCTS-Depth+r	15765	2254	0.45%	45.63%

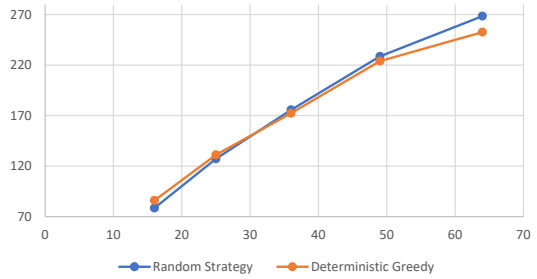


Fig. 13. The average numbers of added ancillary CNOT gates derived by two native QCT algorithms (vertical axis) vs. numbers of qubits in tested AGs.

6.6.2 Evaluation of the MCTS Algorithms with the New Simulation Module. To demonstrate the efficacy of our MCTS-based algorithms on QPUs with large qubit numbers, experiments are also done on IBM Rochester and Google Sycamore. We then make comparisons with t|ket⟩, Qiskit, and SABRE. The results are summarised in Table 4, for IBM Rochester, and Table 5, for Sycamore, from which we can see that our MCTS-based algorithms still consistently outperform those industrial-level algorithms, especially when the target is circuit depth.

To explain why the deterministic strategy performs better than the random one on large AGs, a series of experiments are devised by introducing two *kernel* QCT algorithms are designed: The first one utilises the random simulation strategy directly to transform the input circuit while the second adopts the deterministic heuristic in Eq. (9). Then we compare the performance of the two kernel algorithms on five hypothetical AGs: Grid 4×4, 5×5, 6×6, 7×7 and 8×8. For each AG, we generate 20 logical circuits each consisting of 30 randomly placed CNOT gates and then transform them to physical ones by the two native algorithms, respectively. For each AG, we record the average

number of added ancillary CNOT gates among its 20 transformed physical circuits. As depicted in Fig. 13, the algorithm with the deterministic greedy strategy performs better than that with the random strategy on AGs with more than 36 qubits, indicating the rationality for adopting the new Simulation strategy for large AGs.

7 CONCLUSION

In this paper, an MCTS framework is proposed for the quantum circuit transformation problem, which aims at minimising either the size or the depth overhead to transform an ideal logical circuit to a physical one executable on a QPU with connectivity constraints. For this purpose, a scoring mechanism (cf. Eq.s 6 and 7) is designed which takes into account both the short-term reward of introducing a SWAP and a long-term value obtained by random simulations. Furthermore, when backpropagating rewards collected by states to their ancestors, a discount factor is introduced to guide the algorithm towards a cheapest path to a goal state. The MCTS-based algorithms, viz., MCTS-Size and MCTS-Depth, run in polynomial time with respect to all relevant parameters. With six parameters, they are very flexible in meeting different optimisation objectives, can stop whenever a preassigned resource limit is reached, and search much deeper than existing algorithms. Empirical results on extensive realistic circuits on IBM Q20, Rochester, and Google Sycamore confirmed that MCTS-Size (MCTS-Depth, resp.) can reduce, on average, the CNOT (depth, resp.) overhead by as high as 75% (84%, resp.) when compared with t|ket⟩, an industrial level product.

When designing QCT algorithms, we assume that logical circuits are transformed into executable physical circuits by inserting ancillary CNOTs stepwise. Besides CNOTs, several other tools may be considered in our MCTS-based algorithms. For example, the algorithm in [Gheorghiu et al. 2020] tries to 're-compile' part of or the whole logical circuit to make the CNOTs in the newly compiled one satisfying the connectivity constraints; gate commutation rules are used in [Itoko et al. 2019] to further simplify the transformed circuit; and quantum teleportation is introduced in [Hillmich et al. 2021] as a complementary method for the transforming process. Combination of these approaches may generate functional equivalent physical circuits with lower depth.

Recently, Tan and Cui [Tan and Cong 2021] proposed a random circuit library QUEKO for evaluating the optimality of quantum circuit transformers, which contains circuits with known optimal depth overhead. We evaluated MCTS-Depth on QUEKO and the results show a total score of 2.88 (meaning the ratio of the total depths of the output and input circuits), which is, though better than that of t|ket⟩ (3.76), is still too far from the optimal ratio, viz. 1. This is partially due to that we used naive initial mappings, instead of the optimal mappings, e.g., those found by subgraph isomorphism [Li et al. 2021]. On the other hand, it suggests that there is still much room to improve the implementation of our algorithms. This is the first problem we intend to attack for future studies. Second, parameters presented in our algorithms are QPU-dependent, and a careful study of their correlation may provide a better insight on how to choose them in practice. Third, more objectives, e.g., fidelity and error rate, should be included in evaluating the quality of output physical circuits. Last but not least, it is promising to develop a parallelised implementation of our MCTS-based algorithms in a multi-thread way, where hundreds or thousands computational processes can run in parallel and share the same memory. The success implementation of this parallelised MCTS framework could help us get even better results (by going deeper) more quickly.

8 ACKNOWLEDGEMENTS

We thank the reviewers for their very helpful comments and suggestions. This work was supported by the National Key R&D Program of China (Grant No. 2018YFA0306704), the Australian Research Council (Grant No.s DP220102059, DP180100691), and the National Science Foundation of China (CN) (Grant No.s 61871111, 12071271).

REFERENCES

Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

Kyle EC Booth, Minh Do, J Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. 2018. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

Guillaume M JB Chaslot, Mark HM Winands, H JAAP VAN DEN HERIK, Jos WHM Uiterwijk, and Bruno Bouzy. 2008. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation* 4, 03 (2008), 343–357.

Donny Cheung, Dmitri Maslov, and Simone Severini. 2007. Translation techniques between quantum circuit architectures. In *Workshop on Quantum Information Processing*.

Andrew M Childs, Eddie Schoute, and Cem M Unsal. 2019. Circuit Transformations for Quantum Architectures. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*.

Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the Qubit Routing Problem. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*.

Alexandre AA de Almeida, Gerhard W Dueck, and Alexandre CR da Silva. 2019. Finding optimal qubit permutations for IBM's quantum computer architectures. In *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*. 1–6.

Gadi Aleksandrowicz et al. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

Will Finigan, Michael Cubeddu, Thomas Lively, Johannes Flick, and Prineha Narang. 2018. Qubit allocation for noisy intermediate-scale quantum computers. *arXiv preprint arXiv:1810.08291* (2018).

Vlad Gheorghiu, Sarah Meng Li, Michele Mosca, and Priyanka Mukhopadhyay. 2020. Reducing the CNOT count for Clifford+T circuits on NISQ architectures. *arXiv preprint arXiv:2011.12191* (2020).

Thomas Häner, Damian S Steiger, Krysta Svore, and Matthias Troyer. 2018. A software methodology for compiling quantum programs. *Quantum Science and Technology* 3, 2 (2018), 020501.

Stefan Hillmich, Alwin Zulehner, and Robert Wille. 2021. Exploiting Quantum Teleportation in Quantum Circuit Mapping. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 792–797.

Toshinari Itoko, Rudy Raymond, Takashi Imamichi, Atsushi Matsuo, and Andrew W Cross. 2019. Quantum circuit compilers using gate commutation rules. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 191–196.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *15th European Conference on Machine Learning*. Springer, 282–293.

Janusz Kusyk, Samah M. Saeed, and Muharrem Umit Uyar. 2021. Survey on Quantum Circuit Compilation for Noisy Intermediate-Scale Quantum Computers: Artificial Intelligence to Heuristics. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–16. https://doi.org/10.1109/TQE.2021.3068355

Lingling Lao, Daniel M Manzano, Hans van Someren, Imran Ashraf, and Carmen G Almudever. 2019. Mapping of quantum circuits onto NISQ superconducting processors. *Quantum* 2 (2019), 3.

Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1001–1014.

Sanjiang Li, Xiangzhen Zhou, and Yuan Feng. 2021. Qubit Mapping Based on Subgraph Isomorphism and Filtered Depth-Limited Search. *IEEE Trans. Computers* 70, 11 (2021), 1777–1788. https://doi.org/10.1109/TC.2020.3023247

Aaron Lye, Robert Wille, and Rolf Drechsler. 2015. Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits. In *The 20th Asia and South Pacific Design Automation Conference*. IEEE, 178–183.

Dmitri Maslov, Sean M. Falconer, and Michele Mosca. 2007. Quantum Circuit Placement: Optimizing Qubit-to-qubit Interactions through Mapping Quantum Circuits into a Physical Experiment. In *Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007*. IEEE, 962–965. https://doi.org/10.1145/1278480.1278717

Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1015–1029.

Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. 2020. Quantum circuit optimizations for NISQ architectures. *Quantum Science and Technology* 5, 2 (2020), 025010.

Michael A Nielsen and Isaac Chuang. 2002. *Quantum computation and quantum information*. Cambridge University Press.

Shin Nishio, Yulu Pan, Takahiko Satoh, Hideharu Amano, and Rodney Van Meter. 2020. Extracting Success from IBM's 20-Qubit Machines Using Error-Aware Compilation. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 16, 3 (2020), 1–25.

Angelo Oddi and Riccardo Rasconi. 2018. Greedy randomized search for scalable compilation of quantum circuits. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 446–461.

Alexandru Paler. 2019. On the Influence of Initial Qubit Placement During NISQ Circuit Compilation. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 207–217.

Matteo G Pozzi, Steven J Herbert, Akash Sengupta, and Robert D Mullins. 2020. Using reinforcement learning to perform qubit routing in quantum compilers. *arXiv preprint arXiv:2007.15957* (2020).

Riccardo Rasconi and Angelo Oddi. 2019. An innovative genetic algorithm for the quantum circuit compilation problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7707–7714.

Mehdi Saeedi, Robert Wille, and Rolf Drechsler. 2011. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing* 10, 3 (2011), 355–377.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

Animesh Sinha, Utkarsh Azad, and Harjinder Singh. 2021. Qubit Routing using Graph Neural Network aided Monte Carlo Tree Search. *arXiv preprint arXiv:2104.01992* (2021).

Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2019a. Qubit allocation as a combination of subgraph isomorphism and token swapping. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 120:1–120:29. https://doi.org/10.1145/3360546

Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2019b. Qubit allocation as a combination of subgraph isomorphism and token swapping. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–29.

Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 113–125.

Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t|ket>: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (2020), 014003.

Bochen Tan and Jason Cong. 2021. Optimality Study of Existing Quantum Computing Layout Synthesis Tools. *IEEE Trans. Computers* 70, 9 (2021), 1363–1373. https://doi.org/10.1109/TC.2020.3009140

Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. 2018. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology* 3, 2 (2018), 025004.

Davide Venturelli, Minh Do, Eleanor G Rieffel, and Jeremy Frank. 2017. Temporal Planning for Compilation of Quantum Approximate Optimization Circuits.. In *Twenty-Sixth International Joint Conference on Artificial Intelligence*. 4440–4446.

Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Chi Zhang, Yanhao Chen, Yuwei Jin, Wonsun Ahn, Youtao Zhang, and Eddy Z Zhang. 2020. A Depth-Aware Swap Insertion Scheme for the Qubit Mapping Problem. *arXiv preprint arXiv:2002.07289* (2020).

Chi Zhang, Ari B Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z Zhang. 2021. Time-optimal Qubit mapping. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 360–374.

Xiangzhen Zhou, Yuan Feng, and Sanjiang Li. 2020a. A Monte Carlo Tree Search Framework for Quantum Circuit Transformation. In *2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.

Xiangzhen Zhou, Yuan Feng, and Sanjiang Li. 2021. Supervised Learning Enhanced Quantum Circuit Transformation. *arXiv preprint arXiv:2110.03057* (2021).

Xiangzhen Zhou, Sanjiang Li, and Yuan Feng. 2020b. Quantum Circuit Transformation Based on Simulated Annealing and Heuristic Search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4683–4694. https://doi.org/10.1109/TCAD.2020.2969647

Pengcheng Zhu, Xueyun Cheng, and Zhijin Guan. 2020. An exact qubit allocation approach for NISQ architectures. *Quantum Information Processing* 19, 11 (2020), 1–21.

Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 7 (2018), 1226–1236.