Check for updates

Nested Qubit Routing

Harshit Dhankhar[1(✉)] and Tristan Cazenave[2]

[1] Indian Institute of Technology, Patna, India
harshit_2101mc20@iitp.ac.in
[2] LAMSADE, Université Paris Dauphine - PSL, Paris, France
tristan.cazenave@lamsade.dauphine.fr
https://harshit2807161.github.io/ ,
http://www.lamsade.dauphine.fr/ cazenave/index.php

Abstract. In the current NISQ era, it has become difficult to schedule increasingly complex quantum tasks with limited connectivity of the QPU (Quantum Processing Unit). This is partially attributed to the fact that we require qubits that share a task to be physically connected in the hardware topology. To satisfy this connectivity constraint, quantum circuits must make use of SWAP gates or reversing existing CNOT gates. Adding these gates comes with added computational cost and errors, which creates a need for efficient routing agents that can optimize this problem of qubit routing. We present a Nested Monte Carlo Search (NMCS) based agent (NesQ router) which aims to solve this problem by efficiently sampling the state space. In our experiments, NesQ was able to outperform other routing algorithms while offering a much lower runtime.

Keywords: Qubit Routing · Quantum Circuits · Monte Carlo Search · Artificial Intelligence

1 Introduction

The advent of Noisy Intermediate-Scale Quantum (NISQ) technology in recent years has witnessed an array of quantum computers with unique hardware architectures [1–3]. Such quantum devices support instructions which may be realized as a series of one and two-qubit operations (or gates), which may be assembled into a quantum circuit. One such circuit is shown in Fig. 2(a).

To execute these instructions and perform quantum computation, a circuit must be first compiled on the hardware architecture of the quantum device. Every quantum architecture has an associated device topology, also known as a connectivity graph, consisting of physical qubits (nodes) and connections (edges) between them. To compile the circuit, a routine must transform it to satisfy the connectivity constraints [4]. This is done by strategically placing SWAP gates such that any gate operation in the modified circuit only occurs between two physically linked qubits. This problem of "routing" the circuit to satisfy target device topology is known as Qubit routing [5].

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
S. Ali et al. (Eds.): QC+AI 2025, CCIS 2813, pp. 70–81, 2026.
https://doi.org/10.1007/978-3-032-15931-1_6

The qubit routing problem is specifically of interest due to the resource-constraint nature of quantum devices, with low fidelities, limited connectivity and poor quality of qubits [4]. It is also crucial to optimize the placement of SWAP gates to minimize the resulting circuit's overhead depth. This is essential to ensure effective computation before decoherence of qubits [6]. Lastly, minimizing output circuit depth also helps maximize Quantum Volume (QV), which quantifies the upper bound of circuit size that can be reliably executed on a quantum device [4].

Therefore, we aim to minimize the output circuit depth rather than choosing alternative objectives such as minimizing added gate count. We found this metric to be more significant as one may be readily convinced that long-running sparse circuits possessing a low total gate count may not be favorable. Further, minimizing gate count is a much easier problem which may be formulated in a simple action space, rather than a combinatorial one which we consider [4].

A few off-the-shelf transpilers to route quantum circuits exist, such as Cirq by Google [7], Qiskit (basic, SABRE, stochastic) by IBM [8], t|ket developed at Cambridge Quantum Computing (CQC) [9]. t|ket also employs BRIDGE gates which has shown to improve performance by Itoko et al. [10]. In the past, IBM organized a competition to find the best routing algorithm, won by Zulehner for his solution based on A* search [11]. Following this, people have approached the qubit routing problem in various ways. Some authors like Paler et al. [12] showed equivalence to traveling salesman problem on a torus, others such as Cowtan et al. [5] developed architecture-agnostic methods using graphs. Pozzi et al. [4] approached the problem using Double DQNs with simulated annealing. Sinha et al. [6] employed Monte Carlo Tree Search (MCTS) aided by Graph Neural Networks (GNN) and presented Qroute. More recently, Tang et al. [13] presented AlphaRouter, a solution which integrates MCTS with Reinforcement Learning (RL). Many heuristic-based approaches have also been implemented by [14–16].

Nested Monte Carlo Search (NMCS) [17] is a search algorithm that performs playouts at different recursive levels. A playout of level one chooses its move to play until a terminal state by playing a standard playout for each possible move at each step (see Fig. 1). A playout of level two does the same except that it plays playouts of level one for each possible move at each step of the level two playout. NMCS has been applied with success to various domains, starting from puzzles such as Morpion Solitaire and Kakuro to more recently retrosynthesis in Chemistry [18] improving the search of AIZynthFinder [19], Refutation of Spectral Graph Theory Conjectures [20] and generation of molecules [21].

We employ NMCS for our modified state and combinatorial action space in the qubit routing paradigm and name the framework NesQ. We also implemented optimization passes to further optimize the routed circuit, naming the resulting algorithm NesQ+.

We compiled our routing algorithm as an off-the-shelf Python framework named NesQ. We acknowledge the authors of Qroute [6] for their module doc-

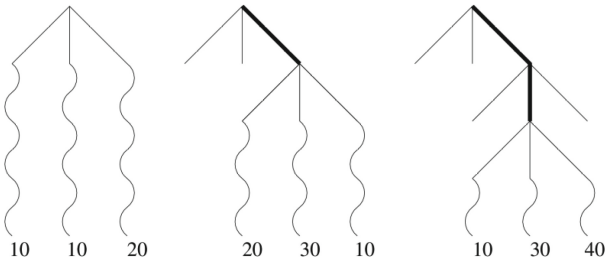


Fig. 1. A playout of level one. In the left tree the best playout scores 20 and is associated to the third move. So the third move is played at level 1, leading to the middle tree. Playouts are played for the three possible moves in the state of the middle tree and the middle move is associated to the best playout, leading to the tree on the left. The level one playout continues like this until a terminal state.

umentation upon which we built our package. The framework has been made public on Github along with the supplementary material.

We will now list the major contributions of our work:

- We present a novel framework employing nested version of Monte Carlo search to solve the qubit routing problem.
- We added further optimization passes to optimize the routed circuit and noticed a 13% lower average circuit depth in our realistic circuit benchmark.
- We ran benchmarks based on various aspects: i) Scalability on random circuits, ii) Realistic circuits, and iii) Generalizability across device topologies. We found our algorithm to exhibit an average of 10.55% lower circuit depth than the existing state-of-the-art framework in each of the benchmark while exhibiting a 37.17% lower average runtime.

2 Preliminaries and Related Works

2.1 Qubit Routing

The problem of qubit routing consists of transforming a given initial quantum circuit $\mathcal{C}$ by inserting SWAP gates such that the routed circuit $\mathcal{C}'$ satisfies the node connections of device topology $\mathcal{D}$. Any quantum circuit can be decomposed into layers such that each layer contains non-overlapping qubits, i.e., the qubit gates involved in a layer can be executed upon in parallel. The depth of a quantum circuit is defined as the number of such layers the circuit can be divided into, i.e., the number of distinct time steps required to schedule all the gates. The goal is then to minimize these overhead layers (the added depth) after adding the SWAP gates. For a given routing method $\mathcal{R}$, the process of qubit routing can be formulated as:

$$\mathcal{R}(\mathcal{C}, \mathcal{D}) \rightarrow \mathcal{C}' \tag{1}$$

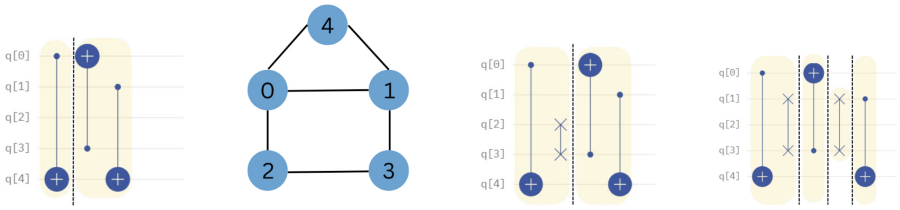


Fig. 2. Elementary example to demonstrate qubit routing. (a) depicts the input circuit, (b) is the connectivity graph of the device on which the circuit is compiled. (c) and (d) are two possible routing strategies that satisfy the device topology constraints. The parallelizable operations which constitute a layer are highlighted in yellow. (Color figure online)

Let us take a look at the elementary qubit routing example presented in Fig. 2. We are given a quantum circuit (Fig. 2(a)) to be compiled on a device with a connectivity graph as in Fig. 2(b). One may either come up with the idea of swapping q[1] and q[3] which adds a circuit depth of 2, or swapping q[2] and q[3] which adds no overhead depth. This example highlights the importance of strategically placing SWAP gates to minimize overhead depth, especially as the number of operations in the input circuit increases.

2.2 Nested Monte Carlo Search

Nested Monte Carlo Search (NMCS) is a search algorithm that combines nested calls with randomness in playouts and memorization of the best sequence [17]. When the search is guided by random playouts instead of a heuristic, it becomes important to memorize the best sequence in the case when nested search gives worse results than a previous search or a search at lower levels. This is the key idea behind NMCS. A step at each level plays each possible move and tries to search for the best sequence (sequence associated with the best score) using nested search at a lower level. If the best sequence is updated, we play the best move from the updated sequence; else, we play the move from the previously saved best sequence. The procedure is presented in Algorithm 1.

2.3 MCTS for Quantum Circuit Transformation

Previous works have used either plain MCTS for quantum circuit transformation [22,23] or MCTS guided by Graph Neural Networks (GNN) [6].

The GNN was used in combination with MCTS in a similar fashion to AlphaGo [24]. The GNN policy was used as a prior in the MCTS bandit to explore more the moves with a high probability. The GNN evaluation was used to evaluate the long-term reward of the newly created leaf of each MCTS descent. Our proposed algorithm is different since we use NMCS not MCTS.

Algorithm 1. The NMCS algorithm.

```
1: NMCS (state, level)
2:   if level == 0 then
3:     return playout (state)
4:   end if
5:   while state is not terminal do
6:     for m in possible moves for state do
7:       s ← play (state, m)
8:       s ← NMCS (s, level − 1)
9:       update best sequence using score (s)
10:    end for
11:    state ← play (state, move of the best sequence)
12:  end while
13:  return state
```

3 Framework

3.1 Environment Dynamics

The algorithm is fed the input circuit design and an initial injective mapping, $\mathcal{M} : \mathcal{L} \rightarrow \mathcal{P}$ where $\mathcal{L}$ and $\mathcal{P}$ represent the set of logical and physical qubits, respectively. The aim of the algorithm is to iteratively schedule valid gate operations onto the target device topology by placing some SWAP gates at each time step. At each time step, the algorithm tries to convert current operations (which are the set of first unscheduled operations for each qubit involved) to local operations, which are the set of free qubit operations that satisfy the hardware constraints. The algorithm first schedules all current operations that are inherently local. Then to evolve $\mathcal{M}_t$, it leverages NMCS to find an optimal set of SWAP gates such that no involved qubit is overlapping or locked from a previous operation and all gates are valid according to the device connectivity graph. This process is repeated at each time step until all gates in the input circuit are compiled. It is important to note that as the number of possible states while building a tree varies exponentially with the depth of the tree, we terminate the search at an intermediate state where we choose to commit the set of SWAPs to the current state. Lastly, we keep track of the set $\mathcal{Y}_t$ by employing mutex locks to freeze nodes that are currently being operated upon [6]. This helps us naturally address the different execution times required by different types of gates.

We now formulate the notion of state, move and action for applying the Monte Carlo based method. Here, move refers to a step in the tree search and action is the step taken in the environment after building it up move-by-move in a nested fashion according to Algorithm 1.

State: The state space at each time step t tries to encapsulate the device topology $\mathcal{D}$ and current compilation progress, and is defined as: $s_t = \{\mathcal{M}_t, \mathcal{U}_t, \mathcal{Y}_t, \mathcal{D}\}$ where $\mathcal{M}_t$ is the current mapping to the device, $\mathcal{U}_t$ is the set of unscheduled gates, and $\mathcal{Y}_t$ stores the nodes locked at time step t due to a previous unfinished operation.

Move: Since the set of SWAP gates to be scheduled at any time step, i.e., the action has an exponential relation with the number of device edges, the action space is combinatorial and thus we are forced to build up the SWAP set move-by-move. Let us first define the set of conditions $\mathcal{C}$ required to perform any valid action or move:

- The involved nodes must not belong to $\mathcal{Y}_t$, i.e., not being operated upon in the current time step.
- The added operation along with the set of all scheduled operations at the current time step must form a parallelizable set.
- The added operation must contain local gates (must be executable on the target device).

We are now in a position to define the two types of moves:

- **SWAP**(n_1,n_2): This adds a SWAP gate between nodes n_1 and n_2 such that $\mathcal{C}$ is satisfied. This move is appended to the move-set (action) being built up.
- **COMMIT**: This is a terminal move which indicates the completion of the formation of the move-set for current time step. It also schedules the current action (set of SWAPs) to the circuit and updates the state. Finally, it resets the move-set to an empty list for the next time step.

Action: Here, action is defined as the set of SWAP gates to be scheduled at a time step t, such that all its elements follow the condition set $\mathcal{C}$. The action space is combinatorial in nature as there are 2^n possible actions for n device edges.

3.2 Method

We use Nested Monte Carlo Search (NMCS) to guide the search of optimal SWAP sets to execute at any given time step t (see Algorithm 1). Given a state s_t, our algorithm aims to return a set of device edge indices where SWAP gates are to be scheduled. For this, we perform a level one NMCS at s_t. A playout of level one chooses its best move to play by searching until a terminal state is encountered by playing a random playout for each possible move and then finally choosing either the move from the new or previously saved best sequence. It plays this chosen move and transitions to a new state. This process is repeated for each new state until the best move chosen is COMMIT. These best moves are a series of suggested SWAPs stored in a bestSequence array which is finally returned to guide the circuit compilation at time step t.

To keep track of possible legal moves while doing playouts, we devise an action mask A corresponding to any given state s, with cardinality of $N + 1$, where N are the number of edges in the target device. The elements of A are either true (if the move satisfies condition set $\mathcal{C}$) or false (if not). Further, the last element of the mask points to the possibility of performing a COMMIT action. We name this complete framework NesQ and exhibit its efficacy in the following section.

3.3 Optimization Passes

The circuit depth of a routed circuit can be further optimized by performing an additional step of transpiler pass which has proven to improve the final circuit depth in Qiskit [8]. We extend the NesQ algorithm to include the following passes:

- **Optimize1qGates**: Optimize chains of single-qubit u1, u2, u3 gates by combining them into a single gate.
- **CommutativeCancellation**: Cancel the redundant (self-adjoint) gates through commutation relations.

With the transpiler pass on, we noticed a 13% lower average circuit depth in the realistic circuit benchmark. This encouraged us to include optimization passes in our algorithm and name the final procedure NesQ+.

4 Experiments and Results

In this section, we will delineate the performance of our algorithm on various circuit benchmarks by assessing the output circuit depth, circuit depth ratio (CDR) (see Eq. 2) and runtime. We choose these benchmarks to realize three expected outcomes:

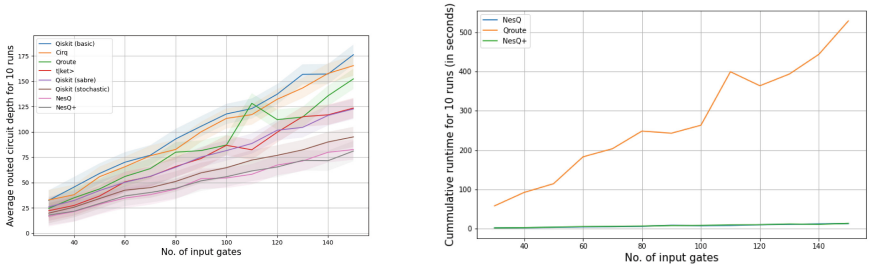
- The router is able to scale well with respect to number of gates in the circuit
- The router is able to perform well on realistic circuits
- The router is able to generalize well across various hardware topologies

We compare our NMCS based agent to various routing algorithms based on state-of-the-art frameworks: Qiskit (basic, SABRE, stochastic) by IBM [8], Cirq by Google [7], t|ket developed at Cambridge Quantum Computing (CQC) [9] and Qroute [6]. We ran Qroute for a search depth of 250 in random and small circuit experiments and 300 in large circuit and generalizability experiments. The results are compiled on IBM's QX20 Tokyo device.

$$\text{CDR} = \frac{1}{\#\text{circuits}} \sum_{\text{circuits}} \frac{\text{Output Circuit Depth}}{\text{Input Circuit Depth}} \tag{2}$$

4.1 Scalability on Random Circuits

To demonstrate the scalability of our router on quantum circuits, we design circuits on the fly with a same number of qubits as nodes on the hardware topology. We then insert two-qubit gates randomly between any two logical qubits to build the final circuit [6]. For our experiment, we simulate circuits with number of gates varying from 30 to 180 with a step size of 5, giving us a total of 30 randomly simulated circuits. We run all algorithms 10 times at each step and plot the average output circuit depth with respect to number of gates in the input circuit. These results are presented in Fig. 3(a). The average runtimes



(a) (b)

Fig. 3. Average output circuit depth (a) and cumulative runtime (for 10 runs) in seconds (b) plotted with respect to increasing number of gates in randomly simulated circuits.

are presented as a table in the supplementary material. Note that we do only compare the runtime of NesQ and NesQ+ with other MCTS based routers to have a fair comparison. Circuit depth ratios are presented in Table 1.

We observed NesQ to have a lower output circuit depth than other state-of-the-art routing algorithms, to scale well with number of input gates (lower slope in Fig. 3(a)) and retain an average runtime faster by 35.26x than the Qroute algorithm presented by Sinha et al. [6]. This runtime advantage can be attributed to the sample efficient way of doing rollouts in a nested fashion by NMCS. On average, we found the output circuit depth to be lower by 48.75% than Cirq, 51.60% than Qiskit basic, 14.83% than Qiskit stochastic, 32.57% than Qiskit sabre, 30.42% than t—ket, and 40.02% than Qroute.

4.2 Realistic Circuit Benchmarks

Apart from simulated circuits, it is important to examine how well a qubit routing agent can perform on a real-world circuit. We take the 158 circuit IBM-Q realistic quantum circuit dataset provided by Zulehner et al. [25]. We divide the dataset into two parts: i) Small circuits with 100 or fewer quantum gates and ii) Large realistic circuits with up to 5960 gates. The circuit depths and ratios along with runtimes are discussed in the following subsections.

Small Circuits: We ran NesQ and NesQ+ along with all state-of-the-art routers on the filtered small circuit dataset and plotted the cumulative output circuit depths in Fig. 4. While NesQ had a slightly higher cumulative circuit depth than Qroute (2690 and 2583 respectively), NesQ+ was able to beat all routers by a minimum of 5.27% with a circuit depth of 2447. In the cumulative runtime analysis, we found NesQ+ to have a runtime of 1.864 min which was faster by a factor 64.91x than Qroute which took 121 min for routing the dataset. The runtime results are presented in the supplementary material as a table. Lastly, Circuit depth ratios are presented in Table 1.

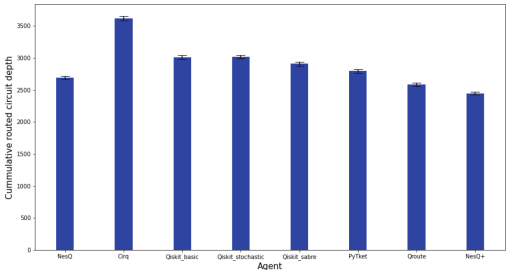
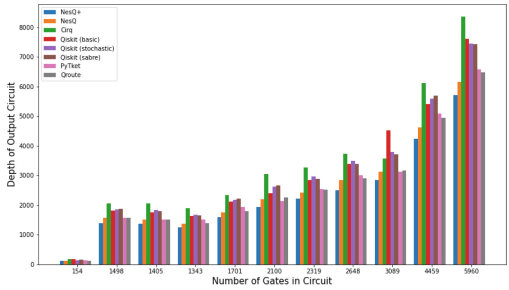
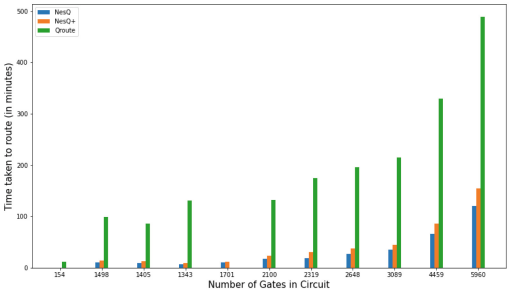


Fig. 4. Cumulative output circuit depth for different routing algorithms in small circuit experiment (upto one standard deviation considered in error bars).



(a)



(b)

Fig. 5. Output circuit depth (a) and runtime (in minutes) (b) for different routing algorithms on different large circuits.

Large Circuits: Lastly, we ran all algorithms on 11 large realistic circuits in the dataset with gates varying from 154 to 5960. We present the output circuit depths in Fig. 5(a). NesQ+ was able to outperform all other routers with a circuit depth ratio of 1.1512 next to Qroute with a CDR of 1.307 and Qiskit stochastic with 1.517. All circuit depth ratios are presented in Table 1. On average, NesQ+ found an 11.54% lower depth than Qroute and 24.26% than Qiskit stochastic. Runtime analysis table is presented in the supplementary material. Comparing

runtimes, we found NesQ+ to be 11.35x faster than Qroute on average while Cirq took 66.44 h for 5960 gates and 24.586 h for 4459 gates. Also note that Qroute took 1944.404 min to route "sym6_145" circuit with 1701 gates. We do not report these values in Fig. 5(b) due to potential scaling issues.

4.3 Generalizability Across Quantum Devices

In this era of increasing quantum processors, each having its unique network of physical qubits, it becomes pertinent for a qubit routing agent to be able to generalize across these devices without the need of training models from scratch. For our experiment, we consider IBM QX20 Tokyo (20 qubits), IBM-QX5 (16 qubits) and Rigetti 19Q-Acorn (19 qubits). For these devices, we test the best-performing algorithms (NesQ+ and Qroute) across various large circuits and present the ratio of routed circuit depth by NesQ+ and Qroute. These results are presented as a heatmap in Fig. 6. The runtimes are presented in the supplementary material. On average, NesQ+ is 94.86% faster than Qroute on IBM-QX5, 83.16% faster on IBM QX20 Tokyo and 94.36% faster on Rigetti 19Q-Acorn. We observe that NesQ+ outperforms Qroute on all configurations except one. The lowest average ratio of CDRs is observed on Rigetti 19Q-Acorn which is impressive as it is an architecture with sparse connectivity, containing nodes with either a degree of 2 or at most 3 [4].

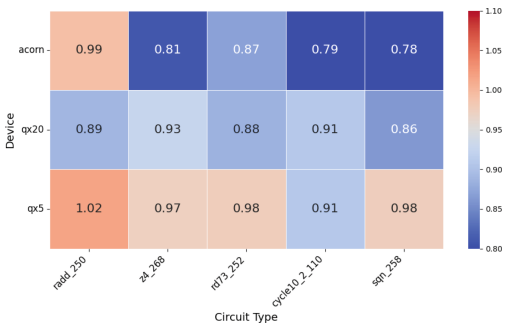


Fig. 6. Ratio of CDR[NesQ+] to CDR[Qroute] for different large circuits across various hardware topologies.

Table 1. Circuit depth ratios (CDRs) for all benchmarks

Benchmark	NesQ	Cirq	Q-basic	Q-stochastic	Q-sabre	t—ket	Qroute	NesQ+
Random	1.9641	3.8373	4.0654	2.3063	2.9131	2.8384	3.3097	1.9837
Small	1.2201	1.6209	1.3587	1.3600	1.2843	1.2109	1.1569	1.0903
Large	1.2635	1.6979	1.5592	1.5175	1.5480	1.3222	1.9866	1.1512

5 Conclusion

We presented NesQ+, a Nested Monte Carlo Search algorithm which optimizes Qubit Routing for quantum circuits. NesQ+ achieves lower circuit depths than all the other routing algorithms we could test by an average of 10.55%. It is also faster than other Monte Carlo based methods by an average of 37.17%. It particularly shines in large circuit benchmarks which displays its robustness with respect to increasing number of gates and layers.

There are many possible future works. For other combinatorial problems, using a prior improved nested search algorithms a lot. NesQ+ could benefit from using a heuristic as a prior for routing. Another possible improvement would be to try Deep Reinforcement Learning algorithms for learning either an evaluation function or a policy. The policy could be used associated to the evaluation function in an MCTS algorithm, or it could serve as a prior for policy learning.

References

1. Arute, F., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**, 505–510 (2019)
2. Karalekas, P.J., et al.: A quantum-classical cloud platform optimized for variational hybrid algorithms. Quantum Sci. Technol. **5**(2), 024003 (2020)
3. IBM, IBM debuts next-generation quantum processor IBM quantum system two, extends roadmap to advance era of quantum utility (2023). https://newsroom.ibm.com/2023-12-04-IBM-Debuts-Next-Generation-Quantum-Processor-IBM-Quantum-System-Two,-Extends-Roadmap-to-Advance-Era-of-Quantum-Utility
4. Pozzi, M.G., et al.: Using reinforcement learning to perform qubit routing in quantum compilers. ACM Trans. Quantum Comput. **3**(2), 10 (2022)
5. Cowtan, A., et al.: On the qubit routing problem. In: 14th Conference on Theory of Quantum Computation, Communication, and Cryptography (TQC), pp. 5:1–5:32 (2019)
6. Sinha, A., Azad, U., Singh, H.: Qubit routing using graph neural network aided Monte Carlo tree search. Proc. AAAI Conf. Artif. Intell. **36**(9), 9935–9943 (2022)
7. Cirq Developers, "Cirq," version v1.4.0 (2024). https://doi.org/10.5281/zenodo.11398048
8. Aleksandrowicz, G., et al.: Qiskit: an open-source framework for quantum computing (2019). https://doi.org/10.5281/ZENODO.2562111
9. Sivarajah, S., et al.: t|ket⟩: a retargetable compiler for NISQ devices. Quantum Sci. Technol. **6**(1), 014003 (2020)
10. Itoko, T., et al.: Optimization of quantum circuit mapping using gate transformation and commutation. Integration **70**, 43–50 (2020)
11. Zulehner, A., Paler, A., Wille, R.: An efficient methodology for mapping quantum circuits to the IBM QX architectures. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **38**(7), 1226–1236 (2019)
12. Paler, A., Zulehner, A., Wille, R.: NISQ circuit compilation is the travelling salesman problem on a torus. Quantum Sci. Technol. **6**, (2021)
13. Tang, W., et al.: Alpharouter: quantum circuit routing with reinforcement learning and tree search. In: IEEE Quantum Week (2024)

14. Wagner, F., et al.: Improving quantum computation by optimized qubit routing. J. Optim. Theory Appl. **197**(3), 1161–1194 (2023)
15. Cheng, C.-Y., et al.: Robust qubit mapping algorithm via double-source optimal routing on large quantum circuits. ACM Trans. Quantum Comput. **5**(3), 20 (2024)
16. Chand, S., et al.: Rollout based heuristics for the quantum circuit compilation problem. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 974–981 (2019)
17. Cazenave, T.: Nested monte-carlo search. In: Twenty-First International Joint Conference on Artificial Intelligence, pp. 456–461 (2009)
18. Roucairol, M., Cazenave, T.: Comparing search algorithms on the retrosynthesis problem. Mol. Inform. e202300259 (2024)
19. Genheden, S., et al.: Aizynthfinder: a fast, robust and flexible open-source software for retrosynthetic planning. J. Cheminform. **12**(1), 70 (2020)
20. Roucairol, M., Cazenave, T.: Refutation of spectral graph theory conjectures with Monte Carlo search. In: COCOON 2022 (2022)
21. Roucairol, M., et al.: Drugsynthmc: an atom-based generation of drug-like molecules with Monte Carlo search. J. Chem. Inf. Model. (2024)
22. Zhou, X., Feng, Y., Li, S.: A monte carlo tree search framework for quantum circuit transformation. In: Proceedings of 39th International Conference on Computer-Aided Design, pp. 1–7 (2020)
23. Zhou, X., Feng, Y., Li, S.: Quantum circuit transformation: a Monte Carlo tree search framework. ACM Trans. Des. Autom. Electron. Syst. (2022)
24. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
25. Zulehner, A., Paler, A., Wille, R.: An efficient methodology for mapping quantum circuits to the IBM QX architectures. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **38**(7), 1226–1236 (2018)