Nested Qubit Routing

Harshit Dhankhar[1], **Tristan Cazenave**[2]

[1]Indian Institute of Technology, Patna,
[2]LAMSADE, Université Paris Dauphine - PSL

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 which 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.

Introduction

The advent of Noisy Intermediate-Scale Quantum (NISQ) technology in the recent years has witnessed an array of quantum computers with unique hardware architectures (Arute et al. (2019), Karalekas et al. (2020), IBM (2023), etc). 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 (Pozzi et al. 2022). 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 (Cowtan et al. 2019).

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 (Pozzi et al. 2022). 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 (Sinha, Azad, and Singh 2022). 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 (Pozzi et al. 2022).

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 (Pozzi et al. 2022).

A few off-the-shelf transpilers to route quantum circuits exist, such as Cirq by Google (Developers 2024), Qiskit (basic, SABRE, stochastic) by IBM (Aleksandrowicz et al. 2019), t|ket developed at Cambridge Quantum Computing (CQC) (Sivarajah et al. 2020). t|ket also employs BRIDGE gates which has shown to improve performance by (Itoko et al. 2020). In the past, IBM organized a competition to find the best routing algorithm, won by Zulehner for his solution based on A* search (Zulehner, Paler, and Wille 2019). Following this, people have approached the qubit routing problem in various ways. Some authors like Paler, Zulehner, and Wille (2021) showed equivalence to traveling salesman problem on a torus, others such as Cowtan et al. (2019) developed architecture-agnostic methods using graphs. Pozzi et al. (2022) approached the problem using Double DQNs with simulated annealing. Sinha, Azad, and Singh (2022) employed Monte Carlo Tree Search (MCTS) aided by Graph Neural Networks (GNN) and presented Qroute. More recently, Tang et al. (2024) presented AlphaRouter, a solution which integrates MCTS with Reinforcement Learning (RL). Many heuristic-based approaches have also been imple-

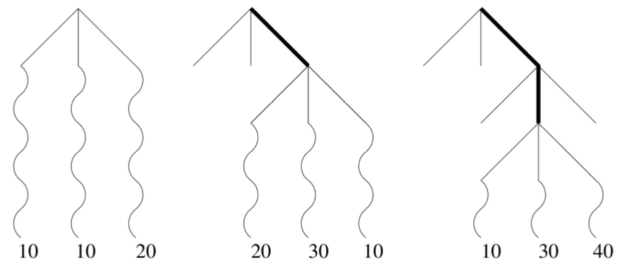


Figure 1: A playout of level one. In the left tree the best playout scores 20 and is associated to the third move. So the third move is played at level 1, leading to the middle tree. Playouts are played for the three possible moves in the state of the middle tree and the middle move is associated to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

mented by Wagner et al. (2023), Chand et al. (2019), Cheng et al. (2024).

Nested Monte Carlo Search (NMCS) (Cazenave 2009) 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 Figure 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 (Roucairol and Cazenave 2024) improving the search of AIZynthFinder (Genheden et al. 2020), Refutation of Spectral Graph Theory Conjectures (Roucairol and Cazenave 2022) and generation of molecules (Roucairol et al. 2024).

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 module named NesQ. We acknowledge the authors of Qroute (Sinha, Azad, and Singh 2022) for their module documentation upon which we built our package. The framework will be made public on Github as an all-inclusive Python module upon acceptance at AISTATS 2025. The same ready-to-route module is attached in the supplementary material for the reviewer's reference.

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.

The remainder of this paper is organized as follows: We begin by presenting a brief summary of preliminary topics in Section . This is followed by a formulation of our algorithm in Section . We benchmark our proposed algorithm in Section and draw a conclusion in Section .

Preliminaries and related works

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:

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

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 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 increase.

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 (Cazenave 2009). 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 lower level. If the best sequence is updated, we play the best move from updated sequence, else we play the move from the previously saved best sequence. The procedure is presented in Algorithm 1.

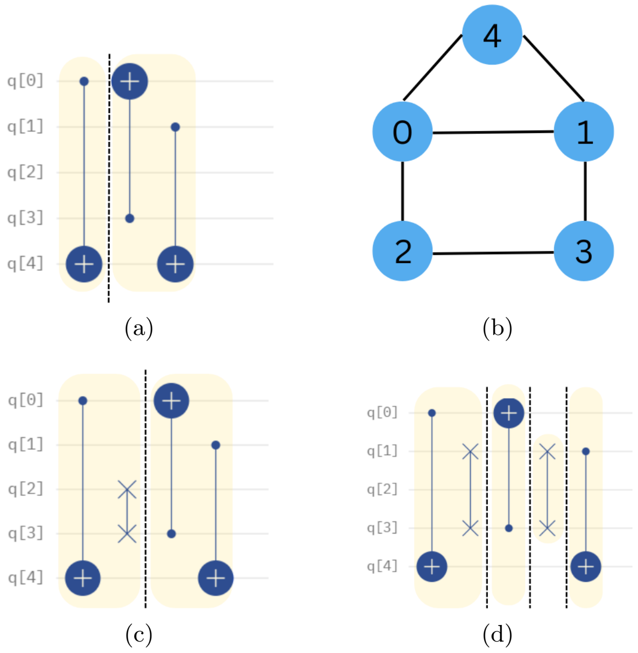


Figure 2: Elementary example to demonstrate qubit routing. (a) depicts the input circuit, (b) is the connectivity graph of the device on which the circuit is compiled. (c) and (d) are two possible routing strategies that satisfy the device topology constraints. The parallelizable operations which constitute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

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:             $s \leftarrow$ play ($state$, $m$)
8:             $s \leftarrow$ NMCS ($s$, $level - 1$)
9:             update best sequence using score ($s$)
10:        end for
11:        $state \leftarrow$ play ($state$, move of the best sequence)
12:    end while
13:    return $state$
```

MCTS for Quantum Circuit Transformation

Previous works have used either plain MCTS for quantum circuit transformation (Zhou, Feng, and Li 2020, 2022) or MCTS guided by Graph Neural Networks (GNN) (Sinha, Azad, and Singh 2022).

The GNN was used in combination with MCTS in a similar fashion to AlphaGo (Silver et al. 2016). 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.

Framework

Environment dynamics

The algorithm is fed the input circuit design and an initial injective mapping, $\mathcal{M} : \mathcal{L} \to \mathcal{P}$ 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 which satisfy the hardware constraints. The algorithm first schedules all current operations which 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 the 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 (Sinha, Azad, and Singh

2022). This helps us to 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 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.

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 Section .

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 (Aleksandrowicz et al. 2019). 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 discussed in Section . This encouraged us to include optimization passes in our algorithm and name the final procedure NesQ+.

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 equation 2) and runtime. We choose these benchmarks to realize three expected outcomes:

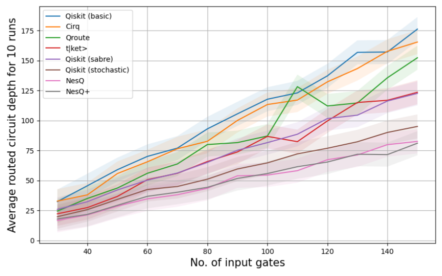
- The router is able to scale well with respect to number of gates in the circuit
- The router is able to perform well on realistic circuits
- The router is able to generalize well across various hardware topologies

We compare our NMCS based agent to various routing algorithms based on state-of-the-art frameworks: Qiskit (basic, SABRE, stochastic) by IBM (Aleksandrowicz et al. 2019), Cirq by Google (Developers 2024), t|ket developed at Cambridge Quantum Computing (CQC) (Sivarajah et al. 2020) and Qroute (Sinha, Azad, and Singh 2022). We ran Qroute for a search depth of 250 in sections , and 300 in sections , . The results are compiled on IBM's QX20 Tokyo device.

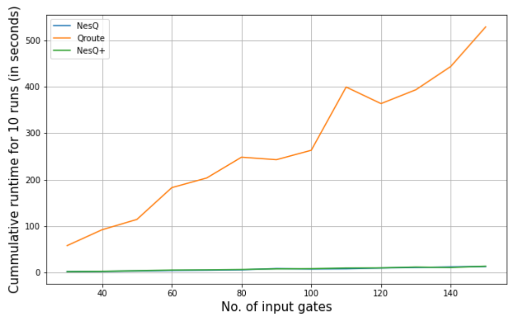
$$\text{CDR} = \frac{1}{\#\text{circuits}} \sum_{\text{circuits}} \frac{\text{Output Circuit Depth}}{\text{Input Circuit Depth}} \quad (2)$$

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 (Sinha, Azad, and Singh 2022). 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 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.



(a)



(b)

Figure 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.

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, Azad, and Singh 2022). 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.

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, Paler, and Wille 2018). 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 minutes which was faster by a factor 64.91x than Qroute which took 121 minutes 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.

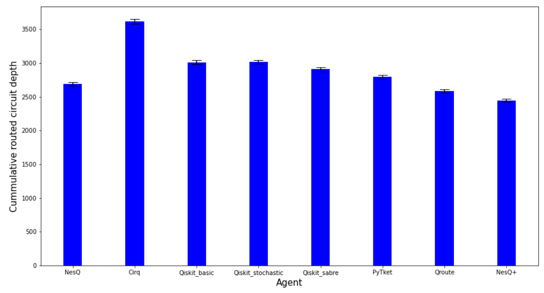
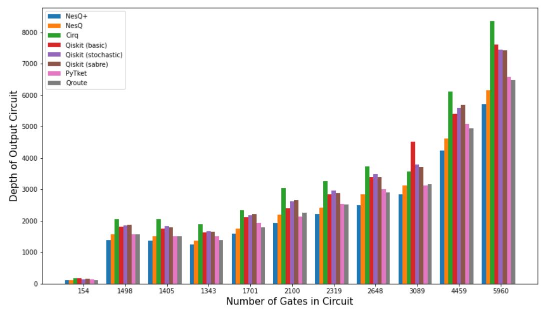
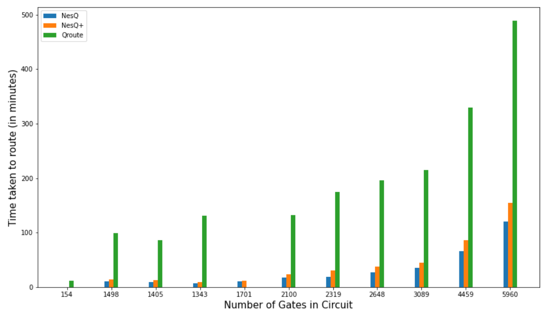


Figure 4: Cumulative output circuit depth for different routing algorithms (upto one standard deviation considered in error bars).

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 hours for 5960 gates and 24.586 hours for 4459 gates. Also note that Qroute took 1944.404 minutes to route “sym6_145” circuit with 1701 gates. We do not report these values in Fig. 5(b) due to potential scaling issues.



(a)



(b)

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

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 (Pozzi et al. 2022).

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

Aleksandrowicz, G.; Alexander, T.; Barkoutsos, P. K.; Bello, L.; Ben-Haim, Y.; Bucher, D.; Cabrera-Hernández, F. J.; Carballo-Franquis, J.; Chen, A.; Chen, C.-F.; Chow, J. M.; Córcoles-Gonzales, A. D.; Cross, A. J.; Cross, A. W.; Cruz-Benito, J.; Culver, C.; González, S. D. L. P.; Torre, E. D. L.; Ding, D.; Dumitrescu, E. F.; Duran, I.; Eendebak, P. T.; Everitt, M.; Sertage, I. F.; Frisch, A.; Fuhrer, A.; Gambetta, J. M.; Gago, B. G.; Gomez-Mosquera, J.; Greenberg, D.; Hamamura, I.; Havlicek, V.; Hellmers, J.; Łukasz Herok; Horii, H.; Hu, S.; Imamichi, T.; Itoko, T.; Javadi-Abhari, A.; Kanazawa, N.; Karazeev, A.; Krsulich, K.; Liu, P.; Luh, Y.; Maeng, Y.; Marques, M.; Martín-Fernández, F.; McClure, D.; McKay, D.; Meesala, S.; Mezzacapo, A.; Moll, N.; Rodríguez, D. M.; Nannicini, G.; Nation, P. D.; Ollitrault, P. J.; O'Riordan, L. J.; Paik, H.; Pérez, J.; Phan, A.; Pistoia, M.; Prutyanov, V.; Reuter, M.; Rice, J. E.; Davila, A. R.; Rudy, R. H.; Ryu, M.; Sathaye, N.; Schnabel, C.; Schoute, E.; Setia, K.; Shi, Y.; Silva, A.; Siraichi, Y.; Sivarajah, S.; Smolin, J. A.; Soeken, M.; Takahashi, H.; Tavernelli, I.; Taylor, C.; Taylour, P.; Trabing, K.; Treinish, M.; Turner, W.; Vogt-Lee, D.; Vuillot, C.; Wildstrom, J. A.; Wilson, J.; Winston, E.; Wood, C. J.; Wood, S. P.; Wörner, S.; Akhalwaya, I. Y.; and Zoufal, C. 2019. Qiskit: An Open-source Framework for Quantum Computing.

Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F.; Buell, D.; Burkett, B.; Chen, Y.; Chen, J.; Chiaro, B.; Collins, R.; Courtney, W.; Dunsworth, A.; Farhi, E.; Foxen, B.; Fowler, A.; Gidney, C. M.; Giustina, M.; Graff, R.; Guerin, K.; Habegger, S.; Harrigan, M.; Hartmann, M.; Ho, A.; Hoffmann, M. R.; Huang, T.; Humble, T.; Isakov, S.; Jeffrey, E.; Jiang, Z.; Kafri, D.; Kechedzhi, K.; Kelly, J.; Klimov, P.; Knysh, S.; Korotkov, A.; Kostritsa, F.; Landhuis, D.; Lindmark, M.; Lucero, E.; Lyakh, D.; Mandrà, S.; McClean, J. R.;

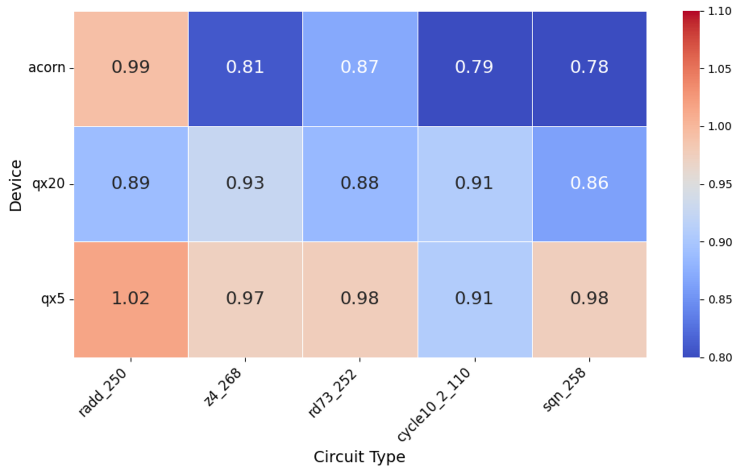


Figure 6: Ratio of CDR[NesQ+] to CDR[Qroute] for different large circuits across various hardware topologies.

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

McEwen, M.; Megrant, A.; Mi, X.; Michielsen, K.; Mohseni, M.; Mutus, J.; Naaman, O.; Neeley, M.; Neill, C.; Niu, M. Y.; Ostby, E.; Petukhov, A.; Platt, J.; Quintana, C.; Rieffel, E. G.; Roushan, P.; Rubin, N.; Sank, D.; Satzinger, K. J.; Smelyanskiy, V.; Sung, K. J.; Trevithick, M.; Vainsencher, A.; Villalonga, B.; White, T.; Yao, Z. J.; Yeh, P.; Zalcman, A.; Neven, H.; and Martinis, J. 2019. Quantum Supremacy using a Programmable Superconducting Processor. *Nature*, 574: 505–510.

Cazenave, T. 2009. Nested Monte-Carlo Search. In *Twenty-First International Joint Conference on Artificial Intelligence*, 456–461.

Chand, S.; Singh, H. K.; Ray, T.; and Ryan, M. 2019. Rollout based Heuristics for the Quantum Circuit Compilation Problem. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, 974–981.

Cheng, C.-Y.; Yang, C.-Y.; Kuo, Y.-H.; Wang, R.-C.; Cheng, H.-C.; and Huang, C.-Y. R. 2024. Robust Qubit Mapping Algorithm via Double-Source Optimal Routing on Large Quantum Circuits. *ACM Transactions on Quantum Computing*, 5(3).

Cowtan, A.; Dilkes, S.; Duncan, R.; Krajenbrink, A.; Simmons, W.; and Sivarajah, S. 2019. On the qubit routing problem. In van Dam, W.; and Mancinska, L., eds., *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 5:1—-5:32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. ISBN 9783959771122. 14th Conference on the theory of Quantum Computation, Communication and Cryptography, TQC 2019 ; Conference date: 03-06-2019 Through 07-06-2019.

Developers, C. 2024. Cirq.

Edelkamp, S.; Gath, M.; Cazenave, T.; and Teytaud, F. 2013. Algorithm and knowledge engineering for the TSPTW problem. In *Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on*, 44–51. IEEE.

Edelkamp, S.; Gath, M.; and Rohde, M. 2014. Monte-Carlo Tree Search for 3D Packing with Object Orientation. In *KI 2014: Advances in Artificial Intelligence*, 285–296. Springer International Publishing.

Edelkamp, S.; and Greulich, C. 2014. Solving physical traveling salesman problems with policy adaptation. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 1–8. IEEE.

Edelkamp, S.; and Tang, Z. 2015. Monte-Carlo Tree Search for the Multiple Sequence Alignment Problem. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015*, 9–17. AAAI Press.

Elkael, M.; Aba, M. A.; Araldo, A.; Castel-Taleb, H.; and Jouaber, B. 2022. Monkey business: Reinforcement learning meets neighborhood search for virtual network embedding. *Computer Networks*, 216: 109204.

Genheden, S.; Thakkar, A.; Chadimová, V.; Reymond, J.-L.; Engkvist, O.; and Bjerrum, E. 2020. AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. *Journal of Cheminformatics*, 12(1): 70.

IBM. 2023. Technical report.

Itoko, T.; Raymond, R.; Imamichi, T.; and Matsuo, A. 2020. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70: 43–50.

Karalekas, P. J.; Tezak, N. A.; Peterson, E. C.; Ryan, C. A.; da Silva, M. P.; and Smith, R. S. 2020. A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Science and Technology*, 5(2): 024003.

Paler, A.; Zulehner, A.; and Wille, R. 2021. NISQ circuit compilation is the travelling salesman problem on a torus. *Quantum Science and Technology*, 6.

Pozzi, M. G.; Herbert, S. J.; Sengupta, A.; and Mullins, R. D. 2022. Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers. *ACM Transactions on Quantum Computing*, 3(2).

Rosin, C. D. 2011. Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 649–654.

Roucairol, M.; and Cazenave, T. 2022. Refutation of Spectral Graph Theory Conjectures with Monte Carlo Search. In *COCOON 2022*.

Roucairol, M.; and Cazenave, T. 2024. Comparing search algorithms on the retrosynthesis problem. *Molecular Informatics*, e202300259.

Roucairol, M.; Georgiou, A.; Cazenave, T.; Prischi, F.; and Pardo, O. E. 2024. DrugSynthMC: An Atom-Based Generation of Drug-like Molecules with Monte Carlo Search. *Journal of Chemical Information and Modeling*.

Sentuc, J.; Cazenave, T.; and Lucas, J. 2022. Generalized Nested Rollout Policy Adaptation with Dynamic Bias for Vehicle Routing. In *AI for Transportation at AAAI*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489.

Sinha, A.; Azad, U.; and Singh, H. 2022. Qubit routing using graph neural network aided Monte Carlo tree search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, 9935–9943.

Sivarajah, S.; Dilkes, S.; Cowtan, A.; Simmons, W.; Edgington, A.; and Duncan, R. 2020. t— ket¿: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1): 014003.

Tang, W.; Duan, Y.; Kharkov, Y.; Fakoor, R.; Kessler, E.; and Shi, Y. 2024. AlphaRouter: Quantum circuit routing with reinforcement learning and tree search. In *IEEE Quantum Week 2024*.

Wagner, F.; Bärmann, A.; Liers, F.; and Weissenbäck, M. 2023. Improving quantum computation by optimized qubit routing. *Journal of Optimization Theory and Applications*, 197(3): 1161–1194.

Zhou, X.; Feng, Y.; and Li, S. 2020. A monte carlo tree search framework for quantum circuit transformation. In *Proceedings of the 39th International Conference on Computer-Aided Design*, 1–7.

Zhou, X.; Feng, Y.; and Li, S. 2022. Quantum Circuit Transformation: A Monte Carlo Tree Search Framework. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*.

Zulehner, A.; Paler, A.; and Wille, R. 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): 1226–1236.

Zulehner, A.; Paler, A.; and Wille, R. 2019. 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): 1226–1236.

Appendix

In this section, we present the runtime tables for different benchmarks shown in the main text. The results for GNRPA are also included in this section. The small circuit benchmark runtime table is attached as a csv file along with the code.

Random circuit benchmark

The runtimes are presented in Table 2. Note that these results are averaged over 10 runs for any given number of gates.

Large circuit benchmark

The runtimes for different large scale circuits are shown in Table 3.

Device benchmark

The time taken by NesQ+ and Qroute to route the chosen circuits on different devices is given in Table 4.

Generalised Nested Rollout Policy Adaptation (GNRPA)

Nested Rollout Policy Adaptation (NRPA) (Rosin 2011) combines nested search, memorizing the best sequence of moves found at each level, and the online learning of a playout policy using this sequence. NRPA has world records in Morpion Solitaire and crossword puzzles and has also been applied to many other combinatorial problems such as the Traveling Salesman Problem with Time Windows (Edelkamp et al. 2013), 3D Packing with Object Orientation (Edelkamp, Gath, and Rohde 2014), the physical traveling salesman problem (Edelkamp and Greulich 2014), the Multiple Sequence Alignment problem (Edelkamp and Tang 2015), Logistics, Graph Coloring, Vehicle Routing Problems, Network Traffic Engineering, Virtual Network Embedding (Elkael et al. 2022) or the Snake in the Box.

Generalized Nested Rollout Policy Adaptation (GNRPA) generalizes the way the probability is calculated using a bias. The bias is a heuristic that performs non uniform playouts and using it usually gives much better results than uniform playouts. The use of a bias has been theoretically demonstrated more general than the initialization of the weights. The GNRPA paper also provides a theoretical derivation of the learning of the policy, using a cross entropy loss associated to a softmax. GNRPA has been applied to some difficult problems such as Inverse RNA Folding and Vehicle Routing Problems (Sentuc, Cazenave, and Lucas 2022) with better results than NRPA.

While surveying different MCTS-based algorithms for solving the qubit routing problem, we experimented with Generalised Nested Rollout Policy Adaptation and observed results at-par with most of the routers (see Section), except when circuit size increases (i.e. large circuit experiment).

Algorithm

The idea behind GNRPA is to learn a rollout policy by adapting the weights on each action. These weights of choosing an action are modelled by first, hashing a move m using a *code* function which gives each action a unique address and then mapping this to the weights using *policy* function. The GNRPA algorithm (see Algorithm 4) performs fixed number of calls (N iterations) to the lower level search and adapt function. At its base level, the algorithm performs playout (see Algorithm 2) with a probability equal to the softmax function applied to the weights plus the bias of the possible moves. The idea of finding a new sequence of moves from lower level searches and updating it based on score (reward) holds as in NMCS. In each iteration, the policy is also adapted (via Algorithm 3) by pushing the weights according to moves in the best sequence found yet.

Algorithm 2 The playout algorithm

```
 1: playout ($policy$)
 2:     $state \leftarrow root$
 3:     while true do
 4:         if terminal($state$) then
 5:             return (score ($state$), $sequence(state)$)
 6:         end if
 7:         $z \leftarrow 0$
 8:         for $m \in$ possible moves for $state$ do
 9:             $o[m] \leftarrow e^{policy[code(m)]+\beta_m}$
10:             $z \leftarrow z + o[m]$
11:         end for
12:         choose a $move$ with probability $\frac{o[move]}{z}$
13:         play ($state$, $move$)
14:     end while
```

Algorithm 3 The adapt algorithm

```
 1: adapt ($policy$, $sequence$)
 2:     $polp \leftarrow policy$
 3:     $state \leftarrow root$
 4:     for $b \in sequence$ do
 5:         $z \leftarrow 0$
 6:         for $m \in$ possible moves for $state$ do
 7:             $o[m] \leftarrow e^{policy[code(m)]+\beta_m}$
 8:             $z \leftarrow z + o[m]$
 9:         end for
10:         for $m \in$ possible moves for $state$ do
11:             $p_m \leftarrow \frac{o[m]}{z}$
12:             $polp[code(m)] \leftarrow polp[code(m)] - \alpha(p_m - \delta_{bm})$
13:         end for
14:         play ($state$, $b$)
15:     end for
16:     $policy \leftarrow polp$
```

Results

We ran level 1 GNRPA router for 250 iterations on random and small quantum ciruit benchmarks. The reason we leave out it for large scale quantum circuits

Gates	**NesQ**	**Cirq**	**Qroute**	**GNRPA**	**NesQ+**
30	1.741882	0.095985	57.910772	80.375707	1.875454
40	2.321188	0.137800	91.984908	116.2286	2.524887
50	3.224417	0.185684	114.231547	158.796500	3.862081
60	4.343773	0.329320	182.626015	233.352610	5.248788
70	4.880676	0.414190	203.357469	233.359562	5.783488
80	5.690395	0.515367	248.183977	255.631882	6.534142
90	8.582040	0.704814	242.724301	312.194811	7.850141
100	7.485916	0.923527	263.054020	377.962436	8.297419
110	8.045412	1.279350	399.084266	400.819585	9.705672
120	9.823262	1.537304	363.455770	450.026435	9.970981
130	10.797684	2.012254	393.310650	503.998836	11.788189
140	12.106278	2.219034	443.379280	581.633442	10.778473
150	12.924456	3.202757	528.393336	650.770277	13.553065

Table 2: Average Runtimes (of 10 runs) for random circuit benchmark (in seconds)

Circuit Name	**Gates**	**NesQ**	**Cirq**	**Qroute**	**NesQ+**
rd84_142	154	0.202823	0.037918	12.207869	0.270372
adr4_197	1498	11.086658	54.814014	99.807621	14.060009
radd_250	1405	9.683338	46.239436	86.420729	13.461079
z4_268	1343	7.301565	35.613734	131.339545	9.568750
sym6_145	1701	10.237229	58.775746	1944.404549	11.602539
misex1_241	2100	18.023073	142.878779	132.625253	23.869278
rd73_252	2319	19.430781	198.861971	175.083901	30.877758
cycle10_2_110	2648	27.617641	321.146342	195.689001	38.172365
square_root_7	3089	35.293369	356.253962	214.469363	45.133008
sqn_258	4459	66.416509	1491.360893	330.1394166	85.826042
rd84_253	5960	120.109507	3986.860282	489.3008025	154.411345

Table 3: Runtimes for large realistic circuit benchmark in minutes.

Circuit Name	Device	NesQ+	Qroute
radd_250	qx5	11.30	133.39
	qx20	12.05	89.18
	acorn	13.30	245.71
z4_268	qx5	8.61	1238.17
	qx20	9.64	119.50
	acorn	10.96	1652.84
rd73_252	qx5	25.34	1396.74
	qx20	25.80	165.37
	acorn	29.18	2009.93
cycle10_2_110	qx5	34.80	356.34
	qx20	36.07	188.15
	acorn	43.84	460.99
sqn_258	qx5	91.05	1833.69
	qx20	87.89	315.64
	acorn	113.56	1015.75

Table 4: Time taken (in minutes) for NesQ+ and Qroute to route selected large circuits over across different devices topologies.

Algorithm 4 The GNRPA algorithm

```
1: GNRPA ($level$, $policy$)
2:   if level == 0 then
3:     return playout ($policy$)
4:   else
5:     $bestScore \leftarrow -\infty$
6:     for N iterations do
7:       (score,new) ← GNRPA($level - 1$, $policy$)
8:       if score ≥ bestScore then
9:         bestScore ← score
10:        seq ← new
11:      end if
12:      $policy$ ← adapt ($policy$, $seq$)
13:    end for
14:    return (bestScore, seq)
15:  end if
```

is that we found it to take atleast 25 hours for circuits with gate count of 1498 (adr4_197) or more.

Random circuit benchmark The runtimes are presented in Table 2. GNRPA maintains a slightly higher runtime than Qroute for all circuits (25.29% on average) but successfully retains a lower circuit depth than Qroute (by 20.23%), Cirq (by 31.86%), Qiskit basic (by 35.69%), Qiskit sabre (by 34.38%) and Pytket (by 7.42%). NesQ was able to beat GNRPA by 24.6% lower output circuit depth, reinforcing the superiority of our algorithm.

Small realistic circuits benchmark GNRPA exhibited a cumulative circuit depth of 3324 in this benchmark, next to Qiskit stochastic (3016) and lower than the worst performing router, Cirq (3616). GNRPA took 234 minutes to route the dataset which was almost twice compared to Qroute (121 minutes).