Quantum Circuit Routing Optimises the Wrong Metric

Closing the Proxy Gap Between SWAP Count and Schedule Length

Author: Dan Cernatinschi | Supervisors: S. Feld, A. Kundu, M. Spaan | EEMCS, Delft University of Technology

1. Introduction

To run a quantum circuit on real hardware, a compiler inserts SWAP gates until every interacting pair of qubits is physically adjacent on the chip. How it chooses them shapes the circuit's success probability, the chance the quantum computer returns the right answer. A standard model from the noise-aware compilation literature, the Expected Success Probability (ESP), estimates it from two terms:

$$\mathrm{ESP} \;=\; \underbrace{(1-\epsilon)^{N}\,(1-\epsilon_1)^{N_1}}_{\text{gate error}} \cdot \underbrace{\textstyle\prod_q e^{-t_{\mathrm{idle}}(q)/T_2}}_{\text{decoherence}}.$$

It falls as the gate count N and the makespan, the schedule length, grow. The makespan is a good measure of the decoherence term, which dominates on present-day superconducting hardware.

Routers across the literature, including SABRE, the one Qiskit ships, minimise only the SWAP count: a proxy for gate count, but not for the makespan, which is the explicit objective of the next pass, scheduling. Routing thus tunes a proxy for a cost the next stage settles directly. We call this the *proxy gap*, and Fig. 1 shows it is real.

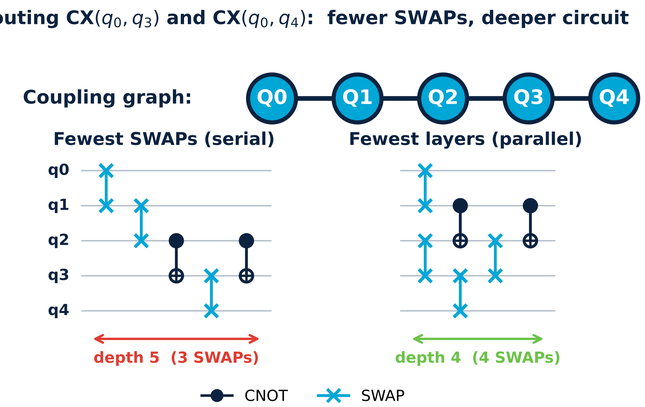


Figure 1. Even with only two CNOTs, the fewest-SWAP routing runs deeper than one using a single extra SWAP placed to run in parallel. Larger circuits give far more room to diverge.

Research question

How large is the loss from optimising the SWAP-count proxy, can a router that also scores the downstream makespan recover it without giving up that objective, and does the shorter schedule raise success probability on real hardware?

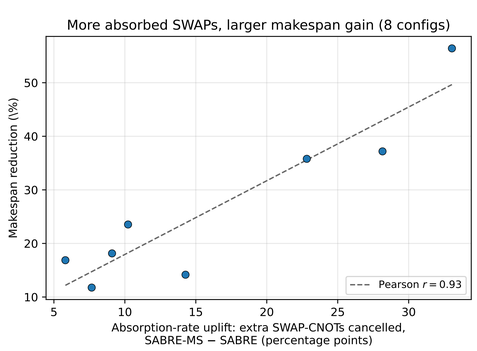


Figure 2. Mechanism (Section 3): more SWAPs absorbed, larger gain ($r = 0.93$).

2. Methodology

We fix every pipeline stage (Fig. 3) but routing: mapping is Qiskit's `SabreLayout`, and the makespan is read off Qiskit's standard ASAP scheduler, which starts each gate at the first cycle on which its qubits are free. With per-gate durations (CNOT 2, SWAP 6, single-qubit 1 cycle),

$$M \;=\; \max_g \big[\mathrm{start}(g) + \mathrm{dur}(g)\big], \quad \mathrm{start}(g) = \max_{q \in Q(g)} r_q,$$

where r_q is qubit q's next free cycle. A fixed scheduler isolates routing's effect on M.

SABRE-MS keeps SABRE's distance score $H(s)$ and adds one makespan term, so routing balances both costs. For a candidate SWAP s on (q_1, q_2),

$$H_{\mathrm{MS}}(s) \;=\; H(s) \;+\; \lambda \max\big(\mathrm{finish}[q_1],\, \mathrm{finish}[q_2]\big),$$

whose new term is the cycle the SWAP could start on. By steering routing toward SWAPs on qubits that are free now, it lets them run in parallel with ongoing gates instead of extending the critical path. The weight λ, chosen per circuit, sets the balance; $\lambda = 0$ recovers SABRE, at constant-factor cost.

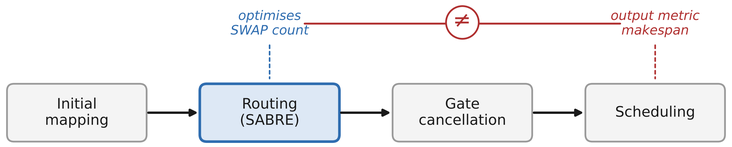
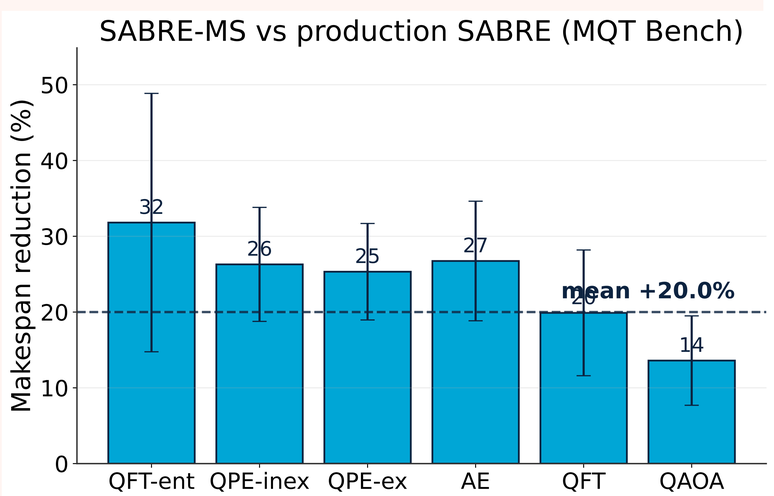


Figure 3. The compilation pipeline. SABRE-MS adds a makespan signal to the routing stage and leaves every other pass unchanged.

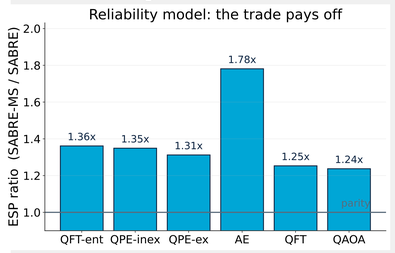
3. Results: makespan gain

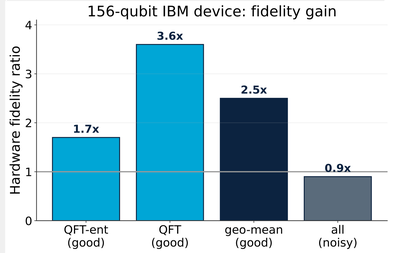
SABRE-MS cuts mean makespan by **20%** on MQT Bench and 13% on a synthetic set, against production SABRE, with the largest gains on deep, entangling families. It does not insert fewer SWAPs but places them better: across configurations the makespan gain tracks how many of its SWAPs the cancellation pass absorbs (Fig. 2, Pearson $r = 0.93$).



4. Results: reliability and hardware

Here we test whether the shorter schedule actually improves reliability, despite the extra SWAPs it costs. Charged in full against the ESP model it does: SABRE-MS improves 36 of 46 routing-heavy configurations (median 1.33×). ESP is only a model, so we confirm the gain on a 156-qubit IBM device, scoring each routing by its measured fidelity under a mirror test (U then $U^\dagger$, whose correct output is known). On low-error qubits, where decoherence sets the limit, SABRE-MS reaches 2.5× the fidelity of SABRE (geometric mean). On a noisier subgraph it falls to 0.9×: SABRE-MS improves only the decoherence term, which gate and readout error bury once they dominate.



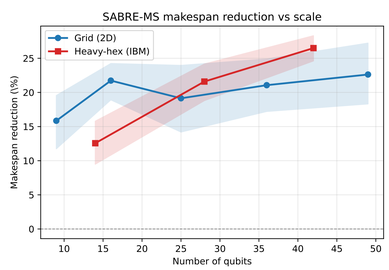


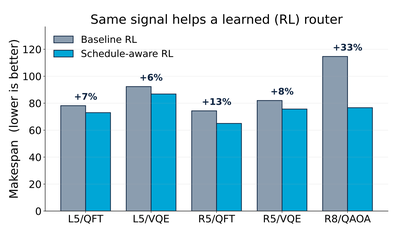
5. Results: scalability and generality

The reduction does not fade with size, holding near 20% out to 49 qubits. It is also not specific to SABRE. We add the same makespan signal to a structurally unrelated reinforcement-learning router (qgym): its observation gains the per-qubit finish times computed as in SABRE-MS, and its reward gains a terminal penalty on the final schedule length,

$$R_{\mathrm{sched}} \;=\; R_{\mathrm{SwapQuality}} \;-\; \lambda \cdot \mathrm{makespan}.$$

It shortens schedules on 8 of 9 tested configurations, so the benefit comes from the objective, not the algorithm.





6. Conclusions

A circuit's success probability turns on its gate count and its makespan, and on near-term hardware the makespan is the heavier cost. SABRE optimises a proxy for the gate count and acts on the makespan only by chance. Adding one schedule-aware term, alongside the SWAP-count term rather than in place of it, cuts mean makespan by 20% at constant-factor cost and raises measured hardware fidelity by 2.5×. Because the same signal helps a structurally unrelated learned router, the makespan belongs in the routing objective itself, not in any one algorithm.