POLITECNICO DI TORINO

Master degree course in Quantum Engineering

Master Degree Thesis

Hardware-Aware Qubit Mapping and Exact Routing for a Five-Qubit Superconducting Star QPU



Supervisor
Prof. Giovanna Turvani

Candidate
Mirko Ciardo

Co-supervisors:
Dott. Giacomo Vitali, Prof. Deborah Volpe

September 2026

Abstract

Quantum programs are expressed in terms of logical qubits, whereas a superconducting QPU provides a finite set of physical qubits connected by a restricted coupling graph. Compilation must therefore choose an initial placement and introduce routing operations whenever a logical two-qubit interaction is not directly executable. Since routing affects both gate reliability and execution time, minimising inserted SWAPs alone does not fully capture its hardware cost.

This thesis develops hardware-aware exact routing for a frozen five-qubit superconducting star model representing the IQM Spark configuration used by the Lagrange service. Routed circuits are evaluated with

$$J = -\ln(P_{\text{gates}}) + \frac{T_{\text{exec}}}{T_{\text{coh,eff}}}$$

where P_{gates} is the product of primitive-gate success probabilities and T_{exec} is the makespan of an as-soon-as-possible physical-resource schedule. A SABRE-inspired heuristic supplies a validated feasible upper bound, while an exact dynamic-programming search optimises J over a restricted action model containing direct, SWAP-and-execute and BRIDGE-and-execute transitions.

Two exact non-routing scheduling semantics are studied. Forced closure deterministically materialises the non-routing work prescribed by its closure rule before the next routing decision. JIT-exact instead advances through projected two-qubit progress, materialising only the missing prerequisites of the selected event and completing residual work deterministically. Queue exhaustion certifies the minimum J only within the route-and-schedule space defined by the selected semantics.

The thesis establishes two main theoretical results. First, a continuation-safe Max-Plus dominance relation uses $z_i = G + \kappa t_i$ to combine accumulated gate cost and physical-resource availability. For labels with the same future routing state, componentwise dominance guarantees a no-worse objective for every common complete continuation. Second, on the common supported domain, every canonical route R_P produced by the permutation-based minimum-SWAP router is representable within the JIT model, so JIT queue exhaustion implies

$$J_J^\star \leq J(R_P)$$

The final benchmark evaluates 70 five-qubit circuits with eight methods under one frozen hardware-aware evaluator. Both J-exact modes certify all 70 circuits. The

permutation-based router certifies the minimum SWAP count for 49 circuits and returns valid incumbents for the remaining 21 circuits when the time limit is reached. On the prespecified 28-circuit primary set, the paired signed-rank tests favour JIT over the permutation router, tket, and both MQT QMAP baselines. These results should be interpreted together with the reported bootstrap intervals and family-level robustness checks. The comparison with best-of-ten Qiskit SABRE-only gives a less clear result: JIT wins on 10 of the 28 primary circuits and loses on 18, and the signed-rank test does not detect a systematic advantage for JIT under this restart budget. In contrast, the seed-zero and deterministic-median controls favour JIT. The outcome therefore depends noticeably on the restart policy. Qiskit opt3 yields lower J than JIT on every primary and complete-corpus circuit, including after mirror-family exclusion, but its end-to-end optimisation and resynthesis prevent attributing this advantage specifically to routing. These results separate route quality, compilation policy and objective-specific exactness while preserving explicit certificate semantics.

Contents

Notation and Acronyms IX

1 Introduction 1
- 1.1 Compilation between an abstract circuit and a physical QPU 1
- 1.2 Why hardware-aware cost is necessary 2
- 1.3 The five-qubit star setting 2
- 1.4 Exact routing framework and scheduling semantics 3
- 1.5 Research questions 4
- 1.6 Scope and claims 4
- 1.7 Contributions of the thesis 5

2 Quantum Compilation and Routing Background 7
- 2.1 Logical circuits and physical execution 7
- 2.2 Superconducting transmon qubits 8
- 2.3 Connectivity as a graph 8
- 2.4 Dependency DAGs and ready operations 8
  - 2.4.1 Forced-closure scheduling 9
  - 2.4.2 JIT non-routing materialisation 10
- 2.5 Scheduling and makespan 10
- 2.6 Placement, mapping, routing and optimisation 11
- 2.7 Heuristic and exact perspectives 11
- 2.8 Position relative to prior mapping work 12
  - 2.8.1 Permutation-based minimum-SWAP exact routing for IQM Spark 12
- 2.9 Interchange through OpenQASM 14

3 Target Hardware: Lagrange and IQM Spark 15
- 3.1 Platform nomenclature and evidence boundary 15
- 3.2 Star coupling graph 15
- 3.3 Derived action parameters 16
- 3.4 Effective coherence scale 17

3.5 Consequences of heterogeneity . . . 17
3.6 What the hardware model omits . . . 18

4 The Qubit Mapping and Routing Problem 19
4.1 Input objects . . . 19
4.2 Progress and scheduling state . . . 20
4.2.1 Forced-closure progress semantics . . . 20
4.2.2 JIT projected progress . . . 20
4.3 Legal actions . . . 21
4.3.1 Forced-closure transition . . . 22
4.3.2 JIT-exact transition . . . 23
4.4 Feasible and optimal routed circuits . . . 23
4.5 Combinatorial structure . . . 24

5 Hardware-Aware Objective and Scheduling Semantics 25
5.1 From primitive success to additive gate cost . . . 25
5.2 Coherence-normalised duration . . . 26
5.3 Integer ticks and exact duration updates . . . 26
5.4 Incremental objective state . . . 27
5.5 Upper bounds, lower bounds and certificates . . . 27
5.6 Complete numerical example . . . 27
5.7 Model limitations . . . 28

6 Exact Routing Architecture and Scheduling Modes 29
6.1 End-to-end flow . . . 29
6.2 Parsing and normalisation . . . 30
6.3 Non-routing scheduling modes . . . 30
6.3.1 Forced-closure scheduling . . . 30
6.3.2 JIT prerequisite materialisation . . . 30
6.3.3 Terminal flush . . . 30
6.4 Authoritative candidate handoff . . . 31
6.5 Exact-search boundary . . . 31
6.6 Reconstruction and final validation . . . 31
6.7 Termination and certificate semantics . . . 31
6.8 Separation of concerns . . . 32

7 SABRE-Inspired Upper-Bound Construction 33
7.1 Original SABRE in context . . . 33
7.2 Upper-bound construction . . . 33
7.3 Score function terms and formulation . . . 34
7.3.1 Hardware-aware interaction matrix . . . 34
7.3.2 Front-layer and look-ahead costs . . . 35
7.3.3 Immediate action cost . . . 35

7.3.4 Consumed-gates term N_{consumed} . . . 35
7.3.5 Anti-undo penalty P_{undo} . . . 36
7.3.6 Total score . . . 36
7.4 Initial placement . . . 36
7.5 Production configuration . . . 36
7.6 Retained configuration . . . 36
7.7 Validation of the incumbent . . . 37
7.8 Role in the exact search . . . 37

8 Exact Dynamic-Programming Router 39
8.1 Common dynamic-programming machinery . . . 39
8.2 Forced-closure exact mode . . . 40
8.3 JIT-exact mode . . . 40
8.3.1 Projected progress and eligible events . . . 41
8.3.2 Prerequisite materialisation and routing successors . . . 41
8.3.3 Terminal completion . . . 41
8.3.4 JIT transition pattern . . . 42
8.3.5 Worked JIT-exact example . . . 42
8.4 Common frontier representation . . . 44
8.4.1 JIT progress determinacy . . . 44
8.4.2 Dynamic coordinates . . . 45
8.4.3 Stable frontier storage . . . 45
8.5 Transactional candidate commit . . . 45
8.6 Exact duplicates and dominance . . . 45
8.7 Lower-bound modes and incumbent pruning . . . 46
8.7.1 Bound-mode off . . . 46
8.7.2 Bound-mode cheap . . . 46
8.7.3 Effect on exactness and runtime . . . 46
8.8 Terminal completion and deterministic selection . . . 47
8.9 Main search loops . . . 47
8.10 Termination and timeout . . . 48
8.11 Why the exact search remains difficult . . . 49

9 Max-Plus Objective Dominance 51
9.1 Motivation . . . 51
9.2 Comparable labels . . . 51
9.2.1 JIT continuation-state equivalence . . . 52
9.3 Transformed objective coordinates . . . 52
9.4 Continuation-safe Max-Plus dominance . . . 52
9.5 Numerical dominance example . . . 54
9.6 Pruning relation and deterministic ties . . . 54
9.7 Binary64 interval guard . . . 54

9.8 Exactness scope and limitations . . . 55
9.9 Algorithmic significance . . . 55

10 Numerical Safety and Implementation Invariants 57
10.1 Why implementation details affect exactness . . . 57
10.2 Python and C++ boundary . . . 57
10.3 Lossless state representation . . . 58
10.4 Finite-precision pruning safety . . . 58
10.5 Atomic transitions and interruption safety . . . 58
10.6 Validation strategy . . . 58
10.7 Post-search constructive validation . . . 59

11 Correctness, Certificates and Complexity 61
11.1 Statement of the exactness claims . . . 61
11.2 Mode-specific completeness and scheduling invariants . . . 62
11.2.1 JIT progress determinacy . . . 62
11.2.2 Further JIT Completeness Obligations . . . 63
11.3 Shared reduction and implementation invariants . . . 63
11.3.1 Classic and Max-Plus dominance . . . 64
11.3.2 Reconstruction correctness . . . 66
11.4 Queue-exhaustion certificates . . . 66
11.4.1 Forced-closure theorem . . . 66
11.4.2 JIT theorem . . . 66
11.5 Relationships between the routing models . . . 67
11.5.1 Forced closure versus JIT-exact . . . 67
11.5.2 JIT-exact versus the permutation-based router . . . 67
Common supported domain. . . . 67
Paper-order representation . . . 68
Permutation-route embedding theorem . . . 68
Strictness and limits . . . 69
11.5.3 Forced closure versus the permutation-based IQM Spark router 69
11.6 Timeout semantics . . . 70
11.7 Complexity . . . 70
11.8 Evidence versus proof . . . 71

12 Experimental Methodology 73
12.1 Experimental goals and comparison scope . . . 73
12.2 Frozen corpus and experimental regimes . . . 74
12.3 Common hardware-aware evaluation . . . 75
12.3.1 ASAP scheduling and critical path . . . 75
12.3.2 Operation representation and gate counts . . . 76
12.4 Algorithms and budgets . . . 77
12.4.1 JIT, scheduling ablation and permutation baseline . . . 77

12.4.2 Qiskit opt3 and SABRE-only . . . 78
12.4.3 tket and MQT QMAP . . . 78
12.5 Validation and certificate semantics . . . 79
12.6 Paired analysis and uncertainty . . . 79
12.7 Timing protocol and reproducibility . . . 81

13 Experimental Results and Discussion 83
13.1 Campaign completeness and correctness . . . 83
13.2 Unified paired comparison against JIT . . . 84
13.2.1 Absolute predicted execution quality . . . 86
13.3 Circuit scale and decomposition of the objective . . . 87
13.4 Seed and mirror sensitivity . . . 89
13.5 Classical compilation cost on the common host . . . 91
13.6 Summary of empirical findings . . . 92

14 Conclusions and Future Work 93
14.1 Summary . . . 93
14.2 Answers to the research questions . . . 95
14.3 Main contributions and limitations . . . 96
14.4 Future work . . . 97
14.5 Closing perspective . . . 99

A Software Architecture and Reproducibility 101
A.1 Repository and Frozen Thesis Release . . . 101
A.2 Software Architecture . . . 102
A.3 Mapping Thesis Concepts to Source Modules . . . 104
A.4 Native Implementation Details . . . 108
A.5 Build and Reproduction Paths . . . 110
A.5.1 Core router installation . . . 110
A.5.2 Data-only reconstruction of the final analysis . . . 110
A.5.3 Executing or resuming a new benchmark . . . 111
A.6 Frozen Benchmark Artifacts and Validation . . . 111
A.7 External Benchmark Configuration Details . . . 113
A.7.1 tket configuration . . . 113
A.7.2 Qiskit configurations . . . 113
A.7.3 MQT QMAP adapter . . . 113

B Frozen Hardware Configuration 115
B.1 Interpretation of the effective coherence scale . . . 117

C Supplementary Correctness Proofs and Counterexamples 119
C.1 Supporting completeness proofs . . . 119
C.1.1 Initial-state completeness . . . 119

C.1.2 Forced-closure successor completeness 120
C.1.3 Forced-closure contraction/soundness 122
C.1.4 JIT prerequisite-materialisation soundness 123
C.1.5 JIT successor completeness 126
C.1.6 JIT terminal-flush soundness 128
C.2 Supporting reduction proofs 130
C.2.1 Common-continuations equivalence 130
C.2.2 Exact duplicate removal 132
C.2.3 Incumbent pruning 133
C.2.4 Classic Dominance 135
C.2.5 Reconstruction correctness 135
C.3 Queue-exhaustion proofs 137
C.3.1 Complete forced-closure theorem proof 137
C.3.2 Complete JIT theorem proof 138
C.4 Counterexamples between forced closure and JIT-exact 140
C.4.1 Counterexample forced closure better 140
C.4.2 Counterexample JIT better 141
C.4.3 Consequence 142
C.5 Counterexamples between forced closure and permutation-based router 143
C.5.1 Forced closure is better 143
C.5.2 Counterexample B: the permutation router is better 145
C.5.3 Consequence 146
C.6 Detailed complexity bounds 147
C.6.1 Time complexity 147
Forced-closure complexity 148
JIT-exact complexity 148
Absence of a polynomial input-size guarantee 149
C.6.2 Space complexity 150
C.7 Proof-obligation summary 151

Notation and Acronyms

This section collects the notation used repeatedly throughout the thesis. Symbols that occur only locally in a derivation are defined at their point of use.

Logical circuit, hardware and mappings

Symbol	Meaning
ℓ_i	Logical qubit i, an identity in the input circuit.
q_j	Physical qubit j, a calibrated hardware resource.
$L = \{\ell_0, \ldots, \ell_4\}$	Logical-qubit set for the five-qubit circuits considered in this thesis.
$Q = \{q_0, \ldots, q_4\}$	Physical-qubit set of the five-qubit star target; q_2 is the physical centre.
π	Logical-to-physical bijection.
m	Integer rank of a logical-to-physical mapping, with $m \in \{0, \ldots, 119\}$.
$\mathcal{D} = (V, E)$	Normalised logical operation dependency DAG.
$X \subseteq V$	Set of logical DAG events already executed or materialised, where this representation is used.
$\mathcal{R}(X)$	Set of unexecuted DAG events whose predecessors are contained in X.
$\mathcal{H} = (Q, C)$	Physical coupling graph.
C	Native undirected CZ edge set $\{\{q_2, q_i\} : i \in \{0,1,3,4\}\}$.

Symbol	Meaning
$D_{2Q} = (V_{2Q}, E_{2Q})$	Projected dependency DAG containing the routing-relevant two-qubit events and their induced precedence relations.
V_{NR}	Set of supported non-routing events, $V_{\mathrm{NR}} = V \setminus V_{2Q}$.

Objective and scheduling

Symbol	Meaning
e_g	Frozen error probability of physical primitive gate g.
τ_g	Frozen duration of physical primitive gate g, in seconds.
δ	Exact scheduler tick duration, in seconds.
t_i	Availability time of physical qubit q_i, expressed in integer scheduler ticks.
c	Centre availability t_2 in the compact centre-relative scheduler representation.
d_i	Centre-relative leaf availability $t_i - t_2$.
P_{gates}	Product of the success probabilities of the emitted primitive gates.
G	Accumulated negative-log gate cost, $G = -\ln P_{\mathrm{gates}}$.
$T_{1,i}, T_{2,i}$	Frozen relaxation and dephasing times associated with physical qubit q_i.
$T_{\mathrm{coh,eff}}$	Effective circuit-level coherence scale, $$T_{\mathrm{coh,eff}}^{-1} = \sum_i \left[(4T_{1,i})^{-1} + (2T_{2,i})^{-1}\right]$$
κ	Conversion factor between scheduler ticks and the time contribution to the objective, $\kappa = \delta / T_{\mathrm{coh,eff}}$.

Symbol	Meaning
T_{exec}	Final ASAP execution makespan, $T_{\mathrm{exec}} = \delta \max_i t_i$.
J	Hardware-aware route objective, $$J = -\ln P_{\mathrm{gates}} + \frac{T_{\mathrm{exec}}}{T_{\mathrm{coh,eff}}} = G + \kappa \max_i t_i$$
U	Authoritative objective value of the best replay-valid complete incumbent.
$\mathrm{LB}(u)$	Admissible lower bound on the objective of every complete continuation of partial label u.
$\mathrm{LB}_{\mathrm{off}}(u)$	Current-objective lower bound used when optional residual lower bounds are disabled.
$\mathrm{LB}_{\mathrm{cheap}}(u)$	Admissible inexpensive residual lower bound used by `bound-mode=cheap`.
$z_i(u)$	Max-Plus objective coordinate of label u on physical resource i, $z_i(u) = G_u + \kappa t_{u,i}$.

Exact-search state and routing actions

Symbol	Meaning
u, v	Committed exact-search labels. Their logical-progress component is mode-specific.
p_u	Logical-progress descriptor stored by label u. In forced-closure mode it denotes progress at a deterministic closure fixed point; in JIT-exact mode it denotes projected two-qubit progress.
$K(u) = (p_u, m_u)$	Discrete frontier key: mode-specific logical progress together with encoded mapping.

Symbol	Meaning
$\mathcal{F}_K$	Active set of mutually incomparable labels associated with frontier key K.
bits(G)	Exact binary64 bit pattern of accumulated gate cost used where exact dynamic duplicate identity is required.
s	Stable logical arena slot used to identify a committed label independently of active-frontier lifetime.
γ	Priority-queue generation counter used for lazy invalidation of stale entries.
$C_{\mathrm{FC}}(u)$	Deterministic forced-closure operator applied to search state u.
S_a, S_b	The two operand-to-centre SWAP-and-execute alternatives for a blocked leaf–leaf two-qubit interaction.
$B_{a,b}$	Fixed mapping-preserving BRIDGE-and-execute construction for a blocked interaction between leaves a and b.

JIT-exact notation

Symbol	Meaning
$C_{2Q}(p)$	Set of routing-relevant two-qubit events already consumed at projected JIT progress p.
$N_{\mathrm{done}}(p)$	Set of non-routing events that must already have been materialised at projected progress p under the canonical JIT rules.
$N_{\mathrm{pending}}(p)$	Residual non-routing event set, $V_{\mathrm{NR}} \setminus N_{\mathrm{done}}(p)$.

Symbol	Meaning
$\Phi(p)$	Non-routing materialisation state implied by JIT projected progress, $\Phi(p) = \Big(N_{\text{done}}(p), N_{\text{pending}}(p)\Big)$
$R_{2Q}(p)$	Set of unconsumed routing-relevant two-qubit events eligible at projected progress p. When the state rather than its progress is the argument, the notation $R_{2Q}(u)$ is used equivalently.
$P(g, p)$	Set of still-unmaterialised non-routing prerequisites that must be materialised before eligible two-qubit event g can be routed from progress p.
$F_{\text{term}}(u)$	Deterministic JIT terminal-flush operator that materialises all remaining supported non-routing work after projected two-qubit progress is complete.

Routing-model and correctness notation

Symbol	Meaning
E	Forced-closure exact scheduling mode.
J (mode index)	JIT-exact scheduling mode when used as a mode index $M \in \{E, J\}$; this use is distinct from the scalar objective J.
$\mathcal{R}_E(C)$	Complete routed-and-scheduled traces admitted by the forced-closure exact model for circuit C.
$\mathcal{R}_J(C)$	Complete routed-and-scheduled traces admitted by the JIT-exact model for circuit C.

Symbol	Meaning
$J_E^\star(C)$	Certified optimum $$J_E^\star(C) = \min_{R \in \mathcal{R}_E(C)} J(R)$$ defined operationally when the forced-closure search reaches `queue_exhausted`.
$J_J^\star(C)$	Certified optimum $$J_J^\star(C) = \min_{R \in \mathcal{R}_J(C)} J(R)$$ defined operationally when the JIT-exact search reaches `queue_exhausted`.
$\mathcal{C}_M(u)$	Set of legal symbolic continuations of label u under exact scheduling mode $M \in \{E, J\}$.
$\mathcal{D}_\cap$	Common supported circuit domain on which the JIT model and the faithful permutation-based IQM Spark reconstruction (introduced by [1]) satisfy the assumptions required for their formal comparison.
ω	Dependency-valid topological order of routing-relevant two-qubit events used by the permutation-based router.
$L_c^\star(\omega)$	Optimal logical-centre sequence selected by the permutation-based method for a fixed two-qubit order ω, according to its inserted-SWAP objective.
$R_P(\omega)$	Canonical complete route reconstructed by the permutation-based router for two-qubit order ω.
R_P	Final canonical route selected by the permutation-based exact router.
$J_P(C)$	Authoritative post-hoc hardware-aware evaluation $J(R_P(C))$ of the canonical permutation-based route. The notation $J_P^\star$ is deliberately not used because the permutation-based router optimises inserted SWAP count rather than J.

Frequently used acronyms and terms

Acronym or term	Meaning
ASAP	As soon as possible; resource-scheduling rule used to compute the routed circuit makespan.
BRIDGE	Mapping-preserving routing construction used to execute a blocked leaf–leaf two-qubit interaction through the physical centre.
CZ	Controlled-Z gate.
DAG	Directed acyclic graph.
DP	Dynamic programming; in this thesis, a best-first label-setting exact search with frontier merging and dominance.
FC	Forced closure; the eager exact non-routing scheduling semantics.
JIT	Just in time; exact scheduling mode in which independent non-routing work may remain pending until required by a selected two-qubit event.
NISQ	Noisy intermediate-scale quantum.
OpenQASM	Open Quantum Assembly Language; OpenQASM 2 is the supported project input/output circuit syntax.
QPU	Quantum processing unit.
RSS	Resident set size; used as an operating-system measure of process memory.
SABRE	SWAP-based bidirectional heuristic search introduced by [2]; this thesis uses a distinct project-specific SABRE-inspired incumbent generator.
SWAP	Physical operation, or routing-action component, that exchanges the logical identities assigned to two adjacent physical qubits.

Acronym or term	Meaning
`queue_exhausted`	Termination status indicating that the active exact search has been completely exhausted and that the mode-specific optimum is certified.
timeout	Termination without complete exhaustion of the exact search. A validated complete incumbent may still be reported, but it is not an exactness certificate.

Chapter 1

Introduction

1.1 Compilation between an abstract circuit and a physical QPU

A gate-model quantum algorithm is normally expressed without committing each logical qubit to one particular device location. This abstraction is useful: the circuit describes the intended unitary transformation and measurement pattern, while the compiler accounts for the gate alphabet, connectivity, timing and calibration of a concrete quantum processing unit (QPU). The separation is also unavoidable on current noisy intermediate-scale quantum (NISQ) systems, whose small, heterogeneous and imperfect resources differ substantially from the ideal circuit model [3].

The abstraction breaks at every logical two-qubit operation. Such an operation can be issued directly only if its two logical operands currently occupy a coupled pair of physical qubits. When they do not, a compiler must change the placement, for example by inserting adjacent SWAP operations, or use a topology-specific decomposition that mediates the interaction. The compiler must respect the original dependency order while choosing among many possible initial placements and routing sequences. This is the qubit mapping and routing problem [4, 5].

The problem is combinatorial even for modest circuits. A five-qubit device has $5! = 120$ bijective initial placements. Every blocked two-qubit gate can create several successor mappings, and those choices alter the cost and legality of later gates. A locally inexpensive decision may therefore be globally poor. General allocation and routing formulations encompass NP-hard subproblems [6, 7]; exact enumeration has exponential worst-case growth, whereas a fast heuristic does not ordinarily provide a proof that a better route is impossible.

1.2 Why hardware-aware cost is necessary

Counting inserted SWAPs is a useful structural proxy, but it is not a complete physical objective. On a calibrated superconducting QPU, edges can have different controlled-gate error probabilities and durations, and qubits can have different relaxation and dephasing times [8, 9, 10, 11]. Two routes with the same number of SWAPs can consequently have different expected gate success and different exposure to decoherence. A bridge-like implementation may execute more primitive gates than a SWAP-based alternative yet preserve a strategically useful mapping. Conversely, a route with fewer gates can have a longer scheduled makespan if it serialises work on a heavily used central qubit. These effects motivate compilation objectives that account explicitly for calibration quality, latency, and resource timing rather than gate count alone [12, 13, 14].

The router studied in this thesis evaluates a route with the scalar objective

$$J = -\ln(P_{\text{gates}}) + \frac{T_{\text{exec}}}{T_{\text{coh,eff}}} \tag{1.1}$$

where P_{gates} is the product of primitive-gate success probabilities, T_{exec} is the makespan of an as-soon-as-possible (ASAP) resource schedule, and $T_{\text{coh,eff}}$ is an effective coherence normalisation derived from the frozen T_1 and T_2 values of all five qubits. The negative logarithm converts the multiplicative success model into an additive gate cost. Equation (1.1) does not claim to be a complete physical noise model; it is the precisely implemented optimisation criterion against which exactness is established.

1.3 The five-qubit star setting

The target configuration is a five-qubit star: physical qubit q_2 is the centre and q_0, q_1, q_3, q_4 are leaves. Controlled-Z gates are native only between the centre and a leaf. A logical interaction between two leaves is therefore blocked under the current mapping. The star is small enough to support an exact study, but it is not trivial. The centre is simultaneously a communication resource, a scheduling bottleneck and a mapping position of high strategic value.

The platform names require care. IQM Spark is the five-qubit superconducting hardware product model described by its manufacturer [15]. Lagrange is the locally deployed service and system identity associated with the installation jointly acquired by LINKS Foundation, Politecnico di Torino and INRiM and operated through the LINKS service [16, 17, 18].

1.4 Exact routing framework and scheduling semantics

The J-exact framework separates feasible incumbent construction from exact proof. A project-specific SABRE-inspired heuristic provides a candidate upper bound, which is accepted only after authoritative validation and recomputation of J. The exact search is therefore independent of how the incumbent is constructed: the heuristic can accelerate the proof, but it does not determine the exact search space or the meaning of the optimality certificate.

The exact router supports two non-routing scheduling semantics. In the original forced-closure mode, every ready non-routing operation is materialised deterministically, and closure is repeatedly applied until a routing decision is reached. This keeps the search state compact, but the resulting order of independent non-routing operations becomes part of the restricted exact model. The JIT-exact mode removes this forced scheduling choice. Search progress is represented primarily by the executed two-qubit routing events, while independent ready non-routing operations are allowed to remain pending. When a two-qubit event is selected, the kernel materialises only the still-unexecuted non-routing predecessors required for that event before evaluating every legal routing action. Any non-routing operations that remain after all two-qubit events have been completed are materialised by a deterministic terminal flush. The JIT-exact mode is therefore used as the reference configuration for the final experimental evaluation, while the forced-closure mode is retained as a separate exact baseline.

One result developed in this thesis is a continuation-safe Max-Plus dominance relation for the exact search. For a label u, let G_u denote the accumulated gate cost and let $t_{u,i}$ denote the availability time of physical resource i. The corresponding coordinates are defined as

$$z_i(u) = G_u + \kappa t_{u,i}$$

Consider two labels with the same remaining obligations. If one label dominates the other componentwise in these coordinates, then the dominated label cannot attain a lower objective value under any complete continuation common to both labels. This criterion also applies in some cases where standard resource-wise dominance does not. In particular, a larger accumulated gate cost can be offset by sufficiently earlier resource availability.

A second main result relates the JIT-exact model to the permutation-based minimum-SWAP router used as an exact reference in this thesis. On their common supported domain, every canonical route produced by the permutation-based router is representable within the JIT route-and-schedule space. Therefore, whenever the JIT

search reaches `queue_exhausted`, its certified optimum satisfies

$$J_J^\star \leq J(R_P)$$

where R_P is the canonical route returned by the permutation-based router and evaluated post hoc with the hardware-aware objective J.

1.5 Research questions

The study is organised around four questions.

1. How can error probability, scheduled execution time and coherence be combined into a deterministic hardware-aware objective that can be evaluated incrementally?
2. Under which conditions can accumulated gate cost and physical-resource availability be combined into a dominance relation that safely removes exact search labels without discarding any potentially improving continuation?
3. How can non-routing operations be deferred without imposing the forced-closure scheduling restriction, while retaining a compact exact state representation and a certifiable optimum?
4. How does the JIT-exact route space relate formally to the permutation-based minimum-SWAP router on their common supported domain, and how does JIT compare empirically with exact, heuristic and end-to-end compilation baselines under the common hardware-aware evaluator?

1.6 Scope and claims

The input language accepted by the studied path is normalised OpenQASM 2 with the supported one-qubit operations, controlled-Z, barriers and terminal measurements. The exact router supports two distinct non-routing scheduling models. In the forced-closure model, every ready non-routing operation is materialised deterministically before the next routing decision. In the JIT-exact model, independent non-routing operations may remain pending until they become prerequisites of a selected two-qubit event, while any residual non-routing operations are completed by a deterministic terminal flush after the last two-qubit event.

In both models, the routing action set permits direct centre–leaf execution and, for a blocked leaf–leaf controlled-Z, exactly two SWAP-and-execute choices and one bridge-and-execute choice. It does not permit arbitrary proactive SWAPs, alternative pulse synthesis, commuting-gate reordering outside the supported dependency semantics, or every unitary identity available to a general compiler.

Let $\mathcal{R}_{\mathrm{FC}}$ denote the set of routed circuits reachable under the forced-closure scheduling semantics, and let $\mathcal{R}_{\mathrm{JIT}}$ denote the corresponding set under the JIT-exact semantics. The exact solutions certified by the two modes are therefore

$$R^{\star}_{\mathrm{FC}} \in \arg\min_{R \in \mathcal{R}_{\mathrm{FC}}} J(R) \qquad \text{and} \qquad R^{\star}_{\mathrm{JIT}} \in \arg\min_{R \in \mathcal{R}_{\mathrm{JIT}}} J(R).$$

No general set-inclusion relation between $\mathcal{R}_{\mathrm{FC}}$ and $\mathcal{R}_{\mathrm{JIT}}$ is assumed: the two modes impose different deterministic scheduling semantics on non-routing operations.

Accordingly, "exact" in this thesis means globally optimal with respect to the implemented normalisation, dependency model, objective, selected non-routing scheduling semantics and restricted routing action set. It does not mean globally optimal over all physical pulse schedules or all mathematically equivalent quantum circuits. A timeout returns a feasible incumbent and search diagnostics, not a proof of optimality. A completed `queue_exhausted` run is the relevant certificate and proves optimality within the corresponding restricted model.

The experimental evidence is also bounded. It supports claims about the specific circuit corpus, frozen hardware configuration and evaluation protocol used in this thesis, not every possible circuit or QPU.

1.7 Contributions of the thesis

This thesis develops two scientific contributions within a formally specified exact-routing framework and evaluates them through a reproducible experimental study.

- **Continuation-safe Max-Plus dominance.** A dominance relation is defined using the transformed coordinates

  $$z_i = G + \kappa t_i.$$

  For labels with the same logical progress and physical mapping, componentwise dominance in these coordinates guarantees an objective value no larger under any complete continuation available to both labels. This gives a sound pruning criterion that also applies in cases where standard componentwise dominance on resource-availability times does not.

- **Embedding of the permutation-based router into the JIT model.** On the common supported domain, the thesis proves that every canonical route generated by the permutation-based minimum-SWAP router is representable within the JIT-exact route-and-schedule space. Consequently, queue exhaustion of the JIT search certifies

  $$J^{\star}_{J} \leq J(R_P).$$

This establishes a formal objective relation between two exact formulations that optimise different primary criteria.

- **JIT-exact scheduling semantics.** The thesis identifies forced materialisation of ready non-routing operations as a modelling restriction of the original exact search and develops a JIT semantics in which independent non-routing work may remain pending. The associated progress-determinacy result shows that future non-routing obligations are determined by projected two-qubit progress, allowing the additional scheduling freedom to be represented compactly.
- **Hardware-aware exact-routing formulation.** The restricted five-qubit star-routing problem is formulated with explicit mapping transitions, exact ASAP resource scheduling and the hardware-aware objective J. The scope of both forced-closure and JIT optimality certificates is stated explicitly, separating model-specific exactness from unrestricted physical circuit optimisation.
- **Controlled eight-method empirical evaluation.** The final benchmark evaluates 70 circuits from seven structured families using JIT-exact, forced closure, the permutation-based exact router, Qiskit opt3, Qiskit SABRE-only, tket, MQT QMAP heuristic and MQT QMAP exact under one frozen hardware-aware evaluator. The analysis separates route quality from algorithm-specific certificate semantics and includes paired inference, multiple-comparison control, seed-policy sensitivity and family-block robustness checks.

Chapter 2

Quantum Compilation and Routing Background

2.1 Logical circuits and physical execution

A quantum circuit specifies operations on logical qubits without assigning those logical identities to particular device elements [19]. A gate such as $CZ(\ell_0, \ell_3)$ states which logical qubits interact, but not where they are physically stored.

A physical QPU has a finite set of physical qubits $Q = \{q_0, \ldots, q_{m-1}\}$, a native gate set, and constraints on which operations can be executed simultaneously. A mapping π assigns each live logical qubit to a different physical qubit. For the width-five circuits considered in this work, π is a bijection.

As an example, suppose that $\pi(\ell_0) = q_2$ and $\pi(\ell_3) = q_0$. A logical CZ acting on ℓ_0 and ℓ_3 can then be executed directly, since q_2 and q_0 are adjacent in the fixed star topology. If the same logical operands are instead mapped to q_0 and q_1, the gate is not directly executable and a routing operation is required.

The routed output is still a circuit. It contains native physical operations and a traceable initial and final mapping. The final mapping matters because measurements and any downstream interpretation must associate physical readout bits with the correct logical identities. A sequence can be hardware-legal yet semantically wrong if this permutation is lost.

2.2 Superconducting transmon qubits

The target belongs to the superconducting transmon family, for which calibrated gate quality, operation duration and coherence times are relevant to circuit compilation [20, 8, 9]. In this thesis, T_1 and T_2, primitive-gate error estimates and operation durations are used as engineering parameters rather than as a complete device-noise model: they do not capture correlated noise, leakage, crosstalk or coherent control effects. The routing objective therefore uses a deterministic surrogate that combines primitive success factors with scheduled duration relative to an effective coherence scale. Its precise definition is given in Chapter 5.

2.3 Connectivity as a graph

The hardware connectivity is described by the coupling graph $\mathcal{H} = (Q, C)$. Its vertices represent physical qubits, while an edge between two vertices indicates that the corresponding qubits support the native two-qubit gate considered here. For some gate sets, the direction of the edge must also be taken into account. Sparse connectivity reduces the number of direct physical connections, but it can require additional routing during compilation.

The effect of the topology depends on the structure of the coupling graph. On a path, two logical qubits mapped to distant physical qubits may have to be brought together through several adjacent SWAPs. On a grid, there can be more than one shortest route between the same pair of qubits. The star topology considered in this thesis has a different structure. Each leaf is at distance one from the centre and at distance two from every other leaf.

Although the diameter of the star is only two, interactions between qubits located on different leaves must use the centre. The centre can therefore be required by several non-local interactions at different points in the schedule. Graph distance alone is not sufficient to determine which logical qubit should be mapped to the centre. That choice also depends on the remaining interactions, the calibrated costs of the physical edges, and the current schedule.

2.4 Dependency DAGs and ready operations

The textual order of a circuit is one serialisation of a partial order. Two operations on disjoint qubits may be independent and can often be scheduled in parallel. A compiler captures the required order with a directed acyclic graph $\mathcal{D} = (V, E)$. Each vertex is a normalised operation. An edge $(u, v) \in E$ means that u must complete before v may execute, usually because the operations share a logical resource and their order is semantically significant. Given a set $X \subseteq V$ of nodes that have

already been executed, the ready set is defined as

$$\mathcal{R}(X) = \{v \in V \setminus X : \mathrm{pred}(v) \subseteq X\}. \tag{2.1}$$

This definition concerns logical readiness only. A ready one-qubit gate can be executed on the physical qubit that currently holds its logical operand. A ready two-qubit gate may instead remain unavailable if the two operands are mapped to physical qubits that are not connected by the required coupling edge. The routing state must therefore include both the set of executed DAG nodes and the current logical-to-physical mapping.

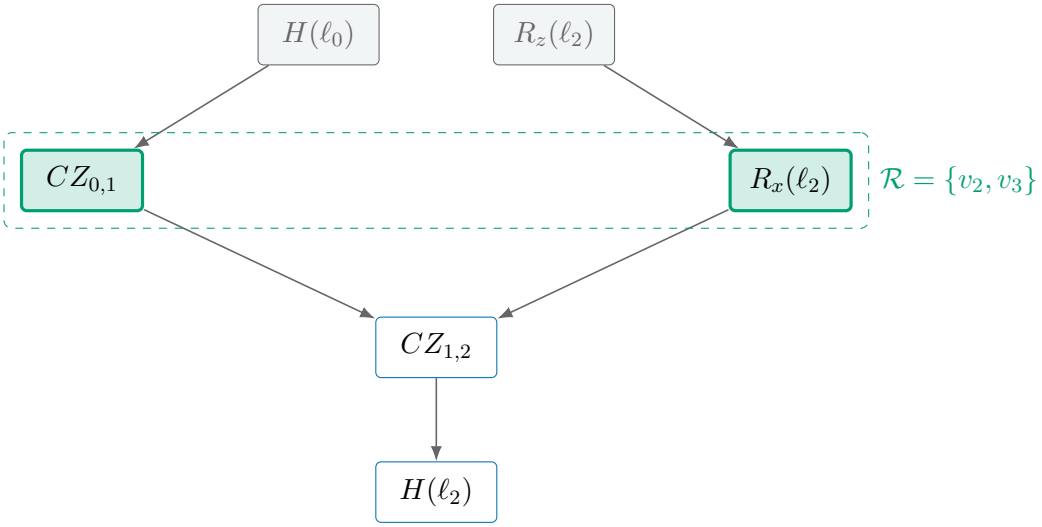


Figure 2.1: A small dependency DAG. Grey operations have already been executed, while the highlighted nodes form the ready set. Readiness describes logical availability only: under forced closure, ready non-routing work is materialised deterministically, whereas JIT scheduling may leave independent ready operations pending until they are required or until terminal completion.

2.4.1 Forced-closure scheduling

The original exact model handles operations that introduce no routing choice through a deterministic forced-closure operator $C_{\mathrm{FC}}(u)$. Starting from a search state u, ready non-routing operations are materialised according to a fixed deterministic order until either a routing decision or circuit completion is reached. This contraction keeps the search representation compact, but it also makes that deterministic ordering part of the restricted exact model. Consequently, forced-closure exactness refers to optimisation over routes generated under this scheduling semantics, not over every valid ordering of independent non-routing operations.

2.4.2 JIT non-routing materialisation

The JIT-exact mode instead projects logical progress onto routing-relevant two-qubit events. Let $D_{2Q} = (V_{2Q}, E_{2Q})$ denote the corresponding projected dependency DAG. When an eligible two-qubit event g is selected at progress p, only the still-unmaterialised non-routing predecessors $P(g, p)$ required to enable g are materialised before its routing action. Ready non-routing operations independent of g remain pending, and a deterministic terminal flush completes any residual non-routing work after all two-qubit events have been consumed.

This distinction matters for the hardware-aware objective because the logical-to-physical mapping may change before a deferred operation is materialised, so the same logical operation can acquire a different physical gate cost and interact differently with the ASAP schedule. Forced closure and JIT define two different restricted exact scheduling models. JIT-exact is used as the reference configuration in the final evaluation, whereas forced closure is reported as a separate exact baseline. The formal definitions and a worked example are given in Chapters 4 and 8.

2.5 Scheduling and makespan

Routing determines the sequence of physical operations. Scheduling then assigns a start time to each operation while respecting dependency constraints and conflicts on physical resources. The distinction is important. Classical routing time measures how long the compiler runs. Quantum execution time measures the duration of the resulting physical circuit. The latter is denoted T_{exec} in this thesis.

The router uses an ASAP list schedule. If an operation g of duration τ_g occupies resource set R_g, its start time is

$$s_g = \max_{q \in R_g} a_q, \qquad a_q \leftarrow s_g + \tau_g \quad \text{for each } q \in R_g \tag{2.2}$$

where a_q is the current availability of physical qubit q. Resources not used by g retain their previous availability. The final makespan is $\max_q a_q$. This model captures parallel operations on disjoint qubits and serialisation through the star centre. Related temporal planning and physical-operation scheduling formulations appear in [21, 22].

This is an operation-level resource model rather than a general pulse scheduler, so the exactness claims are relative to the scheduling model defined here.

2.6 Placement, mapping, routing and optimisation

Terminology in the literature is not completely uniform [6]. This thesis distinguishes between several related concepts. *Initial placement* specifies the first bijection between logical and physical qubits. The term *mapping* denotes the current bijection during the routing process. *Routing* selects and inserts physical operations so that each logical two-qubit gate can be executed without changing the circuit semantics. *Scheduling* assigns execution times to the resulting physical operations. *Optimisation* selects among feasible routed schedules according to the objective J.

A SWAP applied to adjacent physical qubits q_i and q_j exchanges the logical qubits currently mapped to those two vertices. If π is represented as a logical-to-physical tuple, its update is the transposition of the two corresponding physical locations. The SWAP is also a physical operation with its own error and duration. In the frozen model it is derived from controlled and single-qubit primitives.

A bridge construction instead executes a blocked two-qubit interaction through a common neighbour without changing the logical-to-physical mapping [23, 24]. For leaf operands a and b with centre c, the project-specific CZ action used in this thesis is the fixed sequence

$$CZ(c,b), H(c), CZ(a,c), H(c), CZ(c,b), H(c), CZ(a,c), H(c)$$

The template trades additional primitive operations for mapping stability. It is part of the restricted action model studied here and is not claimed as a universal decomposition for arbitrary native gate sets.

2.7 Heuristic and exact perspectives

Heuristic and exact routing address complementary requirements. SABRE uses front-layer and look-ahead costs to guide routing without exploring the full search space [2]. The incumbent generator used in this project follows the same general approach, but it is adapted to the five-qubit star topology.

J-exact uses a complete label-setting search over its restricted routing space. In the architecture considered here, the heuristic is used only to provide a validated feasible upper bound. The exact search then tests whether a route with a lower value of J exists. If the search queue is exhausted, the best route found is certified as optimal within the selected exact model.

2.8 Position relative to prior mapping work

Qubit mapping and routing have been formulated through compiler, graph and search methods in which initial placement and subsequent routing are coupled by hardware connectivity [4, 25, 26, 27, 5]. Heuristic approaches such as SABRE trade optimality guarantees for scalability by using front-layer and look-ahead information rather than exploring the complete mapping state graph [2]. The project-specific heuristic used here follows this general philosophy but is specialised to the five-qubit star and includes calibrated error and duration information in addition to topology.

Hardware-aware mapping work further motivates objectives beyond gate or SWAP count. Variability-, noise-, timing- and latency-aware compilation methods have shown that calibrated qubit and gate properties can materially alter mapping choices [10, 11, 12, 13, 14, 28, 24]. The objective used in this thesis is more specific: primitive success factors and scheduled execution time enter a fixed deterministic scalar J, making incremental exact comparison possible.

Exact mapping and routing have also been addressed through integer programming, temporal planning, satisfiability, MaxSAT and optimal-search formulations [6, 22, 29, 30, 31, 32, 33, 34]. Their optimality certificates are necessarily relative to their action and cost models. In particular, a SWAP-only exact formulation need not contain the mapping-preserving BRIDGE action allowed by J-exact, whereas a more general compiler may admit proactive SWAPs that the restricted J-exact model excludes.

2.8.1 Permutation-based minimum-SWAP exact routing for IQM Spark

Of particular relevance to this thesis is the exact routing method for the five-qubit IQM Spark star architecture introduced by [1]. The method exploits the special structure of a star topology and formulates the input routing problem in terms of a DAG of two-qubit gates. Let

$$D_{2Q} = (V_{2Q}, E_{2Q})$$

denote this dependency DAG, and let

$$\omega = (g_{\omega_1}, g_{\omega_2}, \ldots, g_{\omega_n})$$

denote a feasible topological ordering of its n two-qubit gates. The symbol ω is used here instead of the notation adopted in [1], since π is reserved in this thesis for a logical-to-physical mapping.

The key simplification follows from the star connectivity. For the execution of a two-qubit gate, at least one of its logical operands must occupy the physical centre. If neither operand is currently at the centre, at most one centre–leaf SWAP is required to place one of them there. Consequently, for a fixed gate order ω, the relevant routing information can be reduced to the sequence

$$L_c(\omega) = (c_{\omega_1}, c_{\omega_2}, \ldots, c_{\omega_n})$$

where c_{ω_k} is the logical qubit occupying the physical centre during the execution of gate g_{ω_k}. A change in consecutive elements of L_c corresponds to one inserted SWAP. The routing objective of [1] can therefore be expressed as

$$N_{\mathrm{SWAP}} = \sum_{k=1}^{n-1} \mathbf{1}\left[c_{\omega_k} \neq c_{\omega_{k+1}}\right]$$

Rather than enumerating all possible centre choices for a fixed ω, the method applies a gate-streak rule. The current centre is retained whenever it is an operand of the next gate; otherwise, the operand that can remain at the centre for the longest consecutive sequence of upcoming gates is selected. For a fixed feasible order ω, this produces an optimal centre sequence $L_c^\star(\omega)$ and its SWAP count in $\Theta(n)$ time [1].

The remaining combinatorial search is over the feasible topological orders of the two-qubit DAG. Let Ω denote this set. The algorithm evaluates the optimal centre sequence for every $\omega \in \Omega$ and selects an order

$$\omega^\star \in \arg\min_{\omega\in\Omega} N_{\mathrm{SWAP}}(\omega, L_c^\star(\omega)) \,.$$

The canonical complete route reconstructed from this choice is denoted $R_P = R_P(\omega^\star)$. Although a fixed order is evaluated in linear time, the complete algorithm is not polynomial in general because the number of feasible topological orders can grow exponentially with the dependency structure.

This objective differs fundamentally from the one optimised by J-exact. The method of [1] solves

$$\min N_{\mathrm{SWAP}}$$

whereas J-exact solves

$$\min J = \min\left(-\ln P_{\mathrm{gates}} + \frac{T_{\mathrm{exec}}}{T_{\mathrm{coh,eff}}}\right)$$

A SWAP-optimal route is therefore not necessarily optimal under J: different centre choices and physical mappings may expose the circuit to different calibrated gate

errors and durations. Conversely, the J-exact action model also admits bridge-and-execute transitions, which are not part of the SWAP-only routing model of [1]. These differences make the method particularly useful as a controlled exact reference while preventing its optimality certificate from being interpreted as a certificate for the objective used in this thesis.

For comparison, complete routes returned by both formulations are evaluated under the same frozen hardware model and the common objective J, but their certificates remain distinct: the permutation-based method certifies minimum inserted SWAP count within its routing formulation, whereas a completed J-exact search certifies minimum J within its selected scheduling and action model. The formal relationship between the canonical route R_P and the JIT-exact route space is established in Chapter 11.

2.9 Interchange through OpenQASM

The studied routing path uses a constrained OpenQASM 2 interface [35]. Input circuits are normalised into the supported gate basis and dependency model, and routed OpenQASM is emitted only after reconstruction. The implementation does not support the full language capabilities available in later OpenQASM specifications [36]; unsupported control flow, resets or gate definitions therefore lie outside the exactness domain considered in this thesis.

Chapter 3

Target Hardware: Lagrange and IQM Spark

3.1 Platform nomenclature and evidence boundary

Three layers of hardware description are distinguished in this thesis. *IQM Spark* is the manufacturer's product name for the five-qubit superconducting quantum computer [15]; *Lagrange* identifies the local system and managed service associated with the Turin deployment, jointly acquired by LINKS Foundation, Politecnico di Torino and INRiM and managed by LINKS Foundation [16, 37, 18, 17]; finally, all numerical routing and benchmarking results use the frozen hardware snapshot with signature `9da78e`, reported in full in Appendix B.

3.2 Star coupling graph

The physical qubit set is

$$Q = \{q_0, q_1, q_2, q_3, q_4\},$$

with centre q_2, leaves $L = \{q_0, q_1, q_3, q_4\}$, and undirected controlled-Z coupling set

$$C = \{\{q_2, q_0\}, \{q_2, q_1\}, \{q_2, q_3\}, \{q_2, q_4\}\}. \tag{3.1}$$

Figure 3.1 shows this graph with an example of CZ metrics.

Every native two-qubit gate uses the centre. A centre–leaf logical interaction is direct, whereas a leaf–leaf interaction is not [17]. The graph diameter is two, so distance alone appears benign. The scheduling consequence is more severe: all

native two-qubit operations reserve q_2, which serialises otherwise unrelated leaf interactions. The mapping consequence is also global. Whichever logical qubit occupies q_2 can interact directly with four logical identities, so exchanging the centre occupant changes the legality of many future gates.

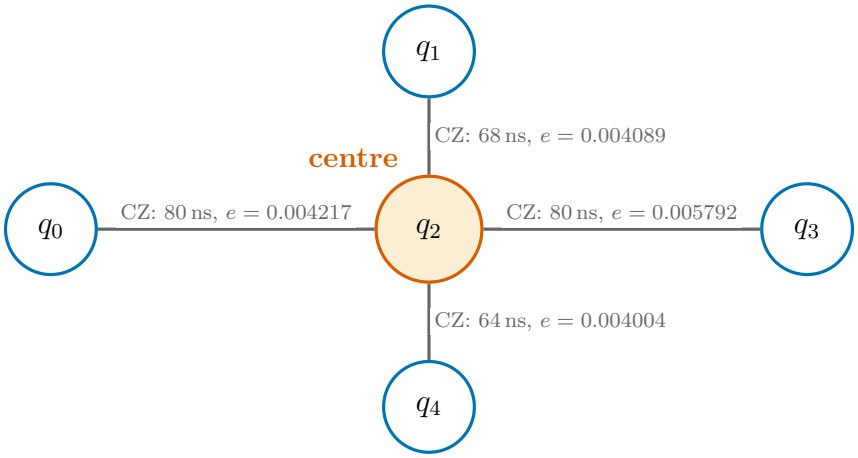


Figure 3.1: Five-qubit star coupling graph used by the router. Only centre–leaf controlled-Z operations are native. Durations and error probabilities correspond to the hardware configuration with signature 9da78e.

The complete per-qubit coherence and single-qubit calibration parameters of the frozen snapshot are reported in Appendix B; only the topology and hardware properties needed to motivate the routing model are retained here. The star topology also makes the restricted action set unusually compact. A blocked leaf–leaf CZ has only two adjacent SWAP directions: exchange the first operand with the centre, or exchange the second. The router adds a third mapping-preserving bridge action. There are no alternative shortest paths to enumerate, as there would be on a grid, but repeated centre competition can still yield a large search tree.

3.3 Derived action parameters

The frozen CZ parameters are shown directly in Figure 3.1, while the complete numerical configuration, including the derived SWAP values, is reported in Appendix B.

SWAP metrics are derived rather than independently calibrated quantities. For each centre–leaf edge, the implementation evaluates the two orientations of a three-CX SWAP decomposition, with each CX expressed through CZ and Hadamard primitives, and retains the candidate with the lower local J-style score.

For an ordered pair of distinct leaves (a, b), the bridge template contains two CZs on $c - b$, two on $a - c$, and four centre Hadamards. Under the independent primitive-success model,

$$P_{\text{bridge}}(a, b) = (1 - e_{CZ(c,b)})^2 (1 - e_{CZ(a,c)})^2 (1 - e_{H(c)})^4, \tag{3.2}$$

$$\tau_{\text{bridge}}(a, b) = 2\tau_{CZ(c,b)} + 2\tau_{CZ(a,c)} + 4\tau_{H(c)} \tag{3.3}$$

3.4 Effective coherence scale

The project computes the circuit-level inverse coherence scale as

$$\frac{1}{T_{\text{coh,eff}}} = \sum_{q \in Q} \left(\frac{1}{4T_{1,q}} + \frac{1}{2T_{2,q}} \right). \tag{3.4}$$

Although a retained configuration label refers to a T_2 coherence mode, the implemented scale uses both T_1 and T_2 through Equation (3.4). This quantity is an aggregate penalty scale used to normalise the makespan in the routing objective; it is neither the coherence time of an individual physical qubit nor a direct prediction of circuit fidelity. Its role in J is formalised in Chapter 5.

3.5 Consequences of heterogeneity

The route optimiser encounters three coupled forms of heterogeneity. Topological heterogeneity distinguishes the centre from the leaves. Edge heterogeneity makes some centre–leaf interactions cheaper and faster than others. Coherence heterogeneity is represented through a single aggregate duration scale. An initial placement may therefore be advantageous for different reasons. It may place a logical qubit with many interactions at the centre, assign frequently interacting logical pairs to lower-cost physical edges, or reduce resource contention and total execution time.

These criteria do not always lead to the same placement. Keeping one logical qubit at the centre can reduce the number of later SWAPs, but it can also increase contention on the central resource. In the frozen `9da78e` snapshot, for example, the SWAP between q_2 and q_4 has the lowest derived duration and error. After that SWAP, however, later interactions may be mapped onto the less favourable $q_2 - q_3$ CZ edge. A bridge preserves the mapping but uses four CZs and four centre Hadamards. No single local proxy therefore resolves every routing trade-off, motivating exact optimisation under the common hardware-aware objective. The complete numerical snapshot is reported in Appendix B.

3.6 What the hardware model omits

The model does not include readout error in the routing objective, time-dependent drift, simultaneous-gate crosstalk, leakage, residual coupling, drive-channel conflicts, pulse-shape constraints, reset, dynamical decoupling, or shot-to-shot correlations. Moreover, it assumes independent primitive success factors and operation-level resource occupancy [8, 9, 13]. These omissions define the boundary of the optimisation problem.

The experiments intentionally use the same frozen hardware snapshot throughout rather than live calibration data, so that differences between routing methods are not confounded by time-dependent hardware drift.

Chapter 4

The Qubit Mapping and Routing Problem

4.1 Input objects

Let $L = \{\ell_0, \ldots, \ell_4\}$ be the logical qubit identities of a normalised width-five circuit. The circuit is represented by a dependency DAG

$$\mathcal{D} = (V, E)$$

whose vertices are supported gate, barrier and measurement events. If $(u, v) \in E$, event u must precede v. The physical target is the star graph $\mathcal{H} = (Q, C)$ of Equation (3.1), together with versioned durations, error probabilities and coherence parameters. A logical-to-physical mapping is a bijection

$$\pi : L \to Q \tag{4.1}$$

When convenient, π is represented as a five-entry tuple whose i-th component gives the physical location of logical qubit ℓ_i; the inverse mapping identifies the logical occupant of each physical resource.

At a partial route, let $X \subseteq V$ be the completed DAG events. The ready set $\mathcal{R}(X)$ is given by Equation (2.1). A ready one-qubit event is legal at $\pi(\ell_i)$. A ready CZ on ℓ_i, ℓ_j is direct if

$$\{\pi(\ell_i), \pi(\ell_j)\} \in C$$

Since C is a star, this is equivalent to saying that one operand occupies q_2 and the other occupies a leaf.

4.2 Progress and scheduling state

An exact-search state must record enough information to determine the remaining logical work and the physical cost of every continuation. Logical progress is represented by a compact identifier p, whose interpretation depends on the selected non-routing scheduling semantics. The remaining state components describe the current logical-to-physical mapping, physical-resource availability and accumulated gate cost.

4.2.1 Forced-closure progress semantics

In the forced-closure model, p denotes logical progress after deterministic closure has been applied. Every ready non-routing operation prescribed by the closure semantics is materialised before the next routing choice, so the resulting fixed point determines both the routing events already consumed and the non-routing work already executed. Intermediate states corresponding to alternative timings or orderings of those forced operations are not part of the model.

4.2.2 JIT projected progress

In JIT-exact, logical progress is projected onto the routing-relevant two-qubit events, so that $p \equiv p_{2Q}$. Independent non-routing operations need not be materialised when they first become ready: they may remain pending until required by a selected two-qubit event or by terminal completion. Under the canonical JIT semantics, however, the non-routing materialisation state is not an additional independent state variable. Denoting by $\Phi(p)$ the materialised and pending non-routing obligations implied by projected progress, the model satisfies $p_u = p_v \implies \Phi(p_u) = \Phi(p_v)$. This JIT progress-determinacy property is proved formally in Chapter 11 and is what permits projected two-qubit progress to serve as a compact logical-state descriptor.

Physical timing is represented in both scheduling modes by availability ticks

$$\mathbf{t} = (t_0, t_1, t_2, t_3, t_4) \in \mathbb{N}^5$$

The implementation stores the centre availability $c = t_2$ and four signed centre-relative deltas

$$d_i = t_i - t_2 \qquad i \in \{0,1,3,4\}$$

This is a lossless coordinate change as long as each checked integer conversion succeeds. Absolute availability can be recovered as

$$t_i = c + d_i$$

The remaining dynamic physical quantity is the accumulated gate-error cost

$$G = -\sum_{g \in A_{\text{executed}}} \ln(1 - e_g)$$

where A_{executed} is the primitive physical instruction sequence materialised so far. At the semantic level, an exact-search label can therefore be represented by $u = (p, m, \mathbf{t}, G)$, where p is the mode-specific logical progress, m is the current mapping, $\mathbf{t}$ is the physical-resource availability vector and G is the accumulated negative-log gate cost. Reconstruction metadata and the compact centre-relative encoding are implementation concerns discussed later; they do not alter the future routing obligations represented by the label.

The same compact label structure is therefore used by both exact scheduling modes, although the semantics of p differ. In forced-closure mode, progress includes the deterministic closure already performed. In JIT-exact mode, p records projected two-qubit progress and the corresponding non-routing obligations are recovered through $\Phi(p)$. This property also allows the JIT frontier to retain the compact discrete key

$$K(u) = (p_u, m_u)$$

because labels with equal projected progress and equal mapping have the same remaining logical obligations under the canonical JIT scheduling semantics.

4.3 Legal actions

The exact model deliberately restricts where routing choices occur. For a routing-relevant CZ, the available physical actions depend only on whether its operands are adjacent on the star. For a ready CZ whose operands are already adjacent, there is a single successor: the calibrated CZ is executed on the occupied centre–leaf edge. If a ready CZ acts on two leaf qubits with physical operands a and b, and the centre is $c = q_2$, three successors are available:

$$S_a : \text{SWAP}(a, c) \text{ followed by the CZ}, \tag{4.2}$$
$$S_b : \text{SWAP}(b, c) \text{ followed by the CZ}, \tag{4.3}$$
$$B_{a,b} : \text{the fixed bridge decomposition}. \tag{4.4}$$

Under S_a, the logical qubits mapped to a and c are exchanged before the CZ is executed. Under S_b, the corresponding exchange occurs between b and c. The bridge decomposition executes the logical interaction without changing the current mapping. Each selected action consumes the logical CZ event. The use of SWAP and bridge-type transformations as connectivity-restoring mapping operations has prior precedent [23]; the particular three-action star model in Equation (4.4) is the deliberately restricted action model adopted in this thesis.

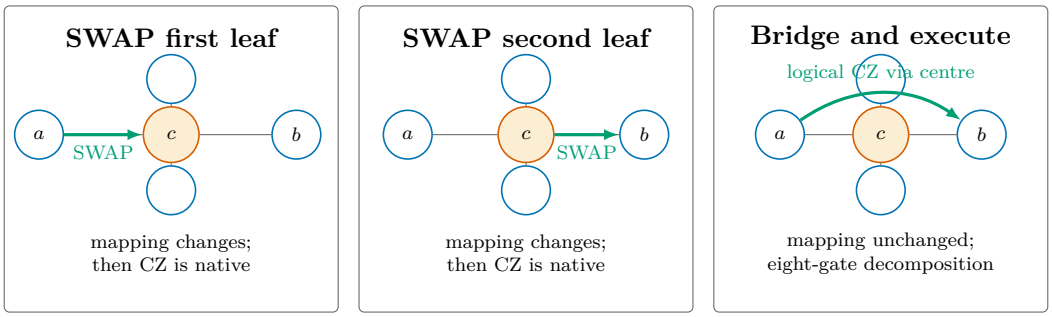


Figure 4.1: When a ready logical controlled-Z acts on two leaf qubits, the router has three available actions. Two of them swap one operand with the centre and then execute the controlled-Z. The third uses the bridge decomposition and does not change the current mapping. The implemented action model does not include proactive SWAPs.

For example, if $CZ(\ell_1, \ell_2)$ is blocked because both operands occupy leaves q_0 and q_1, the model can either swap ℓ_1 to the centre, swap ℓ_2 to the centre, or execute the mapping-preserving bridge. The first two choices leave different logical identities at the centre, while the bridge incurs a larger immediate primitive cost without changing the mapping. Which action is globally preferable therefore depends on the remaining circuit rather than on the local interaction alone.

There is no action that inserts a SWAP in anticipation of a future gate. Thus the search does not enumerate all physically possible adjacent-SWAP circuits. This restriction is central to both tractability and the scope of the certificate. It also means that comparison with a general-purpose router must be framed as a comparison of feasible outputs or objectives, not as identity of action spaces.

The exact transition semantics depend on the selected non-routing scheduling mode. In both cases, a routing action determines the mapping update, physical instruction sequence, timing evolution, accumulated gate-error increment and parent information required for reconstruction.

4.3.1 Forced-closure transition

In the forced-closure model, let $a \in \mathcal{A}(u)$ be a legal routing action at label u. Applying a first produces the corresponding routing successor $F(u, a)$, after which deterministic closure is applied until the next routing decision or terminal state is reached. The complete transition is therefore

$$v = C_{\mathrm{FC}}(F(u, a)) \tag{4.5}$$

The transition updates the mapping, logical progress, physical-resource availability and accumulated gate cost. Direct and bridge actions preserve the mapping,

whereas a SWAP action applies one centre–leaf transposition. The emitted primitive instructions, including those subsequently materialised by closure, update $\mathbf{t}$ through the ASAP rule of Equation (2.2) and contribute additively to G.

4.3.2 JIT-exact transition

In the JIT-exact model, the transition is additionally parameterised by an eligible two-qubit event g. Before any routing action is applied, the kernel deterministically materialises the still-unexecuted non-routing prerequisites required by g. Let

$$\widetilde{u} = M(u, g) \tag{4.6}$$

denote this prerequisite-materialisation step. The resulting successor is:

$$v = F_{\mathrm{JIT}}(M(u, g), g, a)\,, \qquad g \in R_{2Q}(p_u) \tag{4.7}$$

where $R_{2Q}(p_u)$ is the set of two-qubit events eligible at projected progress p_u.

The JIT transition updates the same physical quantities as its forced-closure counterpart, but materialises only the non-routing prerequisites required by the selected event g. Independent non-routing work remains pending, no global closure follows the routing action, and a deterministic terminal flush completes the residual non-routing events after all routing-relevant two-qubit events have been consumed.

4.4 Feasible and optimal routed circuits

A routed circuit is feasible if each normalised logical event is executed exactly once under the selected dependency and scheduling rules. Every physical two-qubit operation must respect the star coupling graph, and each SWAP must be accompanied by the corresponding update of the logical-to-physical mapping. The bridge decomposition may be used only for a blocked interaction between two leaf qubits. The initial and final mappings must both be valid bijections, and the reconstructed measurement operations must preserve the logical meaning of the original circuit.

The term *exact* refers to optimisation within a specified restricted route space. Since the two supported non-routing scheduling semantics do not define the same search model, their feasible route sets are distinguished explicitly. Let

$$\mathcal{R}_E \equiv \mathcal{R}_{\mathrm{FC}}$$

denote the finite set of feasible routed circuits generated under the forced-closure scheduling semantics, and let

$$\mathcal{R}_J \equiv \mathcal{R}_{\mathrm{JIT}}$$

denote the corresponding set generated under the JIT-exact scheduling semantics.

The forced-closure exact search therefore solves

$$J_E^\star = \min_{R \in \mathcal{R}_E} J(R) \tag{4.8}$$

with an optimal route satisfying

$$R_E^\star \in \arg\min_{R \in \mathcal{R}_E} J(R)$$

Likewise, the JIT-exact search solves

$$J_J^\star = \min_{R \in \mathcal{R}_J} J(R) \tag{4.9}$$

with

$$R_J^\star \in \arg\min_{R \in \mathcal{R}_J} J(R)$$

No general inclusion relation between $\mathcal{R}_E$ and $\mathcal{R}_J$ is assumed. The two sets arise from different canonical treatments of independent non-routing operations, and the JIT model does not enumerate arbitrary schedules of such operations.

For the later comparison with the permutation-based minimum-SWAP router of [1], let R_P denote its selected canonical complete route on the common supported domain. That method optimises inserted SWAP count rather than J, so its certificate is distinct from the J-exact certificates. A principal result proved in Chapter 11 establishes that R_P is representable within the JIT route space and therefore implies $J_J^\star \leq J(R_P)$ whenever JIT queue exhaustion certifies $J_J^\star$.

A completed `queue_exhausted` run certifies $J_E^\star$ or $J_J^\star$ for the corresponding restricted model, subject to the correctness assumptions proved later. A timeout may return a complete feasible incumbent but is not an optimality certificate.

4.5 Combinatorial structure

Each of the $5! = 120$ initial mappings defines a search root. A directly executable two-qubit event introduces no routing choice, whereas a blocked leaf–leaf CZ admits the three actions of Equation (4.4). Dependencies and state merging reduce the realised search, but the number of possible routing histories can still grow exponentially, consistently with the known computational hardness of qubit routing [7]. JIT-exact introduces an additional source of branching because several routing-relevant two-qubit events may be simultaneously eligible. The search may therefore explore different dependency-valid orders as well as different routing actions. Histories can later reconverge to the same logical progress and mapping while retaining different gate-cost and resource-availability coordinates; exact duplicate removal and the dominance relations developed in Chapter 9 determine when such alternatives can be safely merged or discarded.

Chapter 5

Hardware-Aware Objective and Scheduling Semantics

5.1 From primitive success to additive gate cost

For each emitted primitive gate g, the model supplies an error probability $e_g \in [0,1)$. Under the independent-success assumption used in this model, a physical instruction sequence $A = (g_1, \ldots, g_k)$ has gate-success probability

$$P_{\text{gates}}(A) = \prod_{r=1}^{k} (1 - e_{g_r}) \tag{5.1}$$

where e_{g_r} is the error associated with instruction g_r.

For incremental optimisation, it is more convenient to work with an additive quantity than with a product of probabilities close to one. Applying the negative logarithm gives

$$G(A) = -\ln P_{\text{gates}}(A) = \sum_{r=1}^{k} -\ln(1 - e_{g_r}) \tag{5.2}$$

Each term in this sum is non-negative. When a new physical action is appended to the sequence, its contribution is therefore added to G, and the gate cost cannot decrease.

Because $-\ln(\cdot)$ is strictly decreasing on $(0,1]$, minimising G is equivalent to maximising the primitive-success product in Equation (5.1). This equivalence holds within the frozen independent-error surrogate and does not interpret calibrated gate-error estimates as a complete statistical model of circuit success.

5.2 Coherence-normalised duration

The duration component is the final ASAP makespan divided by the effective coherence scale of Equation (3.4):

$$C_T(A) = \frac{T_{\text{exec}}(A)}{T_{\text{coh,eff}}} \tag{5.3}$$

Further details regarding the quantity $T_{\text{coh,eff}}$ are provided in Section B.1. Both numerator and denominator have units of time, so C_T is dimensionless and can be added to G. The implemented objective is

$$\boxed{J(A) = G(A) + C_T(A) = -\ln(P_{\text{gates}}(A)) + \frac{T_{\text{exec}}(A)}{T_{\text{coh,eff}}}} \tag{5.4}$$

and the exact router minimises this scalar. The objective J is identical in the forced-closure and JIT-exact modes. Only the set of routed and scheduled traces over which this objective is minimised changes.

The duration term is not the sum of gate duration: it depends on the maximum physical-resource availability. On disjoint resources, operations may overlap. However two centre–leaf CZs can't overlap, since they both reserve q_2. The star centre thus couples routing and scheduling even when the corresponding logical interactions involve different leaves.

Here T_{exec} denotes the quantum-circuit makespan and must not be confused with the classical runtime of the routing procedure; warm-start, exact-kernel and end-to-end timing boundaries are defined with the experimental methodology in Chapter 12.

5.3 Integer ticks and exact duration updates

Floating-point seconds are unsuitable as state-key coordinates: repeated addition can make mathematically equal schedules differ by representation. For this reason the router converts every relevant decimal duration into an exact rational scale; then selects an integer tick duration δ that represents all operation times. Availability is updated by using integer arithmetic. If an operation of n_g ticks uses R_g,

$$s_g^{(\text{tick})} = \max_{q \in R_g} t_q, \qquad t_q \leftarrow s_g^{(\text{tick})} + n_g \quad (q \in R_g) \tag{5.5}$$

The physical time is δt_q. In this way the schedule equality is exact and it is possible to avoid cumulative floating-point drift in resource availabilities. The gate component G, which contains logarithms, remains represented in binary64; the numerical safeguards required for sound pruning are discussed separately in Chapter 10.

5.4 Incremental objective state

For a partial label u, G_u is already final for every emitted primitive, whereas $\max_i t_{u,i}$ is only the current partial makespan. A simple current objective is

$$J_{\text{current}}(u) = G_u + \kappa \max_i t_{u,i} \qquad \kappa = \frac{\delta}{T_{\text{coh,eff}}} \tag{5.6}$$

Future actions have non-negative error cost and cannot decrease resource availability, so J_{current} is a lower bound on every continuation from u. The queue can combine this term with any configured admissible residual lower bound. At a terminal label, all DAG events have been consumed and

$$J_{\text{terminal}}(u) = G_u + \kappa \max_i t_{u,i} \tag{5.7}$$

At a terminal label this expression equals the complete route objective; reconstruction and authoritative replay provide an independent final consistency check.

The form $J(u) = G_u + \kappa \max_i t_{u,i}$ is also the structural basis for the continuation-safe Max-Plus dominance relation developed in Chapter 9.

5.5 Upper bounds, lower bounds and certificates

Any complete replay-valid route R provides an upper bound $J^\star \leq J(R)$ on the optimum of Equation (5.4), irrespective of whether R was produced heuristically. Its validated cost becomes the incumbent U. A partial label may be pruned only when a valid lower bound on every continuation is no smaller than U, subject to the deterministic tie policy; an incomplete or invalid candidate cannot define the incumbent.

5.6 Complete numerical example

Consider a centre Hadamard on q_2 followed by $CZ(q_2, q_0)$ under the frozen 9da78e snapshot. The two operations occupy the centre sequentially, giving $T_{\text{exec}} = 100\,\text{ns}$. With $e_H = 0.0004692$, $e_{CZ} = 0.004217$, and $T_{\text{coh,eff}} = 2.930\,\mu\text{s}$, the resulting objective is $J = 0.03883$. Table 5.1 reports the decomposition into gate and duration contributions.

Table 5.1: Numerical decomposition for a centre Hadamard followed by $CZ(q_2, q_0)$. The example uses frozen metrics and $T_{\text{coh,eff}} = 2.930\,\mu\text{s}$.

Quantity	Value	Calculation
Gate success P_{gates}	0.9953	$(1 - 0.0004692)(1 - 0.004217)$
Gate cost $-\ln P_{\text{gates}}$	0.004695	negative logarithm
ASAP makespan T_{exec}	100 ns	20 ns + 80 ns
Coherence term $T_{\text{exec}}/T_{\text{coh,eff}}$	0.03413	100 ns/2.930 µs
Objective J	0.03883	sum of the two costs

The example illustrates the relative scale of the two terms for this short sequence; for longer circuits, parallelism and mapping-dependent hardware parameters prevent a linear extrapolation.

5.7 Model limitations

The objective in Equation (5.4) is a fixed scalarisation. It does not expose a Pareto frontier between gate reliability and execution time, nor does it support a user-selected trade-off weight. The independent-error product also ignores correlated and coherent effects, the aggregate $T_{\text{coh,eff}}$ treats global makespan as a circuit-level exposure scale, and the ASAP scheduler omits pulse- and control-channel constraints. These assumptions do not affect exactness with respect to the stated model: they define the objective and route-cost semantics over which the certified minimum is taken. Alternative multi-objective formulations are left to future work in Chapter 14.

Chapter 6

Exact Routing Architecture and Scheduling Modes

6.1 End-to-end flow

The J-exact workflow separates feasible incumbent construction from exact proof. A normalised circuit and frozen hardware model are first used to construct a complete heuristic route; only after authoritative validation and recomputation of J can that route become an incumbent. The exact search then explores the restricted routing model independently of the heuristic construction, after which the selected terminal trace is reconstructed and validated. The central architectural boundary is therefore that the heuristic may provide a validated upper bound but cannot modify the exact action space, pruning rules or certificate semantics.

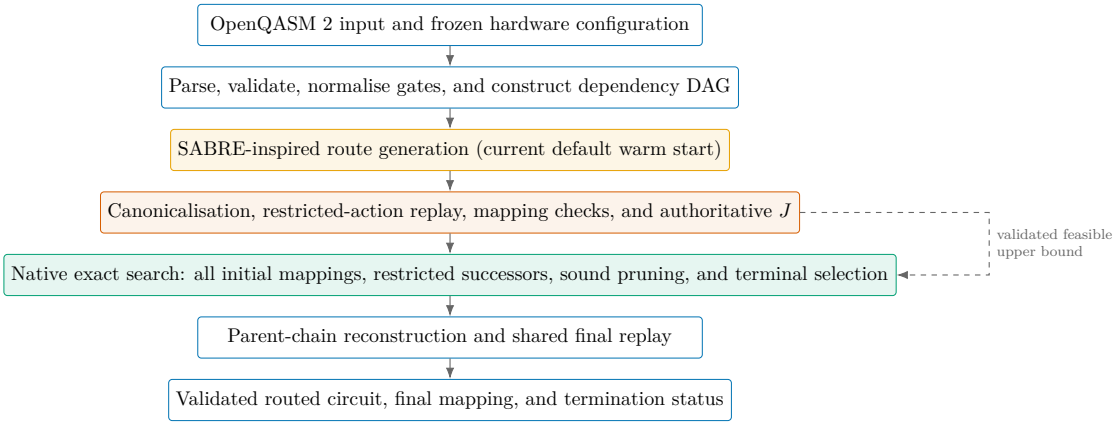


Figure 6.1: Current routing pipeline. The heuristic provides a validated incumbent, leaving the optimality guarantee and its certificate to the exact kernel.

6.2 Parsing and normalisation

Input OpenQASM 2 circuits are normalised into the supported one-qubit basis, controlled-Z interactions, barriers and terminal measurements. Unsupported operations are rejected rather than silently approximated. The normalised circuit is converted into the dependency representation used by the exact models, while physical durations are expressed in the integer-tick convention of Chapter 5. Detailed parsing, source-module and command-line information is reported in Appendix A.

6.3 Non-routing scheduling modes

The exact search supports the two non-routing scheduling semantics defined in Chapter 4: forced closure and JIT-exact. They share the same frozen hardware model, routing-action alphabet and objective J, but differ in when non-routing operations are materialised and therefore optimise over the distinct route-and-schedule spaces $\mathcal{R}_E$ and $\mathcal{R}_J$.

6.3.1 Forced-closure scheduling

In forced-closure mode, every initial state and routing successor is deterministically closed under ready non-routing work before the next routing decision. The closed state is the unit explored by the exact search, so alternative timings or orderings of those forced operations are not separate branches. Forced closure is therefore part of the definition of the restricted exact model rather than a heuristic acceleration.

6.3.2 JIT prerequisite materialisation

JIT-exact instead advances through eligible routing-relevant two-qubit events. When an event g is selected, only the still-unmaterialised non-routing prerequisites required by g are executed before its routing action; independent ready work remains pending. The pending obligations are determined by projected two-qubit progress, as formalised by the JIT progress-determinacy property introduced in Chapter 4 and proved in Chapter 11.

6.3.3 Terminal flush

After all routing-relevant two-qubit events have been consumed, JIT-exact applies a deterministic terminal flush to the remaining non-routing work. The flush updates the same objective and resource state as ordinary materialisation but introduces no routing choice, so it completes the routed trace without enlarging the search branching structure.

6.4 Authoritative candidate handoff

A heuristic candidate becomes an exact-search incumbent only after authoritative validation. The handoff canonicalises the route, replays the restricted routing actions, verifies mapping consistency and independently recomputes J from the frozen hardware model. Consequently, the heuristic's internal score never participates directly in exact pruning: only the objective of a complete validated route can establish the incumbent U.

6.5 Exact-search boundary

After incumbent validation, the exact-search component receives the normalised circuit model, frozen physical parameters, selected non-routing scheduling semantics and validated incumbent. It owns the complete exploration of the corresponding restricted route space, including successor generation, sound pruning and terminal selection. Forced closure and JIT-exact therefore differ semantically within the same overall exact-search framework rather than through different objective functions. Implementation backends, command-line selections and the precise benchmark configuration are documented in Appendix A and Chapter 12, while the search algorithm itself is developed in Chapter 8.

6.6 Reconstruction and final validation

The selected terminal state carries enough ancestry information to reconstruct the corresponding physical route. Reconstruction is followed by an independent validation step that checks logical execution, hardware legality, mapping consistency and the authoritative objective value. Internally generated traces may exploit their recorded construction provenance during replay, but this occurs only after exact search and does not alter successor generation, dominance, incumbent pruning or the meaning of the `queue_exhausted` certificate. Detailed reconstruction and validation paths are documented in Appendix A.

6.7 Termination and certificate semantics

Termination status determines what may be claimed about the returned route. A `queue_exhausted` result means that the selected exact route space has been completely exhausted under the sound reductions proved later and therefore certifies the corresponding optimum. A timeout may still return the best validated feasible incumbent found so far, but it carries no optimality certificate. Invalid inputs,

missing physical data or failed final validation are treated as fail-closed conditions rather than being reported as certified results.

6.8 Separation of concerns

The architecture deliberately separates three claims. Candidate construction establishes only that a route has been found; authoritative validation establishes feasibility and an objective value, thereby creating an upper bound; queue exhaustion under the exact model establishes that no lower-J route remains in that route space. This separation is also why changing the heuristic upper-bound strategy can affect the incumbent and runtime but cannot change the certified optimum, provided the exact action model, objective and sound pruning rules are unchanged.

Table 6.1: Production configuration of the final thesis release and of the two J-exact benchmark campaigns. Implementation backends are fixed by the publication CLI; the non-routing scheduling mode distinguishes forced closure from JIT-exact.

Option	**Final release selection**
Entry point	`python -m star_aware_router_ex INPUT.qasm`
Upper-bound strategy	`-upper-bound-strategy sabre-upper-bound`
Exact kernel	`-exact-kernel-backend native`
Native frontier	`-native-frontier-index linear`
Exact-map backend	`-native-exact-map-backend indirect-slot`
Forced-closure engine	`-forced-closure-engine legacy`
Worker count	`-exact-workers 1`
Bound mode	`-bound-mode cheap`
Scheduling mode	`eager` for forced closure; `jit-exact` for JIT-exact
Hardware	local frozen snapshot with signature 9da78e
Nominal configured limit	900 s per circuit
Expected exact certificate	`queue_exhausted` for a completed proof

Chapter 7

SABRE-Inspired Upper-Bound Construction

7.1 Original SABRE in context

SABRE is a scalable heuristic for qubit mapping based on a front layer of dependency-ready two-qubit gates, candidate routing moves and look-ahead scoring [2]. Its purpose is efficient feasible route construction rather than exact certification: committing greedily to locally scored actions does not prove that no better complete route exists. In this thesis, SABRE is therefore used only as the conceptual basis of a feasible-incumbent generator for a separate exact search.

The project-specific strategy is SABRE-inspired rather than a faithful reproduction of the original algorithm. It retains front-layer and look-ahead information, but specialises candidate generation to the five-qubit star, includes the mapping-preserving BRIDGE action, and incorporates calibrated error and duration in addition to topology. It does not use SABRE's original bidirectional placement refinement or decay mechanism [2].

7.2 Upper-bound construction

Starting from a selected initial mapping, the generator repeatedly materialises ready non-routing operations and directly executable centre–leaf CZs. If the circuit remains incomplete, each blocked ready CZ admits the same three restricted routing actions used by J-exact: either operand may be swapped to the centre and the CZ executed, or the fixed mapping-preserving BRIDGE action may be used. All candidates are scored deterministically, the minimum-score action is committed, and

the process continues until a complete route is obtained. Algorithm 1 summarises the construction.

Algorithm 1 Project-specific SABRE-inspired upper-bound generation

```
procedure SABRE_UPPER_BOUND(dag, hardware, config)
    D <- build_star_hardware_matrix(hardware, config.alpha)
    mapping <- best of all 120 initial mappings under D
    route <- empty physical circuit

    while dag is incomplete do
        execute ready non-routing and direct CZ events
        if dag is complete then break

        candidates <- legal actions for all blocked ready CZs
        for each candidate a do
            post_state <- virtually apply a
            score[a] <- immediate_cost(a)
                      + front_cost(a, post_state, D)
                      + lookahead_cost(a, post_state, D)
                      - consumption_reward(a, post_state)
                      + undo_penalty(a)
        end for

        a <- stable minimum-score candidate
        commit a; update mapping and logical progress
    end while

    reconstruct measurements under the final mapping
    return complete route and mappings
end procedure
```

7.3 Score function terms and formulation

7.3.1 Hardware-aware interaction matrix

The heuristic uses a star-specific interaction matrix D combining topology, derived SWAP error and derived SWAP duration. For distinct physical qubits q_i, q_j, interactions involving the centre require no routing move, whereas a leaf–leaf pair can be made adjacent by swapping either operand with the centre. Accordingly,

$$S_{ij} = \begin{cases} 1 & q_i, q_j \in Q_{\text{leaf}} \\ 0 & \text{otherwise} \end{cases} \qquad E_{ij} = \begin{cases} \min(e_i, e_j) & q_i, q_j \in Q_{\text{leaf}} \\ 0 & \text{otherwise} \end{cases} \qquad T_{ij} = \begin{cases} \min(t_i, t_j) & q_i, q_j \in Q_{\text{leaf}} \\ 0 & \text{otherwise} \end{cases}$$

with zero diagonal, where e_l and t_l are the derived error and duration of the centre–leaf SWAP through leaf l. After max-normalisation of the three matrices, the

interaction score is

$$D = \alpha_S \widetilde{S} + \alpha_E \widetilde{E} + \alpha_T \widetilde{T}.$$

The weights $\alpha_S, \alpha_E, \alpha_T$ control the relative contribution of topology, routing error and routing duration. Because the five-qubit star has known diameter two, these matrices are constructed directly rather than through a general all-pairs shortest-path computation.

7.3.2 Front-layer and look-ahead costs

For a candidate action a, let π_a denote the mapping after the action, with $\pi_a = \pi$ for BRIDGE. If R_{2Q} is the current ready two-qubit set, the front-layer term is

$$H_F(a) = \frac{1}{|R_{2Q}|} \sum_{g(x,y) \in R_{2Q}} D_{\pi_a(x),\pi_a(y)}$$

It favours candidates that make the current ready interactions inexpensive under the hardware-aware matrix.

A bounded virtual DAG traversal additionally constructs a future two-qubit set $L(a)$. If $d(g)$ denotes the look-ahead depth of event g, the corresponding cost is

$$H_L(a) = \begin{cases} \dfrac{\sum_{g(x,y) \in L(a)} \delta^{d(g)} D_{\pi_a(x),\pi_a(y)}}{\sum_{g \in L(a)} \delta^{d(g)}} & L(a) \neq \varnothing \\ 0 & L(a) = \varnothing \end{cases}$$

The discount $0 < \delta \leq 1$ controls the relative importance of more distant events.

7.3.3 Immediate action cost

For each candidate a, let $\Delta G(a)$ and $\Delta T(a)$ denote its immediate negative-log gate cost and physical duration. Normalising within the current candidate batch gives

$$C_A(a) = \alpha_E \frac{\Delta G(a)}{\max_b \Delta G(b)} + \alpha_T \frac{\Delta T(a)}{\max_b \Delta T(b)}. \tag{7.1}$$

SWAP-and-execute uses the derived SWAP and subsequent CZ metrics, whereas BRIDGE uses its corresponding fixed physical template.

7.3.4 Consumed-gates term N_{consumed}

The virtual traversal also records the two-qubit events that become executable before the next blocked frontier. Their number

$$N_{\text{consumed}}(a) = |X_a \cap V_{2Q}|$$

rewards candidates that sustain a useful centre assignment for several interactions before another routing move becomes necessary.

7.3.5 Anti-undo penalty P_{undo}

To discourage immediate routing oscillations, $P_{\text{undo}}(a) = 1$ when the candidate repeats the preceding centre–leaf SWAP and $P_{\text{undo}}(a) = 0$ otherwise. BRIDGE and non-reversing candidates incur no such penalty.

7.3.6 Total score

The final candidate score is

$$\text{Score}(a) = \lambda_A C_A(a) + \lambda_F H_F(a) + \lambda_L H_L(a) - \lambda_C N_{\text{consumed}}(a) + \lambda_P P_{\text{undo}}(a), \quad a^\star = \arg\min_{a \in \mathcal{C}} \text{Score}(a)$$

The five λ coefficients respectively weight immediate physical cost, the current front, look-ahead, centre-phase reward and immediate-undo penalty. This score is used only for heuristic construction and is not the exact objective J.

7.4 Initial placement

Because the target contains only five qubits, the heuristic evaluates all $5! = 120$ initial logical-to-physical mappings. With G_{2Q} denoting the circuit's two-qubit events, it selects

$$\pi_0^\star = \arg\min_{\pi \in S_5} \sum_{g(a,b) \in G_{2Q}} D_{\pi(a),\pi(b)}.$$

This choice affects only the heuristic route: the exact kernel independently considers all initial mappings and is therefore not restricted by the warm-start placement.

7.5 Production configuration

7.6 Retained configuration

The benchmarked configuration uses

$$(\alpha_S, \alpha_E, \alpha_T) = (0.10,0.45,0.45), \qquad (\lambda_A, \lambda_F, \lambda_L, \lambda_C, \lambda_P) = (7.0,0.8,2.0,1.0,1.3),$$

with look-ahead depth 40, at most 18 blocked look-ahead CZs, $\delta = 1$, exhaustive `best120` initial placement and BRIDGE enabled. Candidate selection is deterministic in the retained configuration because $\epsilon = 0$; the configured random seed therefore does not affect routing decisions.

7.7 Validation of the incumbent

The raw heuristic score is never used as an exact-search bound. A completed SABRE-inspired route is first passed through the authoritative handoff of Chapter 6, which verifies the restricted routing semantics, mapping and hardware legality and independently recomputes J. Only a complete route that passes this validation can become an incumbent; rejection of the candidate therefore fails closed before it can affect exact pruning.

7.8 Role in the exact search

Let R_{SABRE} denote the validated complete heuristic route. The only optimisation information transferred to the exact search is the feasible upper bound

$$U = J(R_{\mathrm{SABRE}})$$

The heuristic's internal score, initial-placement cost, front-layer terms and look-ahead estimates are not part of the exact state and cannot directly eliminate exact labels.

The same incumbent mechanism is used with both forced-closure and JIT-exact scheduling. Supplying U changes neither $\mathcal{R}_E$ nor $\mathcal{R}_J$, nor the legal routing actions, dominance relations or certificate semantics; it only permits sound pruning when a lower bound cannot beat the validated incumbent. Consequently, SABRE may affect route-construction time and exact-search effort, but it does not establish global optimality. Only queue exhaustion of the corresponding exact search certifies $J_E^\star$ or $J_J^\star$.

Chapter 8

Exact Dynamic-Programming Router

8.1 Common dynamic-programming machinery

The exact router is a best-first label-setting search over the finite state graph defined in Chapter 4. "Dynamic programming" refers to merging different histories that reach the same future-relevant state and retaining a frontier of incomparable labels. It is not a polynomial-time table indexed only by circuit position. Logical progress, mapping, five physical-resource availabilities and accumulated gate cost create a potentially large multi-dimensional frontier.

The search is initialised from all $5! = 120$ logical-to-physical mappings. Labels that share the same future-relevant discrete state are grouped into a common frontier, where exact duplicates and sound dominance relations can eliminate histories that cannot improve any common continuation. The labels that remain after pruning are expanded in best-first order. Information needed to reconstruct a complete route is stored separately from the state coordinates used for exact comparison.

A complete route that passes replay validation provides an incumbent objective value U. A candidate may be pruned against this incumbent only when a valid lower bound on its objective is no smaller than U. The heuristic used to obtain the incumbent does not otherwise affect successor generation, duplicate detection, or dominance tests. If the exact search reaches a complete terminal label with a lower objective value, U is replaced by that value. Queue exhaustion therefore certifies the minimum objective over the route and scheduling space defined by the selected

exact mode.

The complete exact-search loop is executed by the compiled kernel used in the experiments; the corresponding Python–native boundary and storage representation are documented separately in Appendix A.

This machinery is shared by both exact non-routing scheduling modes. What changes between them is the definition of logical progress and the rule used to determine which non-routing operations are materialised before or after a two-qubit routing decision. Those mode-specific transition semantics are described in the following sections.

8.2 Forced-closure exact mode

In forced-closure mode, the exact search explores only closure fixed points. All $5! = 120$ initial mappings are introduced independently of the heuristic incumbent; each root begins with zero accumulated gate cost and zero resource availability and is deterministically closed before entering the search. The incumbent therefore supplies only an objective upper bound and cannot restrict the set of initial mappings considered by the exact solver.

From a non-terminal closed label, the dependency state exposes the next routing-relevant two-qubit event. A direct centre–leaf interaction has one successor, whereas a blocked leaf–leaf interaction admits the two operand-to-centre SWAP-and-execute actions and the mapping-preserving BRIDGE action. After the selected action updates mapping, logical progress, G and resource availability, forced closure is applied again before the successor is admitted to the frontier.

$$\text{closed state} \longrightarrow \text{routing action} \longrightarrow \text{forced closure} \longrightarrow \text{next closed state}.$$

The deterministic treatment of non-routing work is therefore part of the forced-closure route-and-schedule model rather than an implementation shortcut. Queue exhaustion certifies the minimum J over the corresponding feasible set $\mathcal{R}_E$.

8.3 JIT-exact mode

The JIT-exact mode retains the common dynamic-programming machinery of Chapter 8, but changes the representation of logical progress and the treatment of non-routing operations. Instead of deterministically materialising every ready non-routing event, the search advances primarily through routing-relevant two-qubit events. Non-routing work is materialised only when it is required by the selected two-qubit continuation, with any remaining work completed by a deterministic terminal flush.

This distinction is important because readiness alone no longer implies immediate execution. A one-qubit operation may be logically ready and still remain pending if no selected two-qubit event currently requires it. The objective, physical scheduling rules and restricted routing actions are unchanged; what changes is the set of routed and scheduled traces represented by the exact search.

8.3.1 Projected progress and eligible events

Let D_{2Q} be the dependency structure obtained by projecting the normalised circuit onto its routing-relevant two-qubit events while preserving their induced precedence relations. A JIT label stores the projected progress p. The corresponding non-routing materialisation state, denoted by $\Phi(p)$, is determined uniquely by that progress and is therefore not treated as an independent search variable.

For a label u, let $R_{2Q}(u)$ denote the set of unconsumed two-qubit events whose projected predecessors have already been consumed. Any $g \in R_{2Q}(u)$ may be selected as the next two-qubit event. JIT-exact therefore explores the two-qubit orderings allowed by the dependency constraints, rather than fixing a single ordering before the search begins.

8.3.2 Prerequisite materialisation and routing successors

For a selected eligible event g at projected progress p, let $P(g, p)$ denote the still-unmaterialised non-routing prerequisites required before g can execute. Operations that are ready but independent of g remain pending. The deterministic materialisation operator M produces the intermediate state

$$u_g = M(u, g),$$

by executing exactly $P(g, p)$, updating G and the integer resource-availability vector without yet consuming g.

Routing successors are generated from u_g. If g is direct under the current mapping, it produces one successor; if its operands occupy two leaves, the model generates the two operand-to-centre SWAP-and-execute actions and the mapping-preserving BRIDGE action. Each successor consumes g, updates the mapping where necessary and advances the projected two-qubit progress.

8.3.3 Terminal completion

When all routing-relevant two-qubit events have been consumed, some independent or trailing non-routing operations may remain pending. JIT-exact therefore applies the deterministic terminal operator

$$u_{\text{term}} = F_{\text{term}}(u),$$

which materialises the residual supported operations using the same scheduling and gate-cost rules. Only the resulting fully materialised state is evaluated as a complete terminal. Because F_{term} introduces no routing choice, it completes the JIT trace without adding another search branch.

8.3.4 JIT transition pattern

The complete JIT transition can be summarised as

$$\begin{aligned}\text{projected state} &\longrightarrow \text{select eligible } 2Q \text{ event} \longrightarrow M(u, g)\\ &\longrightarrow \text{routing action} \longrightarrow \text{updated projected state},\end{aligned}$$

with F_{term} applied deterministically when the projected two-qubit progress is complete.

8.3.5 Worked JIT-exact example

To illustrate the JIT-exact semantics, consider the logical circuit shown in Figure 8.1. It involves five logical qubits

$$A,\ B,\ C,\ D,\ E$$

and two routing-relevant two-qubit events:

$$g_1 = CZ(A, B) \qquad g_2 = CZ(C, D)$$

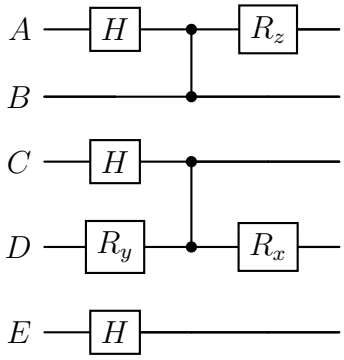


Figure 8.1: Logical circuit used to illustrate the JIT-exact semantics. The routing-relevant events are $g_1 = CZ(A, B)$ and $g_2 = CZ(C, D)$. The gates $H(A)$, $H(C)$, and $R_y(D)$ are prerequisites of those two-qubit events; $R_z(A)$ and $R_x(D)$ are trailing one-qubit gates; $H(E)$ is completely independent.

The projected two-qubit events are $g_1 = CZ(A, B)$ and $g_2 = CZ(C, D)$, with no precedence relation between them. Represent projected progress as $p = (x_1, x_2)$,

where $x_i = 1$ means that g_i has been consumed. The corresponding prerequisite-materialisation state is

p	$\Phi(p)$
(0,0)	$\varnothing$
(1,0)	$\{H(A)\}$
(0,1)	$\{H(C), R_y(D)\}$
(1,1)	$\{H(A), H(C), R_y(D)\}$

Thus $\Phi(p)$ is determined by projected progress rather than stored as an independent scheduling decision.

Assume this initial mapping: $A \mapsto q_0$, $B \mapsto q_1$, $C \mapsto q_2$, $D \mapsto q_3$, and $E \mapsto q_4$, with q_2 as the centre. Both g_1 and g_2 are eligible. However g_1 is blocked because A and B are mapped to two leaves, while g_2 can be executed directly. Suppose to select g_1 first. Its missing prerequisite is

$$P(g_1, (0,0)) = \{H(A)\}$$

so $M(u_0, g_1)$ materialises $H(A)$. The operations $H(C)$, $R_y(D)$, and $H(E)$ remain pending.

Consider the SWAP operation that swaps B with the centre occupant C and then executes g_1. Now the mapping satisfies $B \mapsto q_2$ and $C \mapsto q_1$, and the projected progress is $p = (1,0)$. The remaining event $g_2 = CZ(C, D)$ is now blocked because its operands are mapped to q_1 and q_3.

When g_2 is selected, its missing prerequisites are

$$P(g_2, (1,0)) = \{H(C), R_y(D)\}$$

Then, the operation $H(C)$ is therefore executed on q_1, and not on q_2, where C was initially mapped. Deferring a non-routing operation can thus change the calibrated gate cost and the position in the ASAP schedule, even though the logical operation itself is unchanged.

If the BRIDGE successor is chosen for g_2, the mapping is preserved and projected progress becomes $p = (1,1)$. The remaining operations $R_z(A)$, $R_x(D)$ and $H(E)$ have not been required by any routing-relevant event, so they are materialised only by the terminal operator F_{term}. The example can therefore be summarised as

$$u_0 \xrightarrow{M(\cdot,g_1)} \xrightarrow{\text{SWAP}+g_1} u_1 \xrightarrow{M(\cdot,g_2)} \xrightarrow{\text{BRIDGE}} u_2 \xrightarrow{F_{\text{term}}} u_{\text{term}}$$

The example exposes the defining JIT property: non-routing operations are neither forced immediately when ready nor independently branched over. Their materialisation is determined by the selected two-qubit progress, allowing different valid two-qubit orders to alter the physical placement and timing of deferred work while retaining a compact exact state.

8.4 Common frontier representation

Both exact scheduling modes use the same high-level frontier organisation. Labels are partitioned by the *frontier key*

$$K(u) = (p_u, m_u) \tag{8.1}$$

where p_u is the mode-specific logical-progress descriptor and m_u is the encoded logical-to-physical mapping.

In the forced-closure mode, p_u describes the logical progress reached after deterministic closure has been applied to a fixed point. In the JIT-exact mode, p_u instead represents the projected two-qubit progress introduced in Section 8.3. The same abstract key structure $K = (p, m)$ can therefore be retained even though the interpretation of the progress component differs between the two scheduling semantics.

Labels with different frontier keys are never compared by the dominance predicates. Different progress values may expose different future logical events, while different mappings may place the same logical operands on different physical qubits. In either case, the two labels need not admit the same physical continuation and cannot safely share a dominance frontier.

8.4.1 JIT progress determinacy

The compact JIT key relies on an additional determinacy property. Recall that $\Phi(p)$ denotes the non-routing materialisation state implied by projected two-qubit progress p. Since prerequisite materialisation is deterministic once the sequence of consumed two-qubit events is represented by the progress state,

$$p_u = p_v \quad \Longrightarrow \quad \Phi(p_u) = \Phi(p_v) \tag{8.2}$$

Thus, two JIT labels with equal projected progress have the same materialised-versus-pending non-routing obligations. In particular, the set of residual non-routing work that can be required by future two-qubit events or by the terminal flush is determined by p; it is not an additional independent search coordinate.

Consequently, equal projected progress and equal mapping imply identical remaining logical obligations. This property is what permits JIT labels to share the discrete frontier key $K = (p, m)$ and supplies the common-future-state premise required by the dominance arguments of Chapter 9. Its formal proof is given in Chapter 11.

8.4.2 Dynamic coordinates

Within a common frontier key, labels differ through the accumulated gate cost and physical-resource availabilities,

$$(G_u, t_{u,0}, t_{u,1}, t_{u,2}, t_{u,3}, t_{u,4}).$$

These coordinates are not part of $K(u)$: different histories with the same progress and mapping may therefore coexist in the same frontier when their error–time states are incomparable. They are precisely the coordinates used by classic and Max-Plus dominance.

8.4.3 Stable frontier storage

The native representation of frontier membership, stable label lifetime and reconstruction storage is documented in Appendix A; these choices do not alter the abstract frontier key or the dominance relations defined here.

8.5 Transactional candidate commit

A mode-specific transition first produces a complete candidate successor. Before that candidate enters the persistent search state, it is checked for structural validity, exact duplication, incumbent-based pruning, classic dominance, Max-Plus dominance and deadline validity. A rejected candidate produces no visible search state; a surviving candidate is committed as one complete label. This ordering ensures that every label exposed to the frontier represents a valid transition of the selected exact model. The native storage and atomic-publication details are documented in Appendix A.

8.6 Exact duplicates and dominance

At equal frontier key, an exact duplicate has identical dynamic state and only one deterministic representative need remain active. Classic dominance retains u over v when

$$G_u \leq G_v, \qquad t_{u,i} \leq t_{v,i} \quad \text{for all } i, \tag{8.3}$$

with at least one strict inequality. Monotone scheduling and common non-negative future gate costs then preserve the advantage under every common continuation.

Classic dominance is conservative because it cannot trade accumulated gate cost against earlier resource availability. The continuation-safe Max-Plus relation introduced in Chapter 9 extends this comparison while preserving the same common-continuation requirement.

8.7 Lower-bound modes and incumbent pruning

Let U be the objective of the best validated complete route and let $LB(u)$ be a lower bound on every completion of partial label u. The incumbent rule discards u whenever $LB(u) \geq U$. Exactness requires admissibility,

$$LB(u) \leq J(R) \qquad \forall R \in \mathcal{C}(u) \tag{8.4}$$

where $\mathcal{C}(u)$ is the set of complete continuations admitted by the selected exact model. Different bound modes may change search effort but, if admissible, cannot change the certified optimum.

8.7.1 Bound-mode off

The `off` mode disables optional residual cost-to-go estimates but retains the objective already accumulated by the partial schedule. With $M_u = \max_i t_{u,i}$,

$$LB_{\text{off}}(u) = G_u + \kappa M_u \tag{8.5}$$

Because future gate cost is non-negative and physical-resource availabilities cannot decrease, this is an admissible lower bound on every continuation.

8.7.2 Bound-mode cheap

The `cheap` mode adds two mapping-independent residual estimates. $R_G(u)$ is the sum of optimistic minimum hardware-error contributions for source operations not yet accounted for, while $R_C(u)$ is the corresponding optimistic minimum remaining centre occupation in scheduler ticks. The minima are computed independently over the legal physical realisations, deliberately relaxing the requirement that the same future routing action realise both minima.

$$\boxed{LB_{\text{cheap}}(u) = G_u + R_G(u) + \kappa\,[c_u + \max\{R_C(u), \Delta_u\}]} \tag{8.6}$$

The maximum avoids double-counting current leaf availability and unavoidable future centre work. Because the relaxation may assign every remaining source operation its globally most favourable hardware contribution irrespective of the future mapping, it is optimistic by construction and therefore suitable for admissible incumbent pruning.

8.7.3 Effect on exactness and runtime

For every partial label,

$$LB_{\text{off}}(u) \leq LB_{\text{cheap}}(u) \leq J(R), \qquad \forall R \in \mathcal{C}(u)$$

Hence `cheap` can prune labels retained by `off` without changing the exact optimisation problem. Its practical effect is only on search effort: a stronger bound can reduce active labels but also incurs evaluation overhead, and under a finite timeout different bound modes may therefore return different non-certified incumbents.

8.8 Terminal completion and deterministic selection

A terminal is evaluated only after the selected scheduling mode has produced a complete routed-and-scheduled trace: a forced-closure terminal must be closure-complete, whereas a JIT terminal is completed by F_{term}. Its objective is then evaluated using Equation (5.7); if it improves the incumbent, U is updated. Equal-J terminals are ordered deterministically by BRIDGE count, SWAP count and stable serial order only to choose a reproducible representative; these secondary fields do not modify the primary objective or its certified optimum.

8.9 Main search loops

Both scheduling modes use the same best-first label-setting loop. They differ only in root completion, successor generation and terminal completion: forced closure closes each root and routing successor, whereas JIT branches over eligible two-qubit events, materialises $P(g, p)$ through $M(u, g)$, and applies F_{term} after projected progress is complete.

Algorithm 2 Common exact label-setting search

```
procedure EXACT_SEARCH(mode, circuit, hardware, incumbent, deadline)
    initialise empty frontiers and best-first queue
    U <- objective(incumbent), if present; otherwise +infinity

    for each of the 120 mappings m do
        root <- MODE_INITIAL_ROOT(mode, m)
        COMMIT_IF_ADMISSIBLE(root, U)
    end for

    while queue is not empty do
        if deadline expired then return timeout with best feasible incumbent

        u <- POP_CURRENT_MINIMUM()
        if LB(u) cannot improve U then continue

        if MODE_IS_COMPLETE(mode, u) then
            terminal <- MODE_COMPLETE_TERMINAL(mode, u)
            UPDATE_INCUMBENT(terminal, U)
            continue
        end if

        for each mode-specific exact successor v of u do
            if deadline expired then return timeout without partial commit
            COMMIT_IF_ADMISSIBLE(v, U)
        end for
    end while

    return best terminal with certificate queue_exhausted
end procedure
```

For forced closure, a mode-specific successor consists of the selected routing action followed by deterministic closure. For JIT-exact, successor generation ranges over $g \in R_{2Q}(u)$, constructs $M(u, g)$, and then applies each legal physical action for g. All candidates subsequently use the same duplicate, dominance and incumbent-bound logic. The selected terminal is reconstructed through its stable parent relation and independently replayed; the native storage and reconstruction representation are documented in Appendix A.

8.10 Termination and timeout

The status `queue_exhausted` is returned only when no active exact-search label remains; subject to the correctness of the transition model and pruning rules proved in Chapter 11, the best terminal then certifies the mode-specific optimum. Deadline

expiration instead returns the best validated complete incumbent available at interruption and carries no optimality certificate. Other failures are reported separately rather than being conflated with either outcome.

8.11 Why the exact search remains difficult

The raw combinatorial sources of the search space were described in Section 4.5, while Equation (8.2) establishes that equal JIT projected progress implies equal residual non-routing obligations. These properties allow many different histories to be grouped under the same discrete frontier key, but they do not imply that only one label survives within each frontier.

The absence of a general polynomial input-size guarantee is consistent with the known computational hardness of qubit routing [7].

Histories that reconverge to the same frontier key $K(u) = (p_u, m_u)$ need not collapse to one label. Their dynamic coordinates $(G_u, t_{u,0}, \ldots, t_{u,4})$ may trade lower accumulated gate cost against earlier availability on different physical resources. When neither history dominates the other, exactness requires both to remain active.

This effect is particularly relevant in JIT-exact, where different dependency-valid orders of eligible two-qubit events can reconverge to the same projected progress and mapping while producing different ASAP availability profiles. JIT progress determinacy guarantees a common remaining logical obligation, but not a common physical cost-and-time state.

The exact-search difficulty depends not only on circuit size or two-qubit gate count, but also on branching, reconvergence and the width of the frontier of incomparable labels. The quality of the incumbent and the residual lower bounds affect how long those labels remain admissible. This motivates the stronger continuation-safe comparison developed in Chapter 9, which can eliminate some error–time trade-offs left incomparable by classic dominance.

Chapter 9

Max-Plus Objective Dominance

9.1 Motivation

Classic Pareto dominance, Equation (8.3), is valid but can be conservative. Consider two labels u and v. Label u may have a slightly larger accumulated gate cost than v, while all of its physical resources become available earlier. Under the classic criterion, the two labels are then incomparable: u is worse in G but better in resource availability.

For the objective J, however, the reduction in schedule time may offset the larger accumulated gate cost. In that case, u can remain no worse than v for every continuation that is feasible from both labels.

The central result of this chapter is a continuation-safe dominance relation that makes this error–time compensation explicit. For each physical resource it transforms the dynamic state into the coordinate $z_i = G + \kappa t_i$ and shows that, for labels representing the same future routing problem, componentwise dominance in these coordinates guarantees a no-worse value of J for every common complete continuation. The name reflects the max-plus structure of ASAP scheduling, in which operation start times are obtained by maxima and durations are added [38].

9.2 Comparable labels

Let u and v belong to the same frontier:

$$p_u = p_v, \qquad m_u = m_v \tag{9.1}$$

The interpretation of the progress component depends on the selected exact scheduling mode. In the forced-closure mode, equal progress identifies the same closed logical dependency state. In JIT-exact, p represents the projected two-qubit progress, so an additional continuation-state argument is required.

9.2.1 JIT continuation-state equivalence

The dominance relation is applied only to labels that represent the same future logical routing problem. In forced-closure mode, equal progress identifies the same closure-fixed dependency state. In JIT-exact, this premise follows from JIT progress determinacy: for committed pre-terminal-flush labels, $p_u = p_v$ implies $\Phi(p_u) = \Phi(p_v)$, and therefore the same residual two-qubit events, pending non-routing obligations, eligible-event set and prerequisite sets. The formal proof of this property is given in Chapter 11.

Together with $m_u = m_v$, equal progress therefore gives the same physical operands and the same legal direct, SWAP and BRIDGE choices for every future event. Hence the two labels admit the same legal continuation action sequences. They may nevertheless have different accumulated gate costs and resource availabilities, $G_u \neq G_v$ and $\mathbf{t}_u \neq \mathbf{t}_v$; Max-Plus dominance compares precisely these alternative dynamic histories. Labels with different frontier keys $K = (p, m)$ are never compared.

9.3 Transformed objective coordinates

Let δ be one schedule tick in seconds and define

$$\kappa = \frac{\delta}{T_{\text{coh,eff}}} \tag{9.2}$$

For each physical resource i, define

$$z_i(u) = G_u + \kappa t_{u,i} \tag{9.3}$$

Equivalently, $\mathbf{z}(u) = G_u\mathbf{1} + \kappa\mathbf{t}_u$. The implemented trade-off relation seeks

$$z_i(u) \leq z_i(v) \quad \text{for every } i \in \{0, \ldots, 4\} \tag{9.4}$$

Unlike classic dominance, Equation (9.4) permits $G_u > G_v$ if the availability advantage of u offsets it on every resource.

9.4 Continuation-safe Max-Plus dominance

Assumptions. The result applies to labels that admit the same legal continuations. Future routing choices and primitive costs are determined by the current

logical progress and mapping. Each physical primitive contributes a fixed non-negative gate-cost increment and a non-negative duration. Scheduling is performed by deterministic ASAP placement on the physical-qubit resources.

Under these assumptions, a corresponding continuation consists of the same future logical events and the same physical routing actions applied to both labels.

Main result (continuation-safe Max-Plus dominance). Let labels u and v satisfy Equation (9.1). If Equation (9.4) holds, then for every common complete continuation A,

$$J(u \circ A) \leq J(v \circ A) \tag{9.5}$$

If at least one inequality remains strict at the terminal maximum, the final objective is strictly lower.

Proof idea. Define the shifted availability vectors $\widehat{\mathbf{t}}_u = \mathbf{t}_u + (G_u/\kappa)\mathbf{1}$ and $\widehat{\mathbf{t}}_v = \mathbf{t}_v + (G_v/\kappa)\mathbf{1}$. Condition Equation (9.4) is equivalent to $\widehat{\mathbf{t}}_u \leq \widehat{\mathbf{t}}_v$ componentwise. For any fixed common continuation A, its ASAP scheduling map S_A is monotone and additively homogeneous, so this ordering is preserved through the continuation. Taking the terminal maximum and multiplying by κ converts the shifted scheduling inequality back into the hardware-aware objective; the continuation contributes the same additional gate cost to both labels. Hence $J(u \circ A) \leq J(v \circ A)$. The formal proof is given in Chapter 11.

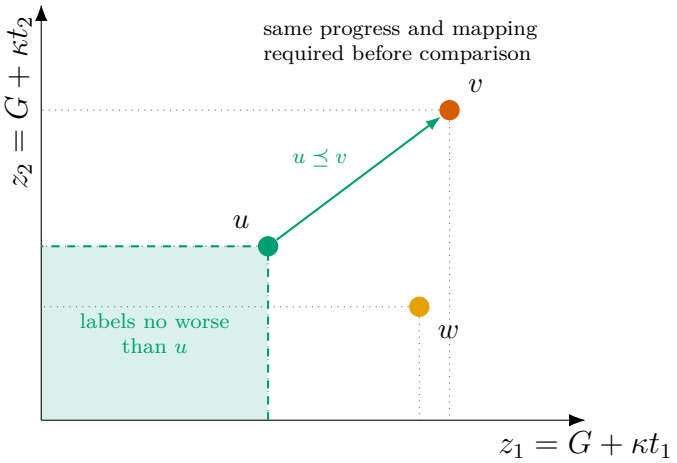


Figure 9.1: Two-dimensional projection of the Max-Plus dominance relation. Labels are compared only when they have the same logical progress and mapping. Within such a state, u dominates v if it is no worse in each transformed coordinate $z_i = G + \kappa t_i$. Label w is incomparable because it is better in one coordinate and worse in another. The exact dominance test uses all five physical-resource coordinates.

9.5 Numerical dominance example

With $\kappa = 0.01$, consider two equal-key labels u and v with $G_u = 1.10$, $\mathbf{t}_u = (10,10,10,10,10)$, $G_v = 1.00$, and $\mathbf{t}_v = (25,25,25,25,25)$. Classic dominance cannot compare them because u has larger accumulated gate cost but earlier resource availability. Their transformed states are

$$\mathbf{z}(u) = (1.20,1.20,1.20,1.20,1.20), \qquad \mathbf{z}(v) = (1.25,1.25,1.25,1.25,1.25)$$

so u Max-Plus-dominates v: its additional 0.10 gate cost is compensated by a 15-tick lead worth 0.15 on every resource. The condition is resource-wise; if even one transformed coordinate of u exceeded the corresponding coordinate of v, no dominance conclusion would follow.

9.6 Pruning relation and deterministic ties

Classic dominance and Max-Plus dominance are complementary safe relations evaluated within the same frontier key. In exact arithmetic, classic dominance, $G_u \leq G_v$ and $\mathbf{t}_u \leq \mathbf{t}_v$, implies weak Max-Plus dominance because $\kappa > 0$. The Max-Plus criterion therefore adds comparisons that are not resolved by classic dominance. In particular, a label with a larger accumulated gate cost may still dominate if its physical resources become available sufficiently earlier.

The implementation first evaluates the classic dominance relation. The Max-Plus test is applied only when the classic criterion does not determine the comparison.

Weak Max-Plus dominance is sufficient to preserve the optimal objective value. If several continuations attain the same value of J, however, more than one terminal representative may remain. To preserve the deterministic route-selection policy, a non-strict Max-Plus comparison additionally requires the retained label to be no worse under the established BRIDGE-count, SWAP-count and serial tie ordering. Exact duplicate states remain handled by the separate duplicate rule.

9.7 Binary64 interval guard

The mathematical comparison $G + \kappa t_i$ mixes a binary64 logarithmic sum with integer scheduling ticks and floating-point scale factors. A rounded comparison near equality could therefore assert a dominance relation that does not hold over the represented real quantities [39]. The implementation instead constructs conservative outward intervals for each transformed coordinate and asserts dominance only when the intervals are robustly separated.

$$\mathrm{nextafter}(U_u + 10^{-12}, +\infty) \leq L_v, \tag{9.6}$$

where $[L_u, U_u]$ and $[L_v, U_v]$ are outward enclosures of the two transformed coordinates. Failure to establish the guarded inequality leaves both labels active, affecting performance rather than correctness. The construction of these intervals and the corresponding finite-precision safeguards are detailed in Chapter 10.

9.8 Exactness scope and limitations

The dominance theorem relies on a common continuation set, state-independent future primitive metrics, deterministic ASAP scheduling and the scalar objective J of Equation (5.4). It does not automatically extend to models in which future legality or cost depends on absolute clock time, idle history, previous simultaneous operations or other history-dependent noise effects. The relation is also restricted to equal mappings: calibrated qubit and edge heterogeneity prevents a naive use of star-graph symmetries across different physical assignments. Finally, genuinely incomparable transformed states must remain in the frontier, so Max-Plus dominance reduces the explored state space without changing the exponential worst-case complexity.

9.9 Algorithmic significance

Max-Plus dominance changes which labels exist in the exact search. The Max-Plus criterion can prune a label that classic Pareto dominance must retain because one of its accumulated coordinates is worse. Its effect is therefore to reduce the number of labels explored by the exact search, rather than to reduce the processing cost of an otherwise unchanged label set.

This dominance test does not depend on how the incumbent is obtained. The upper-bound method only determines the current feasible objective value used for pruning. Max-Plus dominance instead compares alternative exact search histories that share the same continuation state. Its continuation guarantee is therefore part of the exact algorithm itself.

Chapter 10

Numerical Safety and Implementation Invariants

10.1 Why implementation details affect exactness

An exact search can be mathematically sound yet produce an invalid certificate if finite-precision comparisons assert a false pruning relation, compact encodings silently lose state information, an interrupted transition becomes partially visible, or reconstruction no longer represents the terminal state selected by the search [39]. This chapter therefore records the implementation invariants that are correctness-critical for J-exact. Data structures and performance-oriented engineering that preserve the same abstract search graph are documented separately in Appendix A.

10.2 Python and C++ boundary

Python performs input normalisation, heuristic-incumbent orchestration and authoritative post-search validation, while the compiled `ExactKernel` owns the complete exact-search loop. The boundary is crossed once per exact run rather than once per label or successor, so the native implementation evaluates the same progress, routing, scheduling, dominance and incumbent-pruning semantics defined in the preceding chapters without introducing an additional Python-level search state. The detailed payload, source modules and native data structures are documented in Appendix A.

10.3 Lossless state representation

Compact storage is admissible only when it represents the mathematical search state without loss. Every narrowing conversion is therefore range-checked before the corresponding label can become visible to the frontier; values are never silently saturated, wrapped or truncated. In particular, logical progress, mapping identifiers, resource-availability differences and parent identifiers must lie within the ranges of their retained native representations. If the compact representation is not valid for the complete problem instance, the corresponding fast layout is not eligible. Thus storage width can affect performance and memory consumption but cannot identify two mathematically distinct exact states.

10.4 Finite-precision pruning safety

A transformed comparison is asserted only when all required quantities are finite, their scales are valid and the corresponding outward intervals are sufficiently separated. Otherwise the comparison fails closed: no dominance relation is asserted and both labels remain available to the exact search. Report formatting is likewise kept separate from search identity; decimal rounding used for presentation is never used to merge states or establish pruning.

10.5 Atomic transitions and interruption safety

A deadline or runtime failure must not expose a partial exact transition. Successor construction and all correctness-critical checks therefore precede the visible commit of the candidate state; if interruption or validation failure occurs before commit, no partial successor is admitted to the frontier. Deadline expiry is reported as a non-certified interruption, whereas allocation, conversion or reconstruction failures follow an error path and cannot be confused with either timeout or `queue_exhausted`. This preserves the transactional search-state invariant introduced in Section 8.5.

The validated exact configuration uses one worker, so mapping enumeration, action ordering, frontier scans and deterministic terminal ties are serial under the retained execution model. Multi-worker exact search is outside the validated scope considered in this thesis; no claim is made that it would preserve the same transition linearisation or representative trace.

10.6 Validation strategy

Correctness-critical implementation paths are tested at two complementary levels. Differential and focused tests compare compiled transitions, scheduling updates

and dominance decisions against reference or high-precision cases, including finite-precision Max-Plus comparisons and deadline interruption. End-to-end validation instead reconstructs complete routed traces and independently checks logical semantics, hardware legality, mapping evolution and the authoritative objective J.

JIT-exact additionally validates projected progress, prerequisite materialisation and deterministic terminal completion. These tests provide implementation evidence for the invariants used by the formal arguments, but they are not themselves the proof of exactness; the certificate claims and their mathematical justification are developed in Chapter 11.

10.7 Post-search constructive validation

Final route validation depends on the provenance of the route being checked. Routes whose internal construction is not represented by the native exact state trajectory are validated through the general semantic replay path. By contrast, a route produced directly by the native JIT-exact dynamic program carries a stronger internal witness: the selected terminal label is connected to an initial mapping by a stable parent chain, and every transition retains the information required to reconstruct the corresponding exact JIT step.

The current implementation exploits this distinction through a provenance-specific constructive replay for native JIT traces. The validated mode is reported as

`jit-internal-constructive-v1`

For such a trace, replay follows the parent chain in forward order rather than searching for a semantic explanation of the routed circuit. Each recorded transition is reconstructed according to the same JIT semantics used by the exact kernel.

Routes reconstructed from an internal JIT-exact parent chain carry explicit provenance for the selected two-qubit events and routing actions. The final validation replies the recorded construction directly. It reconstructs for each transition the deterministic prerequisite materialisation, checks if the physical action is legal, and verifies the updates to the mapping and logical progress. It finally applies the deterministic terminal completion.

The objective value used for validation is not taken from the stored terminal label but it is recomputed from the reconstructed physical trace. This replay is performed only after the search has finished. It does not participate in successor generation, frontier construction, duplicate detection, classic or Max-Plus dominance, incumbent pruning or terminal selection. Replacing it with another correct final validator would therefore leave the exact search state graph and the `queue_exhausted` certificate unchanged. Detailed replay provenance and implementation paths are documented in Appendix A.

Chapter 11

Correctness, Certificates and Complexity

11.1 Statement of the exactness claims

The two exact scheduling modes optimise the same hardware-aware objective J over different routed-and-scheduled trace spaces. Let $\mathcal{R}_E$ denote the feasible set induced by forced closure and $\mathcal{R}_J$ the feasible set induced by JIT-exact scheduling. Their optimality certificates must therefore be stated separately.

Forced closure. If a forced-closure run terminates with `queue_exhausted`, its reconstructed and authoritatively validated terminal route $R_E^\star$ satisfies

$$R_E^\star \in \arg\min_{R \in \mathcal{R}_E} J(R) \tag{11.1}$$

and hence $J(R_E^\star) = \min_{R \in \mathcal{R}_E} J(R)$.

JIT-exact. If a JIT-exact run terminates with `queue_exhausted`, the reconstructed terminal route is validated independently from the stored search state. Let this route be $R_J^\star$. Then

$$R_J^\star \in \arg\min_{R \in \mathcal{R}_J} J(R), \tag{11.2}$$

so that

$$J(R_J^\star) = \min_{R \in \mathcal{R}_J} J(R).$$

This optimality statement applies only to the JIT-exact route space $\mathcal{R}_J$. It does not imply optimality over dependency-valid schedules or routing actions that are

excluded from the JIT-exact model. No general inclusion relation between $\mathcal{R}_E$ and $\mathcal{R}_J$ is assumed.

11.2 Mode-specific completeness and scheduling invariants

The remaining completeness obligations are technically necessary but are not the principal theoretical results of the thesis. The exact search considers all $5! = 120$ initial mappings; forced-closure successor generation enumerates every action admitted by $\mathcal{R}_E$; and deterministic closure contracts only transition chains that contain no routing choice within that model. The corresponding proofs are collected in Section C.1.

11.2.1 JIT progress determinacy

Let V_{2Q} be the routing-relevant two-qubit events and V_{NR} the remaining supported non-routing events. For projected two-qubit progress p, let $C_{2Q}(p) \subseteq V_{2Q}$ denote the consumed two-qubit events, and let $\mathrm{Anc}_{\mathrm{NR}}(g)$ be the non-routing predecessors that must be materialised before event g. Define

$$N_{\mathrm{done}}(p) = \bigcup_{g \in C_{2Q}(p)} \mathrm{Anc}_{\mathrm{NR}}(g), \tag{11.3}$$

$$N_{\mathrm{pending}}(p) = V_{\mathrm{NR}} \setminus N_{\mathrm{done}}(p), \tag{11.4}$$

and $\Phi(p) = (N_{\mathrm{done}}(p), N_{\mathrm{pending}}(p))$.

Lemma 1 (JIT progress determinacy).
For every committed pre-terminal-flush JIT label u, the set $N^{\mathrm{impl}}_{\mathrm{done}}(u)$ of non-routing events actually materialised along its search history satisfies

$$N^{\mathrm{impl}}_{\mathrm{done}}(u) = N_{\mathrm{done}}(p_u) \tag{11.5}$$

Consequently, $p_u = p_v$ implies $\Phi(p_u) = \Phi(p_v)$: pending non-routing obligations are uniquely determined by projected two-qubit progress and are not an additional independent search coordinate.

Proof
Proceed by induction on $k = |C_{2Q}(p)|$. At a JIT root, $C_{2Q}(p_0) = \varnothing$. No non-routing event is materialised merely because it is ready, and therefore

$$N^{\mathrm{impl}}_{\mathrm{done}}(u_0) = N_{\mathrm{done}}(p_0) = \varnothing$$

Assume that

$$N_{\text{done}}^{\text{impl}}(u) = N_{\text{done}}(p)$$

and let $g \in R_{2Q}(p)$ be the next selected event. Let u' denote the committed successor and p' its projected progress. Before executing g, JIT materialises exactly

$$P(g, p) = \text{Anc}_{\text{NR}}(g) \setminus N_{\text{done}}(p)$$

No other ready non-routing event is consumed at this step. Therefore

$$N_{\text{done}}^{\text{impl}}(u') = N_{\text{done}}(p) \cup P(g, p) = N_{\text{done}}(p) \cup \text{Anc}_{\text{NR}}(g) = N_{\text{done}}(p')$$

because $C_{2Q}(p') = C_{2Q}(p) \cup \{g\}$. Hence the claim holds after every committed JIT transition, independently of the history or dependency-valid two-qubit order that reached the same projected progress. Therefore equal projected progress implies identical materialised and pending non-routing sets. □

Remark
The lemma concerns ordinary JIT frontier labels before terminal completion. After all two-qubit events have been consumed, the deterministic terminal operator materialises the residual $N_{\text{pending}}(p)$; this post-flush completion is a solution, not another frontier state with the same projected-progress key.

11.2.2 Further JIT Completeness Obligations

The remaining JIT completeness obligations are prerequisite soundness, complete enumeration of every eligible two-qubit event and legal physical action, and sound deterministic terminal completion. Their statements and proofs are given in Appendix C; they establish that the compact projected state does not omit a transition admitted by $\mathcal{R}_J$. app:sec:jit-successor

11.3 Shared reduction and implementation invariants

Within a fixed scheduling mode $M \in \{E, J\}$, two labels are compared only when they share the same discrete frontier key $K = (p, m)$. Forced-closure labels with equal progress and mapping have the same remaining deterministic routing problem; for JIT-exact, Lemma 1 additionally guarantees identical pending non-routing obligations. Hence

$$K(u) = K(v) \quad \Longrightarrow \quad \mathcal{C}_M(u) = \mathcal{C}_M(v) \tag{11.6}$$

where $\mathcal{C}_M(u)$ denotes the legal continuation action sequences admitted from u.

Exact duplicate removal, admissible incumbent pruning and classic dominance are immediate reductions on this common continuation set: duplicates have identical dynamic coordinates, an admissible lower bound can discard only labels whose continuations cannot improve the incumbent, and classic dominance preserves a no-worse accumulated gate cost and no-later resource availability. Detailed proofs are collected in Section C.2.1.

11.3.1 Classic and Max-Plus dominance

Common-continuation premise
Both dominance relations compare labels only within a fixed scheduling mode $M \in \{E, J\}$ and within the same frontier key, $K(u) = K(v)$ that is,

$$p_u = p_v, \qquad m_u = m_v$$

Let $\mathcal{C}_M(u)$ denote the set of legal continuation action sequences admitted from label u by scheduling mode M.

- For the forced-closure mode, both labels are closure fixed points. Equal logical progress therefore exposes the same remaining forced-closure dependency state and the same deterministic next routing event. Equal mapping places every future selected interaction on the same physical operands. Applying corresponding actions produces the same progress and mapping updates, after which deterministic forced closure again restores corresponding closure fixed points. Hence
$$\mathcal{C}_E(u) = \mathcal{C}_E(v)$$

- For JIT-exact, equal projected progress implies, by Lemma 1,
$$\Phi(p_u) = \Phi(p_v)$$
Thus the labels have the same consumed and residual two-qubit events, the same pending non-routing obligations, and the same eligible-event set:
$$R_{2Q}(p_u) = R_{2Q}(p_v)$$
For every eligible event g,
$$P(g, p_u) = P(g, p_v)$$
Equal mapping additionally gives the same physical operands and therefore the same legal direct, SWAP or BRIDGE action set for every such event. By prerequisite-materialisation soundness and JIT successor completeness, the same selected event and physical action produce corresponding successor progress and mappings in both labels. Repeating the argument inductively gives
$$\mathcal{C}_J(u) = \mathcal{C}_J(v)$$

Thus, within either fixed scheduling mode, the common-continuation relation of Equation (11.6) applies. The labels may still differ in accumulated gate cost and physical-resource availability. These dynamic differences are precisely what the following dominance relations compare.

The classic dominance lemma is detailed in Section C.2.4. This section presents the primary result: the Max-Plus Dominance.

Continuation-safe Max-Plus dominance
Recall the transformed coordinates $z_i(u) = G_u + \kappa t_{u,i}$ introduced in Chapter 9. For labels with a common frontier key, assume

$$z_i(u) \le z_i(v) \qquad \text{for every physical resource } i.$$

Then every common complete continuation from u has objective no larger than the corresponding continuation from v.

Proof
Let $\mathbf{t}_u$ and $\mathbf{t}_v$ be the two resource-availability vectors and define

$$\widehat{\mathbf{t}}_u = \mathbf{t}_u + \frac{G_u}{\kappa}\mathbf{1}, \qquad \widehat{\mathbf{t}}_v = \mathbf{t}_v + \frac{G_v}{\kappa}\mathbf{1}.$$

The transformed inequalities $G_u + \kappa t_{u,i} \le G_v + \kappa t_{v,i}$ are equivalent to $\widehat{\mathbf{t}}_u \le \widehat{\mathbf{t}}_v$ componentwise. Consider any common complete continuation $A \in \mathcal{C}_M(u) = \mathcal{C}_M(v)$, and let $S_A(\mathbf{t})$ denote the final resource-availability vector produced by deterministic ASAP scheduling of that fixed continuation from initial availability $\mathbf{t}$. ASAP scheduling is monotone, so $\mathbf{x} \le \mathbf{y}$ implies $S_A(\mathbf{x}) \le S_A(\mathbf{y})$, and it is additively homogeneous, $S_A(\mathbf{t} + c\mathbf{1}) = S_A(\mathbf{t}) + c\mathbf{1}$. Therefore

$$S_A(\mathbf{t}_u) + \frac{G_u}{\kappa}\mathbf{1} \le S_A(\mathbf{t}_v) + \frac{G_v}{\kappa}\mathbf{1}.$$

Taking the maximum component and multiplying by $\kappa > 0$ gives

$$G_u + \kappa \max_i S_A(\mathbf{t}_u)_i \le G_v + \kappa \max_i S_A(\mathbf{t}_v)_i.$$

Because the two labels share progress and mapping and execute the same continuation A, the continuation contributes the same additional primitive gate cost ΔG_A to both sides. Adding ΔG_A yields

$$J(u \circ A) \le J(v \circ A).$$

Thus removing v preserves a no-worse representative for every common complete continuation. □

11.3.2 Reconstruction correctness

Finally, the selected terminal is reconstructed and independently validated under the mode-specific replay semantics; this post-search validation confirms that the emitted physical trace realises the terminal state selected by the exact search. The reconstruction proof is given in Section C.2.5.

11.4 Queue-exhaustion certificates

Queue exhaustion has a mode-relative meaning. The two modes use the same dynamic-programming infrastructure, however they search different sets of routed and scheduled traces. Therefore, their optimality claims must be proved separately.

Let's assume for both modes that all hardware metrics are finite, the incumbent has been validated and provides a valid upper bound for the selected mode, and the reconstruction and the final validation succeed. The result supposes that the run does not terminate because of a deadline or a runtime failure.

11.4.1 Forced-closure theorem

Theorem 1 (**forced-closure queue exhaustion**).
Suppose the native kernel is executed in the forced-closure scheduling mode and returns `queue_exhausted`. Let $R_E^\star$ denote its selected terminal route. Then

$$J(R_E^\star) = \min_{R \in \mathcal{R}_E} J(R) \tag{11.7}$$

where $\mathcal{R}_E$ is the routed-and-scheduled trace space admitted by the forced-closure exact model. Equivalently, the returned solution satisfies the forced-closure exactness claim of Equation (11.1).

The proof combines initial-state completeness, complete forced-closure successor generation, sound duplicate/dominance/incumbent reductions and reconstruction validity. The full argument is given in Section C.3.1.

11.4.2 JIT theorem

Theorem 2 (**JIT queue exhaustion**).
Suppose the native kernel is executed in JIT-exact mode and returns `queue_exhausted`. Let $R_J^\star$ denote its selected complete terminal route. Then

$$J(R_J^\star) = \min_{R \in \mathcal{R}_J} J(R) \tag{11.8}$$

where $\mathcal{R}_J$ is the routed-and-scheduled trace space admitted by the JIT-exact model.

The proof additionally uses JIT progress determinacy, prerequisite soundness, complete event/action enumeration and deterministic terminal completion. Together these properties ensure that every route in $\mathcal{R}_J$ is represented or safely subsumed before queue exhaustion. The full proof is given in Section C.3.2.

11.5 Relationships between the routing models

11.5.1 Forced closure versus JIT-exact

The two exact scheduling models are not universally ordered by J. Certified finite counterexamples exist in both directions:

$$\exists\, C_A : \quad J_E^\star(C_A) < J_J^\star(C_A), \qquad \exists\, C_B : \quad J_J^\star(C_B) < J_E^\star(C_B)$$

Hence neither scheduling semantics universally dominates the other, and no universal nesting relation between the families $\mathcal{R}_E(C)$ and $\mathcal{R}_J(C)$ follows. The explicit counterexample circuits and their certified objective decompositions are reported in Section C.4.

The mechanisms are different: forced closure can benefit from early materialisation that exposes ASAP parallelism, whereas JIT can benefit from deferring independent work across mapping changes and later executing it on a different physical resource. The strongly one-sided empirical comparison observed in the benchmark must therefore not be interpreted as a universal theorem.

11.5.2 JIT-exact versus the permutation-based router

The JIT-exact search and the permutation-based IQM Spark exact router solve different optimisation problems and use different search representations. The former minimises the hardware-aware objective J over its JIT search space, whereas the latter searches two-qubit topological orders and constructs a routing solution that is exact with respect to inserted SWAP count [1].

Nevertheless, on their common supported domain, the canonical complete route produced by the faithful reconstruction of the permutation-based algorithm is representable by the JIT-exact search. This section formalises that relationship.

Common supported domain.

Let $\mathcal{D}_\cap$ denote the circuits for which the two implementations use the same five-qubit star target, supported gate/dependency semantics and routing-relevant two-qubit DAG; identify the same supported non-routing prerequisites; reconstruct those prerequisites and terminal residual work according to the same canonical dependency-valid protocol; and evaluate the resulting physical operations with the

same frozen hardware model, integer-tick ASAP scheduler and objective J. Every direct or operand-to-centre SWAP action used by the permutation-based reconstruction belongs to the JIT physical-action alphabet; BRIDGE is an additional JIT option and is not required for embedding.

Paper-order representation

For a dependency-valid two-qubit order $\omega = (g_1, \ldots, g_n)$, let $R_P(\omega)$ denote the complete canonical route reconstructed by the permutation-based router from its selected logical-centre sequence. The final permutation-based route is denoted R_P.

Permutation-route embedding theorem

Theorem 3 (**permutation-route embedding and objective consequence**)**.** For every circuit in $\mathcal{D}_\cap$, the canonical complete route R_P returned by the faithful permutation-based reconstruction is representable in the JIT-exact route space:

$$R_P \in \mathcal{R}_J \tag{11.9}$$

Consequently, if JIT-exact terminates with `queue_exhausted`,

$$J_J^\star \leq J(R_P) \tag{11.10}$$

Proof
Let $\omega = (g_1, \ldots, g_n)$ be the dependency-valid two-qubit order underlying R_P. We construct a JIT path by induction over the prefix of paper events already reproduced. The initial mapping used by R_P is one of the $5! = 120$ mappings represented by the JIT root set. Assume that after $g_1, \ldots, g_k$ the JIT path has reproduced the same canonical physical prefix, and hence has the same consumed two-qubit set, mapping and materialised supported non-routing events as the paper reconstruction.

Because ω is topological, every two-qubit predecessor of g_{k+1} has already been consumed, so $g_{k+1} \in R_{2Q}(p_k)$ and JIT includes a branch selecting the same next event. Before routing g_{k+1}, JIT materialises exactly

$$P(g_{k+1}, p_k) = \mathrm{Anc}_{\mathrm{NR}}(g_{k+1}) \setminus N_{\mathrm{done}}(p_k)$$

By JIT progress determinacy and the common-domain materialisation assumption, this set is the same non-routing segment produced by the canonical paper reconstruction before g_{k+1}. The operations are emitted in the same dependency-valid order.

The routing action used for g_{k+1} in the paper reconstruction is either a direct centre–leaf execution or an operand-to-centre SWAP followed by the CZ. Both actions

are included in the JIT action set. Choosing the corresponding JIT successor therefore reproduces the same physical operations and mapping update; because both constructions use the same hardware metrics and ASAP transition rules, their dynamic coordinates remain identical. The induction invariant is thus preserved.

After g_n, the projected two-qubit progress is complete. By JIT progress determinacy, the remaining non-routing events are the same as those in the faithful paper reconstruction. JIT terminal-flush soundness then materialises these events without adding any further routing choice. The complete canonical paper trace is therefore contained in the JIT route space:

$$R_P \in \mathcal{R}_J.$$

If JIT terminates with `queue_exhausted`, then

$$J_J^\star = \min_{R \in \mathcal{R}_J} J(R)$$

Since $R_P \in \mathcal{R}_J$,

$$J_J^\star = \min_{R \in \mathcal{R}_J} J(R) \leq J(R_P)$$

□

Strictness and limits

The inequality need not be strict. Equality holds when the canonical permutation route already attains the minimum over $\mathcal{R}_J$. A strict improvement occurs only if there exists another route $R' \in \mathcal{R}_J$ such that $J(R') < J(R_P)$. The inequality also requires JIT queue exhaustion; a timeout incumbent is feasible but need not satisfy $J_{\text{inc}} \leq J(R_P)$. Finally, the theorem compares the JIT minimum-J certificate with the post-hoc J of a route that is exact for inserted SWAP count, not for the same primary optimisation problem.

11.5.3 Forced closure versus the permutation-based IQM Spark router

No analogous one-sided theorem holds between the forced-closure optimum and the canonical permutation-based route. Let

$$J_P(C) = J(R_P(C))$$

denote the authoritative post-hoc hardware-aware evaluation of the minimum-SWAP route selected by the permutation router. Certified examples exist in both directions:

$$\exists\, C_A : \quad J_E^\star(C_A) < J_P(C_A), \qquad \exists\, C_B : \quad J_P(C_B) < J_E^\star(C_B)$$

The two objective values are therefore not ordered in general. Their certificates also refer to different optimisation criteria. Forced closure certifies the minimum value of J over $\mathcal{R}_E$, whereas the permutation router certifies the minimum number of inserted SWAPs. The explicit counterexamples are reported in Section C.5.

11.6 Timeout semantics

If the execution deadline expires while the exact-search queue is non-empty, the premise required by the queue-exhaustion theorems is false. The best validated complete incumbent remains a feasible upper bound and may be reported and compared empirically, but it is not $J_E^\star$ or $J_J^\star$. In particular, the inequality $J_J^\star \leq J(R_P)$ applies only when the JIT run reaches `queue_exhausted`; it does not generally hold for a timeout incumbent.

11.7 Complexity

The exact router has no polynomial input-size guarantee. Its running time is search-dependent and governed by the number of labels that survive duplicate removal, admissible pruning and dominance, together with the width of the corresponding incomparable frontiers. Forced closure branches primarily through blocked routing decisions, whereas JIT-exact additionally explores dependency-valid choices among simultaneously eligible two-qubit events.

This extra JIT freedom can itself be combinatorial. For example, two independent chains containing m and n routing-relevant events admit $\binom{m+n}{m}$ valid interleavings. Dynamic programming may merge some resulting histories, but different event orders can produce incomparable resource-availability profiles and therefore remain simultaneously active. Max-Plus dominance reduces this frontier when a continuation-safe error–time ordering exists, but it does not change the exponential worst-case character of the search.

For the fixed five-qubit target, all $5! = 120$ initial mappings are explicitly considered. If hardware width n were treated as variable, full-width initialisation alone would introduce $n!$ possible bijections before routing branching is considered. The method should therefore be interpreted as an exact specialised solver for the studied five-qubit architecture rather than as a scalable general-n routing algorithm [7].

Let N denote the cumulative number of committed labels. Native search storage is $O(N)$ up to frontier, queue, exact-map and reconstruction overhead. JIT progress determinacy is important here because the pending non-routing state does not require an additional arbitrary per-label subset: it is reconstructed from projected progress p.

A detailed analysis of the time and space complexity for both the JIT and forced closure algorithms is presented in Section C.6.1 and Section C.6.2, respectively.

11.8 Evidence versus proof

The results above are mathematical arguments about the stated exact models; implementation tests are supporting evidence rather than substitutes for those proofs. Differential, high-precision and end-to-end replay tests are used to check that the native implementation respects the transition, scheduling, dominance and reconstruction assumptions on which the arguments rely. The experimental methodology therefore treats `queue_exhausted`, successful authoritative validation and the selected scheduling mode as explicit certificate fields rather than inferring exactness from repeated objective values or agreement with another router.

Chapter 12

Experimental Methodology

12.1 Experimental goals and comparison scope

The experiment compares JIT-exact with forced-closure J-exact, the permutation-based exact router reconstructed from [1], Qiskit opt3, Qiskit SABRE-only, tket, MQT QMAP heuristic and MQT QMAP exact. All returned complete circuits are evaluated under the same frozen five-qubit hardware calibration and hardware-aware objective J. The comparison does not treat these methods as solvers of one identical optimisation problem: JIT and forced closure minimise J over their respective restricted trace spaces, the permutation-based router and exact QMAP certify their own minimum-SWAP formulations, and the remaining external methods use different placement, routing and transformation objectives.

The empirical analysis addresses four questions. First, it compares JIT with the routing and compilation baselines using the common J evaluator. Second, it examines how the paired difference in J is divided between the gate-error term and the coherence-normalised duration term across circuit scales. Third, it tests how the Qiskit results change under different seed-selection policies and after including or excluding identity-mirror circuits. Fourth, it reports the classical compilation cost and the certificate coverage associated with the returned routes. Forced closure is also included as an ablation of the scheduling semantics.

All reported conclusions refer only to the frozen circuit set and hardware model defined below.

12.2 Frozen corpus and experimental regimes

The routing-quality dataset contains 70 five-qubit OpenQASM 2 circuits. It is formed from seven parameterised circuit families, with two variants for each family and five CZ-count bands, for a total of 14 circuits per band. The input manifest records, for each circuit, the family, variant, natural scale parameter, CZ band, operation counts, and SHA-256 hash. Only the frozen files associated with this manifest enter the analysis.

Table 12.1: Construction of the benchmark corpus. Each family contributes two variants at five scales, for ten circuits. The scale varies the source construction before basis translation.

Family	Variants	Scale parameter
QAOA MaxCut	Five-cycle; complete graph	QAOA depth p
Hardware-efficient VQE	Linear; ring entanglement	Ansatz repetitions
Trotterised Ising evolution	Line; ring coupling	Trotter steps
Continuous-time quantum walk	Path; cycle, XY hopping	Trotter steps
Phase estimation	Single; alternating targets	Controlled-unitary repetitions
Seeded random layers	Brickwork; dense cycle	Random-layer depth
Seeded random mirror	Brickwork; dense cycle	Forward depth before inversion

The scale parameter changes the underlying source construction rather than repeating an already transpiled instruction pattern. The mirror family appends the exact inverse of its seeded forward circuit. This means that its logical unitary is the identity up to numerical precision. It may therefore expose optimisations available only to end-to-end compilers. All numerical angles are frozen in the input OpenQASM 2 files, whose basis is $R_x, R_y, R_z, \mathrm{CZ}$; the manifest records family, variant, scale parameter, realised gate counts and SHA-256 hash for every circuit.

The primary inferential set is defined as bands b01–b02 and contains 28 circuits. Band b03 contains 14 transition instances, while bands b04–b05 contain 28 stress instances. These groups are defined from the input CZ count, not from the observed routing results, and should not be interpreted as physical thresholds for useful computation.

Table 12.2: Input CZ-count bands and their role in the analysis. Each band contains the same seven circuit families, with two variants per family.

Band	CZ interval	Target	Circuits	Role
b01	0–25	20	14	Primary
b02	26–50	40	14	Primary
b03	51–100	75	14	Transition
b04	101–200	150	14	Stress
b05	201–300	250	14	Stress

12.3 Common hardware-aware evaluation

All returned circuits are evaluated under the frozen hardware signature `9da78e` described in Chapter 3 and Appendix B, with q_2 as the star centre and heterogeneous primitive errors and durations. The common evaluator uses the objective introduced in Chapter 5,

$$J = G + \frac{T_{\text{exec}}}{T_{\text{coh,eff}}} = -\ln P_{\text{gates}} + \frac{T_{\text{exec}}}{T_{\text{coh,eff}}}, \qquad P_{\text{model}} = e^{-J} \tag{12.1}$$

with $T_{\text{coh,eff}} \simeq 2.92969\,\mu\text{s}$ for the frozen snapshot. T_{exec} is the physical ASAP makespan of the evaluated route and is unrelated to classical compilation time. Numerical values are stored at full machine precision and rounded only when reported.

The derived quantity P_{model} is a reproducible score defined by the independent-error and coherence-penalty model used in this work. It should not be interpreted as a measured circuit fidelity or as an empirical success probability. The calibrated primitive error values may already include decoherence effects, so the additional makespan penalty can overlap partly with contributions present in the calibration data. The same definition is applied to all compared routes. Further details are reported in Section B.1.

12.3.1 ASAP scheduling and critical path

The common evaluator replays each physical trace with the operation-level ASAP resource scheduler defined in Chapter 5. Operations on disjoint physical qubits may overlap, while the order of operations sharing a physical qubit is inherited from the returned trace; the evaluator does not commute or resynthesise gates to obtain an independently optimised schedule. The resulting duration-weighted critical path defines T_{exec}. Logical depth and gate depth are recorded separately and are not used as substitutes for this physical makespan.

12.3.2 Operation representation and gate counts

External executable outputs are translated without additional routing to the common $R_x, R_y, R_z, \mathrm{CZ}$ physical basis; the original compiler outputs are retained separately. J-exact routes may instead retain explicit SWAPs and the Hadamards used by BRIDGE. Their common evaluation expands these internal actions through the frozen composition-derived costs: one SWAP contributes three CZ gates and six nonvirtual one-qubit pulses, while BRIDGE is already represented by its four CZ gates and four centre Hadamards and is therefore not counted twice.

$$N_{\mathrm{CZ}}^{\mathrm{native}} = N_{\mathrm{CZ}}^{\mathrm{explicit}} + 3N_{\mathrm{SWAP}}, \qquad N_{\mathrm{1q}}^{\mathrm{pulse}} = N_{R_x} + N_{R_y} + N_H + 6N_{\mathrm{SWAP}}. \tag{12.2}$$

The physical primitive count is $N_{\mathrm{phys}} = N_{\mathrm{CZ}}^{\mathrm{native}} + N_{\mathrm{1q}}^{\mathrm{pulse}}$, excluding virtual R_z gates. For external native outputs the corresponding counts are read directly from the translated primitive trace. A routing SWAP count is reported only when the compiler representation exposes it explicitly; in particular, additional CZs in opt3 are not divided by three to infer a fictitious SWAP count.

12.4 Algorithms and budgets

Table 12.3: Compared configurations and per-circuit budgets. A certificate always refers to the method's own objective and admissible trace space.

Method	Configuration and role	Budget
JIT-exact	`jit-exact`; admissible `cheap` bound; validated SABRE-inspired upper bound	One worker; nominal 900 s
Forced closure	Same exact settings with `eager` non-routing scheduling	One worker; nominal 900 s
Permutation	Literature baseline from Gnatowski et al.; minimum SWAPs; no BRIDGE	Nominal 900 s search
Qiskit opt3	Qiskit 2.1.2; calibrated target; level-three end-to-end compilation	Seeds 0–9; best J
SABRE-only	Qiskit 2.1.2; `SabreLayout`; four iterations; one layout and one SWAP trial	Seeds 0–9; best J
tket	pytket 2.18.1; noise-aware placement and routing	One call; placement timeout 1000 ms
MQT heuristic	QMAP 3.9.0; gate-count/max-distance heuristic; CZ adapter	One call
MQT exact	QMAP 3.9.0; minimum-SWAP baseline; same adapter	One call

12.4.1 JIT, scheduling ablation and permutation baseline

JIT-exact and forced closure use the same objective, frozen hardware model, routing-action alphabet, admissible `cheap` lower bound and validated SABRE-inspired incumbent; they differ only in the non-routing scheduling semantics defined in the preceding chapters. Forced closure eagerly materialises the prescribed ready non-routing work, whereas JIT materialises only the missing prerequisites of the selected two-qubit event and applies deterministic terminal completion. Their distinct exactness claims are established in Chapter 11.

The permutation-based baseline reconstructs the method of [1]. It explores dependency-valid two-qubit orders and centre assignments using direct execution and SWAPs, without BRIDGE, and certifies minimum inserted SWAP count within its own formulation. Its complete returned trace is evaluated post hoc with the common J

evaluator. A timeout route remains a feasible incumbent but is not an optimality certificate. The nominal 900 s search limits and their timing boundaries are specified in Section 12.7.

12.4.2 Qiskit opt3 and SABRE-only

Both Qiskit pipelines use release 2.1.2 and the same calibrated five-qubit `Target` [40, 2]. *Opt3* is an end-to-end `transpile` call at optimisation level three and may alter the logical gate network through optimisation and resynthesis. *SABRE-only* denotes the recorded `SabreLayout` pipeline with routing enabled but without an additional general optimisation or synthesis stage; its subsequent level-zero translation uses the selected physical layout with routing disabled and therefore changes representation without launching another routing search.

For each circuit and each Qiskit pipeline, ten candidates are generated using seeds $0, \ldots, 9$. The primary selection policy chooses the candidate with the lowest value of the common objective J. Ties are resolved first by T_{exec}, then by G, and finally by the seed value. The reported compilation cost for this policy includes all ten compiler runs.

Two additional policies are used for sensitivity analysis. The first retains the candidate obtained with seed zero. The second selects a deterministic median representative. For this policy, the ten candidates are ordered by (J, seed). The fifth and sixth candidates in this ordering are then identified, and the one with the smaller seed is selected. All ten raw candidates are retained for audit.

12.4.3 tket and MQT QMAP

The tket baseline uses pytket 2.18.1 with calibrated `NoiseAwarePlacement` followed by `RoutingPass` on the five-qubit star [41]. The placement stage receives the frozen single-qubit and edge error data; the routed output is subsequently decomposed and normalised for common evaluation. Full parameter settings are recorded with the reproducibility artefacts; the retained tket configuration is detailed in Section A.7.1.

MQT QMAP 3.9.0 with MQT Core 3.9.2 is evaluated in heuristic and exact modes [33]. The heuristic uses the `gate_count_max_distance` strategy with dynamic initial placement; the exact mode certifies minimum SWAP count within the QMAP mapping model, not minimum J. Because this QMAP version expects CNOT interactions, each source CZ is temporarily encoded as one CX adjacency token carrying the corresponding CZ duration and error. Mapping is performed on this adjacency-equivalent representation, after which the interaction tokens are restored one-to-one to CZ before SWAP lowering and common evaluation. CNOT-specific transformations are disabled, and equality of the source-CZ, encoded-CX, routed-interaction and decoded-CZ counts is audited. The adapter is therefore used only

to express the same adjacency constraint, not to identify the physical semantics of CZ and CX.

12.5 Validation and certificate semantics

The integrity of the input and the output are checked independently of the optimality certificate. In particular, the circuit and hardware hashes must match the frozen manifests, the initial and final mappings must be valid permutations of the five physical qubits, and every physical two-qubit operation (including any retained internal SWAP) must respect the star coupling graph.

The full 32×32 unitary matrices check the semantic equivalence. Let P_i and P_f denote the permutation operators associated with the initial and final mappings, respectively. The routed circuit is expected to satisfy

$$U_{\text{routed}} \simeq e^{i\phi} P_f U_{\text{input}} P_i^{\dagger} \tag{12.3}$$

where ϕ is a global phase. The global phase is aligned through the complex Hilbert–Schmidt inner product and the maximum matrix-element discrepancy must not exceed 10^{-10}. At width five this full-unitary check is practical and verifies the complete transformation, including initial and final layout permutations; it is not proposed as a scalable validation method for large circuits.

The audit recomputes topology, mappings, operation counts and J from the stored routed circuits. Moreover, it verifies the corresponding input, output and analysis manifests. Compiler exit status and internal optimality flags are not used as substitutes for semantic validation. Even if a routed circuit does not carry an optimality certificate, it may still be semantically valid.

For J-exact, a run that terminates with `queue_exhausted` certifies the minimum value of J within the selected exact trace space. The permutation-based router and exact QMAP instead certify optimality for their respective minimum-SWAP formulations. The heuristic baselines do not provide a certificate of J-optimality. Route quality and certificate coverage are therefore reported separately.

12.6 Paired analysis and uncertainty

The circuit is the unit of comparison. Each circuit has equal weight, and the paired effect against baseline b is

$$\Delta J = J_b - J_{\text{JIT}} = \Delta G + \Delta C, \quad \Delta G = G_b - G_{\text{JIT}}, \quad \Delta C = \frac{T_b - T_{\text{JIT}}}{T_{\text{coh,eff}}} \tag{12.4}$$

A positive value favours JIT. Under the model, $\Delta J = \ln\left(P^{\text{JIT}}_{\text{model}} / P^{b}_{\text{model}}\right)$; absolute model-survival values are reported separately. Absolute probabilities are considered

alongside this relative change. Gate counts, SWAP counts, and classical runtimes are reported as descriptive quantities and are not included as additional terms in J.

The analysis reports JIT win, tie, and loss counts using a numerical tie tolerance of 10^{-12}. It also reports the mean, median, and quartiles of the paired differences ΔJ. Since

$$\Delta J = \Delta G + \Delta C$$

for each circuit pair, the component means sum exactly to the mean total difference. In general, this is not true for the medians. The reported medians are not computed from differences between separately aggregated baseline and JIT values, but they are computed directly from the paired quantity of interest.

A percentile bootstrap is used in order to get confidence intervals for paired medians. Complete circuit pairs are sampled with replacement for 20,000 replicates. Two-sided Wilcoxon signed-rank tests are computed with SciPy's `wilcox`, using its default zero handling, no continuity correction, and `method="auto"` [42, 43]. Bootstrap seeds and circuit ordering are fixed and recorded. The signed-rank is not a test of algorithmic equivalence. It just evaluates the paired differences under its statistical assumptions.

The primary inferential family is composed by six comparisons between JIT and the other methods on b01–b02: permutation, Qiskit opt3, SABRE-only, tket, MQT heuristic, and MQT exact. Their six two-sided nominal p-values are adjusted jointly by the Holm procedure [44]. Forced closure is treated as a secondary internal ablation; overall, band-level and family-level analyses are likewise secondary or descriptive and are not added to this primary multiple-testing family.

Qiskit seed-policy sensitivity is evaluated by rebuilding the selected rows under the minimum-J, seed-zero and deterministic-median policies from the retained ten-candidate sets and requiring exact agreement with the saved selections. A second sensitivity analysis removes the entire `mirror_random` family from opt3 comparisons while preserving the band definitions and selection rules. Both use 20,000-replicate bootstrap intervals and introduce no additional hypotheses into the six-comparison primary family.

Because the recurring families and variants form a structured benchmark rather than an i.i.d. population sample, a supplementary family-block bootstrap resamples the seven complete families with replacement, preserving both variants and all primary-band observations belonging to each selected family. It uses 20,000 replicates; with only seven blocks, its intervals are interpreted as a robustness check rather than a high-resolution population estimate. Neither a bootstrap interval spanning zero nor a nonsignificant signed-rank test is interpreted as evidence of equivalence.

12.7 Timing protocol and reproducibility

All eight methods were executed serially on the same PC under comparable conditions but in two local environments. JIT, forced closure and permutation used CPython 3.11.15; Qiskit opt3, SABRE-only, tket and both MQT modes used CPython 3.12.13. The recorded host runs Windows 11 with 20 logical processors; one numerical thread and one Qiskit process are used where applicable. Exact package versions and measured timing boundaries are stored with the benchmark records.

Compilation time is defined from the method-specific measured stages rather than from complete command wall time. For Qiskit, policy cost includes all ten compiler calls and, for SABRE-only, the recorded native translation stage; tket and MQT include their recorded placement/routing, lowering and normalisation stages. Validation, file I/O and adapter work outside the measured regions are excluded. Selected-candidate time is reported separately and is never substituted for the cost of obtaining a best-of-ten result.

Timing summaries report calls per circuit, median, mean, total and maximum observed compilation time together with certificate and timeout counts. Certified and timeout runs remain in the observed-time summaries, but timeout cases are right-censored with respect to time to certification. Indeed, a short observed runtime or a small median cannot by itself be interpreted as the time required to prove optimality.

For each circuit, deterministic configurations are run once. Their timing summaries are not estimates of run-to-run variability but describe the observed compiler runtimes. For Qiskit, the ten seeds do not define ten repeated measurements of a single deterministic compilation, but a multi-start selection policy.

Classical compilation time and quantum execution makespan T_{exec} are reported as separate quantities. Search-work counters produced by different algorithmic formulations are also reported separately and are not interpreted as directly comparable measures of memory use or computational complexity.

Analysis scripts reconstruct tables, figures and numerical summaries from the stored JSON, CSV and QASM artefacts without invoking a router or compiler. The retained provenance includes input and output hashes, analysis manifests, candidate-selection records and bootstrap seeds; route validity and primitive counts are obtained from the independent QASM audit.

The empirical scope remains limited to one five-qubit star topology, one frozen calibration, the selected routing models, compiler-dependent lowering and the constructed corpus. The surrogate hardware model omits readout, leakage, crosstalk and calibration drift and may overlap active-time decoherence with calibrated gate

errors. Stress circuits remain useful for studying routing and solver behaviour even when the model assigns negligible survival. The experiment therefore supports controlled relative comparisons under the stated evaluator, not live-QPU fidelity claims or asymptotic conclusions.

Chapter 13

Experimental Results and Discussion

13.1 Campaign completeness and correctness

The benchmark contains one selected valid output for each of the 70 circuits and eight methods, for a total of 560 matched route records. Both J-exact scheduling modes terminate with `queue_exhausted` on all 70 circuits. This means that they certify the minimum value of J within their respective exact models.

The permutation-based router returns 49 minimum-SWAP certificates and 21 valid incumbent routes obtained at timeout. All 28 circuits in the primary b01–b02 set are certified. In each of b03, b04, and b05, seven runs are certified and seven terminate at the time limit with a valid incumbent. Timeout routes remain complete feasible outputs and are therefore retained in applicable route-quality and observed-time summaries, but not treated as certified optima.

All Qiskit best-of-ten, fixed-seed and deterministic-median selections are reconstructed exactly from the retained candidate records, providing an independent check of the seed-selection analysis reported below.

Independent verification confirms the frozen input and output hashes, hardware signature, mappings and star connectivity of the selected routes. Full 32×32 unitary validation gives a largest phase-aligned matrix-element discrepancy of 1.54×10^{-11}, below the 10^{-10} threshold, while independently replayed J differs from the stored score by at most 3.55×10^{-15}. No selected route fails the semantic or hardware audit. Circuit validity is therefore established independently of optimality certification.

13.2 Unified paired comparison against JIT

Table 13.1 and Figure 13.1 compare the seven methods with JIT using

$$\Delta J = J_b - J_{\mathrm{JIT}}$$

Therefore, positive values indicate a lower objective for JIT.

Even if the methods do not solve the same optimisation problem, the common J evaluator allows the returned routes to be compared on the same output metric. Forced closure is used as a scheduling-semantics ablation, while the five routing baselines use different routing objectives or heuristic procedures. Qiskit opt3 is included as an end-to-end compilation control.

On the primary b01–b02 set, JIT has lower objective values than the permutation-based router and tket in the paired comparison. The win/tie/loss counts are 28/0/0 and 23/0/5, respectively, and the median ΔJ is positive in both cases.

The comparison with MQT exact also gives a positive median ΔJ on the primary set. The associated Holm-adjusted signed-rank result is $p_{\mathrm{Holm}} = 0.0131$. MQT heuristic is less uniform: its median is 0.074 611 and its percentile bootstrap interval $[-0.0031, 0.2086]$ slightly crosses zero, whereas the signed-rank result remains significant after Holm adjustment, $p_{\mathrm{Holm}} = 0.0074$. The two summaries answer different statistical questions, so this is not a contradiction.

Best-of-ten SABRE-only is the closest primary routing comparator. JIT wins on ten primary circuits and loses on eighteen; the median paired difference is $-0.018\,464$, with bootstrap interval $[-0.0408, 0.0405]$, while the mean is slightly positive at 0.008 407. The two-sided signed-rank result is $p_{\mathrm{nom}} = p_{\mathrm{Holm}} = 0.7499$. No systematic primary advantage for JIT is therefore detected under this ten-start policy, but this must not be interpreted as evidence of equivalence because no equivalence margin was defined. Seed-policy dependence is analysed separately in Section 13.4.

Opt3 produces lower J than JIT on all 28 primary circuits, with median paired difference $-0.735\,699$ and interval $[-0.8199, -0.4000]$. Because opt3 can cancel, rewrite and resynthesise the logical gate network in addition to placing and routing it, this end-to-end advantage cannot be attributed specifically to routing or to any individual transpiler pass.

Across the six prespecified primary comparisons in the Holm family, the adjusted signed-rank results retain evidence against the null for permutation, tket, both MQT configurations and opt3; SABRE-only does not. Forced closure remains outside this primary multiplicity family as an internal scheduling ablation. Statistical significance is interpreted together with paired effect size and direction rather than as evidence of practical importance by itself.

Table 13.1: Unified paired comparison with JIT. Positive values of $\Delta J = J_b - J_{\mathrm{JIT}}$ indicate a lower objective value for JIT. The intervals are 95% percentile bootstrap confidence intervals for the paired median, based on 20,000 replicates.

Comparator	JIT W/T/L	Mean ΔJ	Median ΔJ	95% interval	p_{nom}	p_{Holm}
Primary: b01–b02, 28 circuits						
Internal ablation						
Forced closure	26/1/1	0.040	0.027	[0.014, 0.051]	7.02×10^{-6}	—
Routing baselines from the literature						
Permutation	28/0/0	0.114	0.080	[0.034, 0.135]	7.45×10^{-9}	4.47×10^{-8}
SABRE-only	10/0/18	0.008	−0.018	[−0.041, 0.041]	0.7499	0.7499
tket	23/0/5	0.136	0.053	[0.027, 0.168]	2.38×10^{-5}	9.52×10^{-5}
MQT heuristic	18/0/10	0.177	0.075	[−0.003, 0.209]	0.0025	0.0074
MQT exact	19/0/9	0.138	0.048	[0.001, 0.167]	0.0065	0.0131
End-to-end compiler control						
Qiskit opt3	0/0/28	−0.755	−0.736	[−0.820, −0.400]	7.45×10^{-9}	4.47×10^{-8}
Overall: 70 circuits (secondary)						
Internal ablation						
Forced closure	65/3/2	0.183	0.060	[0.028, 0.123]	1.53×10^{-12}	—
Routing baselines from the literature						
Permutation	70/0/0	0.858	0.198	[0.118, 0.380]	3.56×10^{-13}	—
SABRE-only	44/0/26	0.311	0.086	[0.033, 0.183]	3.21×10^{-5}	—
tket	58/0/12	0.510	0.168	[0.086, 0.247]	9.45×10^{-11}	—
MQT heuristic	48/0/22	0.711	0.286	[0.075, 0.491]	1.07×10^{-6}	—
MQT exact	47/0/23	0.607	0.179	[0.041, 0.387]	7.34×10^{-5}	—
End-to-end compiler control						
Qiskit opt3	0/0/70	−2.484	−1.460	[−1.899, −1.069]	3.56×10^{-13}	—

p_{nom} denotes the nominal two-sided Wilcoxon p-value, while p_{Holm} denotes the Holm-adjusted value for the six primary comparisons, excluding forced closure. A dash indicates that the comparison is not included in this family. Overall tests and the forced-closure ablation are reported as secondary analyses. Both Qiskit pipelines use the best-of-ten selection policy, and all valid permutation-router incumbents are retained.

The complete 70-circuit summaries are secondary to the prespecified primary analysis. JIT has a positive paired median across the full corpus against SABRE-only, tket, both MQT modes and every returned permutation route. However, opt3 remains lower in J on all 70 circuits. In particular, the complete-corpus advantage over SABRE-only must not replace the mixed b01–b02 result, because 42 observations belong to the transition or stress regimes defined outside the primary inferential set.

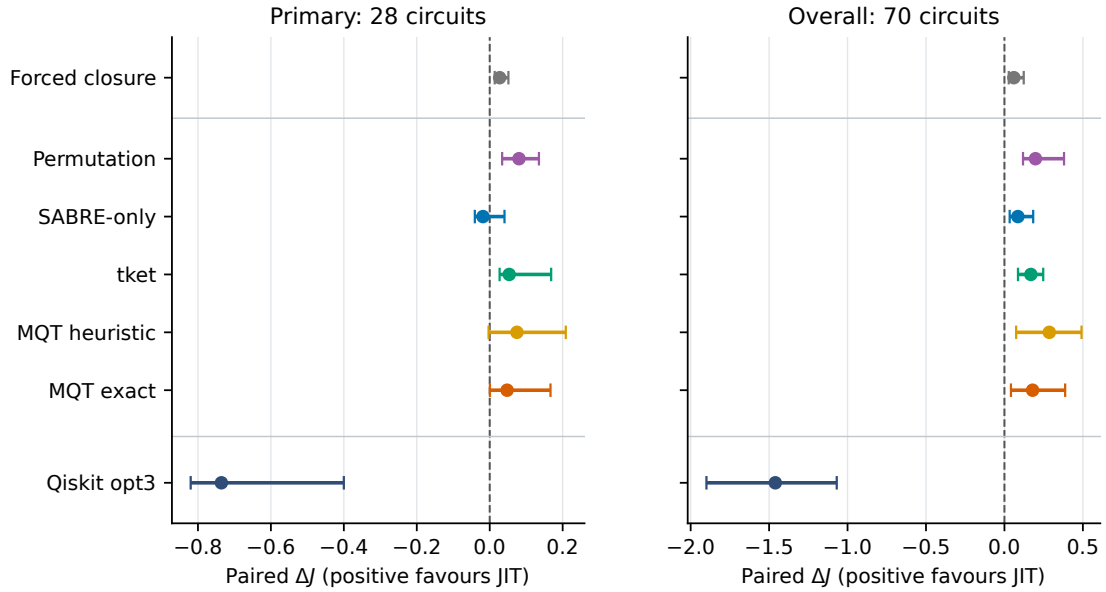


Figure 13.1: Median paired differences and 95% bootstrap intervals for all seven comparators. The separators distinguish forced closure, five routing baselines from the literature, and the opt3 end-to-end control. Each panel includes complete circuit pairs; positive values favour JIT.

13.2.1 Absolute predicted execution quality

Table 13.2: Absolute metrics on the 28 primary circuits. Each entry is an independently computed median; T_{exec} is in μs.

Method	J	T_{exec}	P_{gates}	P_{model}	N_{CZ}	N_{phys}
JIT-exact	1.850	4.648	0.791	0.1614	44.0	182.0
Forced closure	1.878	4.728	0.791	0.1570	44.0	182.0
Permutation	1.922	4.818	0.772	0.1499	44.0	182.0
SABRE-only	1.856	4.730	0.785	0.1608	44.0	182.0
tket	2.028	5.136	0.764	0.1365	44.0	202.0
MQT heuristic	2.113	5.324	0.749	0.1224	48.5	182.0
MQT exact	2.008	5.080	0.760	0.1363	48.5	182.0
Qiskit opt3	1.178	2.928	0.835	0.3078	33.5	92.5

N_{phys} counts native CZ gates and nonvirtual one-qubit pulses. Virtual R_z gates are excluded. Median model survival is computed from circuit-level probabilities, not by exponentiating median J.

Table 13.2 places the paired comparisons on the absolute scale of the frozen surrogate model. On the primary set, median JIT makespan is 4.648 μs, compared with

4.730 μs for SABRE-only and 2.928 μs for opt3; their corresponding separately aggregated median model survivals are 0.1614, 0.1608 and 0.3078. These absolute aggregates are descriptive and must not be confused with the median paired differences reported above.

These survival values are outputs of the frozen surrogate model, not measured device fidelities or probabilities of obtaining the correct algorithmic answer. Likewise, $e^{\Delta J}$ is only a relative model-survival ratio and can be large even when both absolute model survivals are too small to indicate a practically useful execution regime.

13.3 Circuit scale and decomposition of the objective

Figure 13.2 reports the paired differences across the five input-CZ bands. The same circuit families appear at progressively larger natural scales, so the figure is used to describe scale-dependent behaviour within this dataset rather than to infer a general scaling law. Qiskit opt3 is shown separately because its end-to-end differences lie on a different numerical scale.

For SABRE-only, the median paired difference is slightly negative in b01–b02 and positive in b03–b05. This change across the bands accounts for the difference between the primary-set result and the complete-corpus summary. Permutation, tket and both MQT configurations have positive median ΔJ in every band, although individual outcomes remain heterogeneous. Neither a positive band median nor increasing CZ count implies a circuit-level win or a monotonic measure of routing difficulty.

Family-block resampling gives the same qualitative direction for the permutation-based router and tket, and the same negative direction for Qiskit opt3. For SABRE-only and both MQT configurations, the primary family-block intervals include zero. The corresponding interval for the forced-closure ablation remains positive. Because only seven family blocks are available, these intervals are treated as a structural robustness check rather than population-level inference.

The primary mean decomposition in Figure 13.3 shows that much of the observed difference is associated with the coherence-normalised makespan term rather than gate cost alone. The forced-closure ablation is the clearest example: $\overline{\Delta G} = 0.000\,053$ while $\overline{\Delta C} = 0.039\,448$, indicating that the scheduling semantics act primarily through duration. Permutation, tket and both MQT modes have positive mean contributions from both components.

SABRE-only exhibits a genuine trade-off: its positive gate-cost contribution 0.012 113 is partly offset by the negative duration contribution $-0.003\,706$, yielding a small mean total despite a negative paired median. Opt3 improves both components,

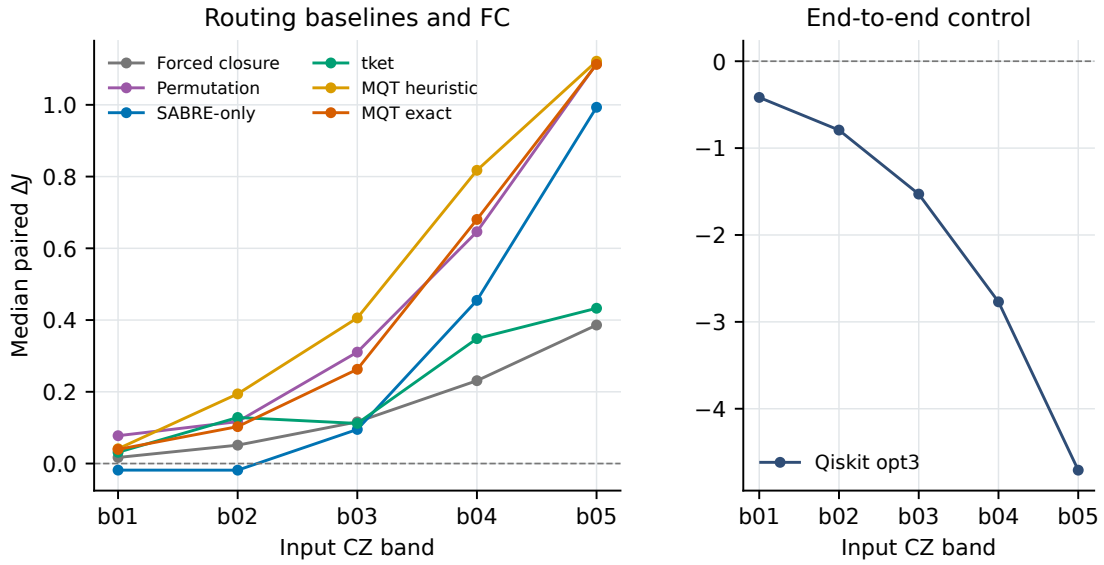


Figure 13.2: Median paired ΔJ by input CZ band. The left panel reports forced closure and the five literature routing baselines, while the right panel reports Qiskit opt3. Each point represents the median of 14 paired circuit comparisons. Connecting lines should not be interpreted as fitted scaling laws.

with $\overline{\Delta G} = -0.083\,110$ and $\overline{\Delta C} = -0.671\,842$. These decompositions describe differences between complete returned traces and do not isolate causal contributions of placement, routing or synthesis.

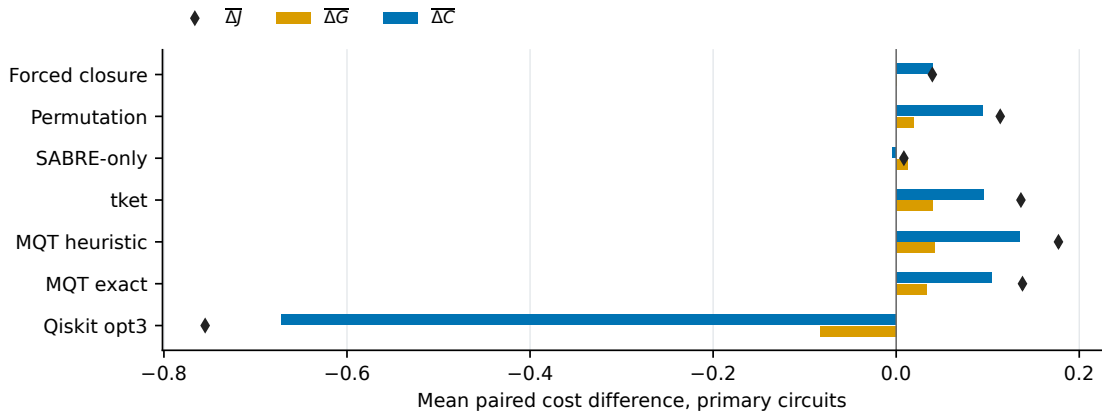


Figure 13.3: Primary mean paired cost differences. Bars show gate-error and coherence-normalised duration terms; diamonds mark their sum. Component means add to the mean total, whereas independently computed component medians generally do not add to the median total.

Absolute quality deteriorates substantially in the stress regime. For JIT, the b04–b05 median makespan is 31.206 μs and median model survival is 4.52×10^{-6}. At this scale, relative route-quality differences remain valid comparisons of the surrogate objective, but do not by themselves establish physical usefulness.

13.4 Seed and mirror sensitivity

Table 13.3 reports fixed seed zero explicitly alongside best-of-ten and the deterministic median representative. The representative is the lower-seed member of the fifth–sixth pair after sorting by (J, seed); no floating-point distance from an arithmetic median is used. It is selected from all ten candidates, so it is not equivalent to choosing a seed before compilation.

Table 13.3: Qiskit seed-policy sensitivity against JIT. Positive ΔJ favours JIT; intervals use 20,000 bootstrap replicates.

Pipeline	Selection policy	JIT W/T/L	Median ΔJ	95% interval
Primary: 28 circuits				
Opt3	Best of ten	0/0/28	−0.7357	[−0.8199, −0.4000]
Opt3	Fixed seed 0	0/0/28	−0.7347	[−0.7924, −0.3633]
Opt3	Median representative	0/0/28	−0.7347	[−0.7924, −0.3823]
SABRE-only	Best of ten	10/0/18	−0.0185	[−0.0408, 0.0405]
SABRE-only	Fixed seed 0	20/0/8	0.0571	[0.0198, 0.0874]
SABRE-only	Median representative	22/0/6	0.0621	[0.0119, 0.0978]
Overall: 70 circuits				
Opt3	Best of ten	0/0/70	−1.4604	[−1.8993, −1.0689]
Opt3	Fixed seed 0	0/0/70	−1.4308	[−1.7617, −0.9729]
Opt3	Median representative	0/0/70	−1.4256	[−1.7593, −0.9380]
SABRE-only	Best of ten	44/0/26	0.0858	[0.0334, 0.1833]
SABRE-only	Fixed seed 0	58/0/12	0.1954	[0.1011, 0.3765]
SABRE-only	Median representative	61/0/9	0.2070	[0.1140, 0.3624]

The median representative is the lower-seed member of ranks five and six after sorting by (J, seed). Fixed seed zero does not use multi-candidate selection; the representative is selected from all ten candidates. These sensitivity controls add no tests to the primary Holm family.

On the primary set, the best-of-ten policy gives JIT 10/0/18, with median −0.0185. Instead, with a seed fixed to zero gives 20/0/8, with median 0.0571. The deterministic median-representative policy gives 22/0/6, with median 0.0621.

For the fixed-seed, the bootstrap intervals and the deterministic median-representative

policies are both positive. For the best-of-ten policy, the corresponding interval includes zero. Thus, the primary JIT–SABRE conclusion is materially restart-sensitive: the detected systematic JIT advantage on the SABRE baseline can be removed by selecting the lowest-J candidate from ten starts.

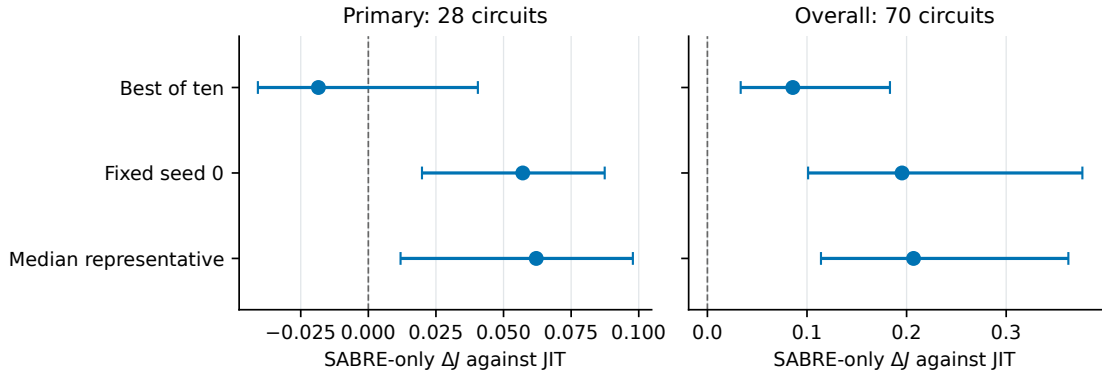


Figure 13.4: SABRE-only selection-policy sensitivity against JIT. The primary best-of-ten interval spans zero; the fixed-seed-zero and median-representative intervals are positive. All intervals use the paired-median bootstrap protocol.

Across the complete corpus, the direction favouring JIT is stable under all three SABRE selection policies, although its magnitude changes; the best-of-ten result must therefore be interpreted together with the classical cost of generating all ten candidates.

When the entire `mirror_random` family is removed, the opt3 result changes a bit. After this exclusion, 24 circuits remain in the primary set and 60 in the complete set. However, under each selection policy, opt3 still attains a lower value of J than JIT.

For the best-of-ten policy, the primary median is -0.6570, with interval $[-0.7807, -0.3899]$, while the overall median is -1.4385.

The lower J values obtained by opt3 are therefore not restricted to the identity-mirror family. The comparison nevertheless remains an end-to-end compilation comparison rather than a routing-only comparison.

Table 13.4: Opt3 sensitivity after excluding the entire `mirror_random` family. Counts remain JIT wins/ties/losses.

Selection policy	JIT W/T/L	Mean ΔJ	Median ΔJ	95% interval
Primary without mirror: 24 circuits				
Best of ten	0/0/24	−0.6900	−0.6570	[−0.7807, −0.3899]
Fixed seed 0	0/0/24	−0.6595	−0.5797	[−0.7778, −0.3357]
Median representative	0/0/24	−0.6610	−0.5816	[−0.7778, −0.3454]
Overall without mirror: 60 circuits				
Best of ten	0/0/60	−2.4902	−1.4385	[−1.8138, −0.9563]
Fixed seed 0	0/0/60	−2.4276	−1.3975	[−1.6428, −0.8344]
Median representative	0/0/60	−2.4388	−1.3975	[−1.7057, −0.8881]

13.5 Classical compilation cost on the common host

Table 13.5: Classical compilation cost on the common host, over all 70 circuits. All times are in seconds; policy time includes all calls used for selection.

Method	Calls	Median	Mean	Total	Maximum	Cert.	Timeouts
JIT-exact	1	0.421	1.289	90.20	19.784	70/70	0
Forced closure	1	0.633	3.212	224.87	62.564	70/70	0
Permutation	1	0.023	283.049	19 813.43	900.088	49/70	21
SABRE-only	10	0.322	0.381	26.66	1.929	—	0
tket	1	0.039	0.089	6.23	1.614	—	0
MQT heuristic	1	0.040	0.087	6.07	1.595	—	0
MQT exact	1	0.507	0.764	53.48	1.883	70/70	0
Qiskit opt3	10	0.790	0.923	64.59	3.440	—	0

JIT, FC and Permutation use a nominal 900 s solve budget. Certificates refer to each method's own objective and space; — means no exact certificate.

Table 13.5 summarises the observed policy-stage compilation costs under the common-host protocol of Section 12.7. These are descriptive single-campaign measurements, not repeated-run timing distributions, and remain distinct from the physical circuit makespan T_{exec}.

JIT and forced closure have median policy times 0.4210 s and 0.6331 s, respectively, and both certify all 70 instances within the configured budget. These measurements do not isolate a runtime effect of Max-Plus dominance or any other individual solver component. The permutation-based router has a much smaller median, 0.0231 s,

but its 21 deadline-limited searches produce a heavy upper tail and incomplete certificate coverage; the mean and total reported in Table 13.5 therefore convey information that the median alone obscures.

The Qiskit policy times include all ten compiler calls used by the best-of-ten selection, giving medians 0.7899 s for opt3 and 0.3219 s for SABRE-only; selected-candidate time is not substituted for the cost of obtaining the policy result. Tket and the two MQT modes use one call per circuit. Their classical costs must therefore be interpreted together with their different optimisation objectives, restart policies and certificate semantics rather than as timings of one common algorithmic task.

13.6 Summary of empirical findings

On the prespecified primary set, the results favour JIT over the tested permutation, tket and MQT baselines under the common hardware-aware evaluator, while best-of-ten SABRE-only remains a close and seed-sensitive routing comparator. Opt3 yields lower J throughout the primary and complete corpora and remains superior after removal of the identity-mirror family, but its broader end-to-end transformations prevent attributing this result specifically to routing. Objective decomposition shows that scheduling-dependent duration is a substantial component of several paired differences, and the complete-corpus band trends confirm that comparator behaviour changes with circuit scale.

Semantic validity, route quality and optimality certification remain distinct outcomes. Every selected route passes the independent semantic audit; both J-exact scheduling modes certify all instances in their own trace spaces, whereas the permutation-based router returns 21 valid but uncertified timeout incumbents. These findings are restricted to the constructed 70-circuit corpus, the frozen five-qubit star and the adopted surrogate objective. Broader device-noise limitations, statistical sampling limitations and the absence of live-QPU validation are discussed with the overall thesis limitations in Chapter 14.

Chapter 14

Conclusions and Future Work

14.1 Summary

This thesis studied hardware-aware qubit mapping and exact routing for a five-qubit superconducting star with heterogeneous primitive errors, operation durations and coherence parameters. Restricted connectivity makes a logical leaf–leaf interaction executable through alternative mapping-changing SWAP-and-execute actions or a mapping-preserving BRIDGE construction, so minimising inserted SWAP count alone does not determine the preferred physical route. Complete routed circuits are evaluated with the hardware-aware objective

$$J = -\ln(P_{\text{gates}}) + \frac{T_{\text{exec}}}{T_{\text{coh,eff}}}$$

The first term is the additive cost derived from the physical gate errors. The second term depends on the makespan of the ASAP schedule over the physical resources. In this work, J is used as a fixed calibrated surrogate for comparing routed circuits. It is not intended as a complete model of device fidelity.

Feasible incumbent construction and optimality proof are treated as separate parts of the search in the exact framework. It also supports two different semantics for scheduling non-routing operations. Forced closure deterministically materialises the non-routing work prescribed by its closure rule before the next routing decision, whereas JIT-exact advances through routing-relevant two-qubit events, materialises only the missing prerequisites of the selected event and completes residual work through a deterministic terminal operation. Queue exhaustion certifies the

minimum J only within the restricted route-and-schedule space defined by the selected semantics.

The two main theoretical contributions are the continuation-safe Max-Plus dominance relation and the formal embedding of the canonical permutation-based route into the JIT route space.

Labels within the same frontier share the same future routing state. In this case, the transformed coordinates $z_i = G + \kappa t_i$ provide a criterion that combines accumulated gate cost with physical-resource availability. If one label is no worse than another in every Max-Plus coordinate, then it cannot produce a larger objective value under any continuation that is legal for both labels. The compact JIT representation is supported by progress determinacy, which proves that projected two-qubit progress uniquely determines the pending non-routing obligations. On the common supported domain, the canonical route R_P reconstructed by the permutation-based IQM Spark router is representable within the JIT model, so JIT queue exhaustion implies

$$J_J^\star \leq J(R_P).$$

The final experimental campaign evaluates 70 five-qubit circuits with eight methods under a common frozen hardware-aware evaluator: JIT-exact, forced closure, the reconstructed permutation-based exact router, Qiskit opt3, Qiskit SABRE-only, tket, MQT QMAP heuristic and MQT QMAP exact. The 28 circuits in bands b01–b02 form the prespecified primary inferential set, while the remaining circuits provide transition and stress regimes. All selected outputs pass the independent semantic and hardware audit. Both J-exact scheduling modes reach `queue_exhausted` on all 70 circuits, whereas the permutation-based router obtains 49 minimum-SWAP certificates and returns 21 valid timeout incumbents.

On the primary set, the paired signed-rank analyses favour JIT over the tested permutation, tket and MQT baselines under the common J evaluator, subject to the bootstrap and family-block qualifications reported in Chapter 13. Best-of-ten SABRE-only is a much closer routing comparator: JIT wins ten primary instances and loses eighteen, and no systematic primary advantage for JIT is detected under the ten-start selection policy. Fixed-seed and deterministic-median SABRE controls instead favour JIT, showing that this comparison is materially restart-sensitive. Qiskit opt3 produces lower J than JIT on every primary and complete-corpus circuit, and this direction remains after removing the identity-mirror family; because opt3 performs end-to-end optimisation and resynthesis in addition to placement and routing, this result cannot be attributed specifically to routing. The forced-closure ablation further shows that non-routing scheduling semantics can materially affect the duration component of the hardware-aware objective.

14.2 Answers to the research questions

RQ1: hardware-aware route evaluation. A route can be evaluated using a state that combines additive primitive-gate cost with physical-resource availability. The negative logarithm transforms the product of primitive success factors into the additive quantity G, while the ASAP makespan contributes through $T_{\mathrm{exec}}/T_{\mathrm{coh,eff}}$. In this way the objective can be updated incrementally during exact search and independently recomputed after reconstruction. The result is a reproducible hardware-aware surrogate for comparing complete routes under one frozen calibration. It is not a complete device-noise model or a measured probability of algorithmic success.

RQ2: continuation-safe Max-Plus dominance. The dynamic state of labels that represent the same future routing problem can be expressed through the resource-wise coordinates $z_i = G+\kappa t_i$. If one label is no worse than another in every transformed coordinate, the monotonicity and additive homogeneity of the ASAP continuation imply that its final value of J cannot be larger under any complete continuation that is legal for both labels. This criterion applies in cases that are not resolved by classic dominance. In particular, a larger accumulated gate cost can be offset by sufficiently earlier physical-resource availability. In order to avoid unsound pruning decision, the implemented test is deliberately conservative under binary64 arithmetic. In this way uncertain comparisons fail closed and retain both labels.

RQ3: JIT scheduling and compact exact state. JIT-exact can enlarge the admissible non-routing scheduling freedom without introducing an arbitrary pending-work subset into every search label. The JIT progress-determinacy result proves that projected two-qubit progress uniquely determines which non-routing prerequisites have already been materialised and which remain pending. The discrete frontier key can therefore remain $K = (p, m)$, while accumulated gate cost and physical-resource availability describe the dynamic history. Complete enumeration of eligible two-qubit events and their legal routing actions, followed by deterministic terminal completion, yields an exact search over the restricted JIT trace space; `queue_exhausted` therefore certifies its minimum J.

RQ4: formal and empirical comparison with alternative routing strategies. The relationship with the permutation-based IQM Spark router can be stated formally rather than only measured empirically. On the common supported domain, its canonical complete route R_P embeds into the JIT route space, and therefore JIT queue exhaustion guarantees $J^\star_J \leq J(R_P)$ under the common hardware-aware evaluator. No corresponding universal ordering holds between

forced closure and JIT, because their restricted scheduling spaces are different. Empirically, the broader eight-method campaign shows that the conclusion depends on the comparator and selection policy. On the primary b01–b02 set, JIT is favoured against the permutation-based router and tket and by the signed-rank analyses against both MQT configurations, with weaker robustness in the bootstrap summaries for the MQT comparisons. Best-of-ten SABRE-only remains statistically unresolved and materially seed-policy dependent. Qiskit opt3 returns lower J on every tested circuit, but its end-to-end optimisation and resynthesis make this a broader compilation result rather than an isolated routing comparison.

14.3 Main contributions and limitations

The contributions of the thesis can be ordered as follows:

1. **Continuation-safe Max-Plus dominance.** The transformed coordinates $z_i = G + \kappa t_i$ yield a hardware-aware dominance relation that safely removes some error–time trade-offs left incomparable by classic Pareto dominance.

2. **Formal embedding of permutation routing into JIT-exact.** On the common supported domain, every canonical route produced by the permutation-based IQM Spark reconstruction is representable in the JIT route space, yielding $J^\star_J \leq J(R_P)$ whenever the JIT search reaches queue exhaustion.

3. **JIT-exact scheduling semantics with progress determinacy.** Non-routing work can be deferred across routing decisions while projected two-qubit progress still determines the pending obligations required by the exact state and continuation arguments.

4. **A hardware-aware exact-routing framework for the five-qubit star.** The framework combines the calibrated objective J, exhaustive initial mappings, the restricted direct/SWAP/BRIDGE action model, validated heuristic upper bounds, sound reductions and explicit certificate semantics for forced closure and JIT-exact.

5. **A controlled eight-method empirical evaluation.** Seventy circuits from seven structured families are evaluated with a common physical-cost model, independent semantic validation, paired statistical analysis, multiple-comparison control, seed-policy sensitivity and family-block robustness checks. Empirically, the broader eight-method campaign shows that the conclusion depends on the comparator and selection policy.

The claims remain deliberately bounded. Exactness concerns five logical qubits on the frozen five-qubit star, the supported OpenQASM normalisation, the restricted direct/SWAP/BRIDGE action model, the adopted ASAP resource scheduler and

the selected non-routing semantics. The hardware-aware objective is a calibrated surrogate rather than a validated live-QPU fidelity model and omits effects such as readout error, leakage, crosstalk and calibration drift. The external methods also do not solve one common mathematical optimisation problem: in particular, the permutation-based and exact-QMAP certificates concern their own minimum-SWAP formulations, while opt3 performs transformations beyond routing.

The empirical conclusions are similarly corpus-specific. The final study contains 70 constructed five-qubit circuits from seven recurring families, with 28 prespecified primary instances, one frozen hardware calibration and one local execution campaign. Only seven blocks are available for the family-level robustness analysis, and deterministic compiler times are observational single-campaign measurements rather than repeated timing distributions. Although both J-exact modes certify every circuit in this corpus, this does not establish scalability: the worst-case search remains exponential and the fixed five-qubit target avoids the factorial and frontier-dimensional growth expected when hardware width increases.

14.4 Future work

Scaling the exact search beyond the present corpus. Both J-exact scheduling modes certify all 70 circuits in the final campaign, but this finite result should not be confused with a scalability guarantee. Larger circuits, wider devices and richer dependency structures can increase routing branching, dependency-valid event orderings and the width of incomparable frontiers. Future work should therefore investigate stronger admissible residual bounds together with safe structural reductions such as partial-order reduction, symmetry handling, component-aware bounds and reusable subproblem summaries. Pattern databases and relaxed scheduling bounds are natural candidates provided that their admissibility or continuation-preservation properties are proved before they are allowed to prune the exact search [32, 34].

Broader coupling graphs and hardware width. Extending the framework beyond the five-qubit star requires a complete action generator, a suitable mapping representation, and a new proof that the successor construction is complete for the target coupling graph. The Max-Plus continuation result is not limited in principle to five physical resources. Indeed it can be applied to larger systems when the future transition map preserves the required monotonicity and additive homogeneity properties. The permutation-route embedding is more specific. The present proof depends on the relation between the supported permutation router and the star architecture. Therefore, a different coupling graph needs a separate embedding argument. Increasing device width also enlarges the initial-placement space and the dimension of the resource frontier, so algorithmic reductions become

increasingly important [25, 33].

Hardware validation and objective design. The surrogate objective should be tested against measured circuit quality on physical hardware across controlled calibration snapshots and shot budgets. Such experiments would quantify how well the primitive negative-log term and coherence-normalised ASAP makespan predict observed behaviour and would reveal whether effects such as readout, leakage, crosstalk or idle-history dependence require additional terms [10, 11, 12]. The same study could examine whether a fixed scalarisation is sufficient or whether route selection is better represented by a Pareto frontier over gate risk, makespan and routing actions.

Restart budgets, warm starts and controlled compiler ablations. Through the sensitivity analysis, it is possible to conclude that the restart policy change the SABRE-only baseline quality. The systematic primary JIT advantage observed with the single-candidate controls is removed by a best-of-ten outer selection. However, it incurs the cost of all ten compilations. Future work should therefore treat incumbent quality and incumbent-construction cost jointly, measuring whether additional starts actually reduce end-to-end exact proof time rather than considering route quality alone. The opt3 results motivate a complementary controlled-ablation study in which placement, routing, cancellation and resynthesis stages are enabled separately, so that the source of its lower final J can be distinguished from the broader effect of end-to-end optimisation [2, 45].

Broader benchmark and calibration sampling. The 70-circuit dataset is structured around seven recurring circuit families and a single frozen hardware snapshot. A broader external-validity study would require additional independently generated circuit families, multiple star-compatible calibration snapshots, and, where available, data from other devices. In order to distinguish algorithm-dependent runtime behaviour from variation introduced by the system used for a single benchmark campaign, repeated measurements of classical compilation time would be required. This would make family-level uncertainty more informative and test whether the observed seed sensitivity, objective decomposition and relative baseline behaviour persist beyond the present benchmark design.

Anytime exact/heuristic operation. For instances beyond practical proof capacity, a validated incumbent together with an admissible lower bound could support an anytime interface reporting a feasible route and, when available, a certified optimality gap. Timeout must remain semantically distinct from `queue_exhausted`; approximate beam or width-limited pruning may be useful in a separate heuristic mode but must not inherit the exact certificate [33].

14.5 Closing perspective

Qubit routing bridges the logical structure of a quantum program and the calibrated constraints of a physical processor. Even on the studied five-qubit star, route quality depends jointly on connectivity, mapping evolution, heterogeneous primitive costs, and resource scheduling. For this reason, the hardware-aware optimisation problem can't be solved by inserted SWAP count alone.

The main outcome of this thesis is a formal exact-routing framework where these assumptions remain explicit. Max-Plus dominance provides a continuation-safe hardware-aware reduction; JIT progress determinacy enables scheduling freedom without pending-state explosion; and permutation-route embedding firmly links the JIT optimum to established minimum-SWAP models. The final eight-method benchmark contextualises these findings: dedicated baselines, restart policies, and end-to-end optimisation yield materially different outcomes even under identical physical-cost surrogates. Therefore, exactness, route quality, and compiler performance must remain distinct claims, strictly qualified by their supporting objectives, trace spaces, and experimental conditions.

Appendix A

Software Architecture and Reproducibility

This appendix maps the algorithmic presentation in Chapters 5 to 9 to the frozen source, records implementation details intentionally omitted from the main body, and documents the reproducibility package of the final benchmark. It covers the Python–native boundary, native storage and indexing mechanisms, benchmark execution and data-only reconstruction. It does not repeat the routing equations or the correctness arguments of Chapter 11.

A.1 Repository and Frozen Thesis Release

The software and benchmark package used for this thesis are available at `https://gitlab.linksfoundation.com/aca/quantum-computing/master_theses/ciardo/j-exact-router`. The final reproducible thesis release is frozen at tag `v1.1-thesis`, corresponding to Git commit `122ccc107eef9d29c38c0d504282c523edeef973`. The release contains the J-exact implementation, the reconstructed permutation-based comparison router, the official 70-circuit five-qubit OpenQASM corpus, benchmark runners and analysis code, frozen hardware reports, tests, and the complete archived eight-method campaign used in Chapters 12 and 13.

The official benchmark circuits are stored directly under `QASM/`. Their identities, families, variants, scale parameters, input-CZ bands and SHA-256 hashes are recorded in `QASM/benchmark_manifest.csv`. Test-only circuits are stored separately and are not members of the experimental corpus. All final benchmark routes use the frozen star configuration `Hardware_Configurations/9da78e.txt`, whose hardware signature is `9da78e`.

The tag `v1.1-thesis` identifies the packaged publication artifact and fixes the exact public repository state described in this appendix. It should not be interpreted as a claim that every archived raw benchmark run was originally executed from that Git commit. Historical source provenance is recorded separately in `results/audits/source_provenance.json`. Archived execution metadata are preserved in their original form rather than being rewritten to use the publication identifier.

Byte-level file inventories are maintained separately for the complete project and for the archived results. These are recorded in `MANIFEST.json`, `SHA256SUMS`, `results/manifest.json`, and `results/SHA256SUMS`.

The package requires Python 3.10 or later. The final benchmark was run serially on the same Windows 11 PC using two local software environments. JIT-exact, forced closure and the permutation-based router used CPython 3.11.15; Qiskit opt3, Qiskit SABRE-only, tket and both MQT QMAP modes used CPython 3.12.13. The recorded machine exposed 20 logical processors, while numerical threading and Qiskit process parallelism were restricted to one where applicable.

The corresponding dependency definitions are retained in `benchmark/environments/environment-internal.yml`, `environment-external.yml`, `requirements-internal.txt`, and `requirements-external.txt`. A third compiler-independent environment, defined by `requirements-analysis.txt`, is sufficient to reproduce the archived statistical analysis, tables and figures without installing the routing compilers.

The default `config.cfg` selects the frozen local hardware report. In local mode, `local.py` validates a stored report; in live mode, `lagrange.py` queries the IQM service and `star_aware_router/hardware_snapshot.py` serialises or verifies a lossless snapshot. Live acquisition failures do not silently fall back to local data. The final thesis benchmark removes this source of temporal variation by using the frozen local snapshot `9da78e`.

A.2 Software Architecture

The release has a Python control plane and a native search plane. Python owns CLI validation, QASM and DAG construction, hardware selection, SABRE incumbent construction, authoritative replay, objective recomputation, and serialization. The compiled kernel owns the complete label-setting loop: FC or JIT scheduling, successors, lower bounds, exact duplicates, guarded Max-Plus dominance, arena and queue management, deadlines, and parent-trace recovery. Python sends one compact payload and receives one result; there is no per-label Python callback.

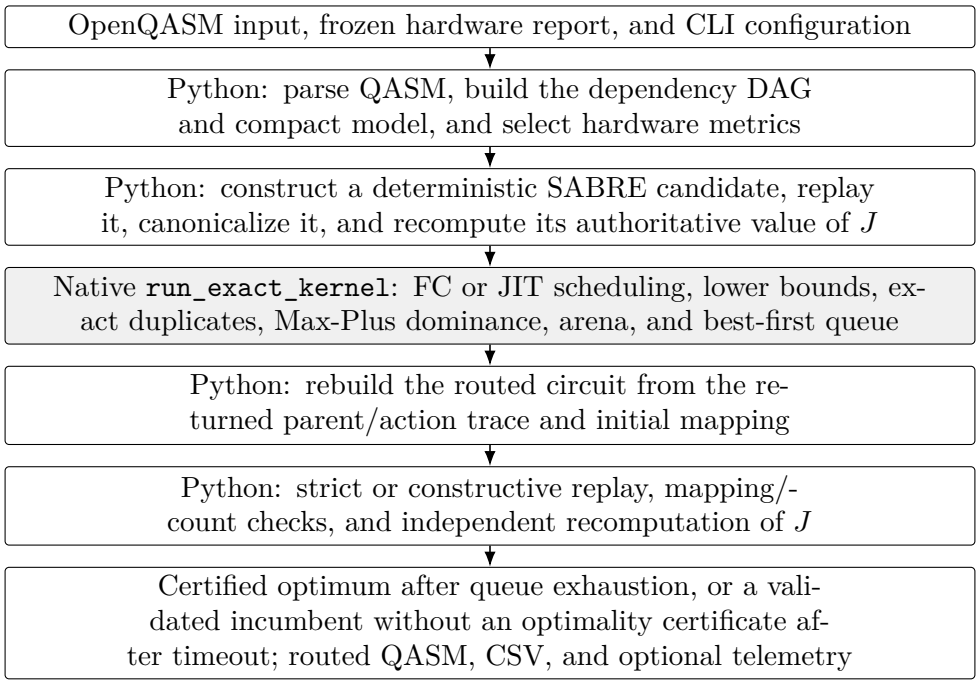


Figure A.1: End-to-end J-exact execution pipeline. The shaded box is the native search boundary; all surrounding stages are controlled and validated in Python.

Figure A.1 distinguishes a valid incumbent from a certified optimum. Only native queue exhaustion proves optimality; timeout may return a validated incumbent without that certificate, as specified in Sections 11.4 and 11.6. In both cases Python replays the route, checks mappings and actions, and recomputes both components of J.

`qasm_io.py` and `dag.py` load the Qiskit circuit, construct the restricted dependency graph, and separate the terminal measurements. The exact router rejects unsupported operations and prepares the per-line progress and dependency information used during the search. `metrics.py` processes the same hardware snapshot for both incumbent construction and exact search.

The SABRE adapter uses a deterministic seed and evaluates all 120 initial mappings. Each returned candidate is interpreted, canonicalised, replayed under strict validation, and rescored with the common objective. Candidates that do not pass validation are rejected. Only this authoritative value becomes the native incumbent, never the heuristic's self-score. This realizes the distinction in Chapter 7.

Forced closure and JIT-exact share the same native exact-search infrastructure. Forced closure applies its prescribed non-routing materialisation eagerly. JIT, instead, uses compiled prerequisite and terminal plans. The archived final configurations use the native kernel, the `indirect-slot` exact map, the linear frontier

index, the admissible `cheap` bound and one exact worker. Their intended algorithmic difference in the scheduling ablation is therefore the selected non-routing semantics.

The permutation-based router under `iqm_spark_paper_exact` [1] is an independent comparison implementation. Its pybind11 core evaluates feasible orders and center sequences for minimum SWAP count, without BRIDGE; J is post-hoc. Its queue-exhaustion claim and outputs are therefore separate from J-exact. Its queue-exhaustion certificate concerns minimum inserted SWAP count rather than minimum J.

The eight-method experimental campaign is orchestrated outside the routing implementations. `benchmark/runners/run.py` coordinates serial execution, `benchmark/runners/common.py` handles the three internal methods, and `benchmark/runners/external.py` contains the Qiskit, tket and MQT adapters together with their timing instrumentation. The analysis layer under `benchmark/analysis/` reads archived outputs and performs independent evaluation, deterministic Qiskit policy reconstruction, statistical analysis and artifact rendering without modifying the routing implementations.

A.3 Mapping Thesis Concepts to Source Modules

Table A.1 is the principal navigation map from the terminology used in the thesis to the inspected implementation. A source file may appear in more than one row because the final release deliberately co-locates tightly coupled native mechanisms in a single translation unit. The table names the active release path; reference and test-only Python implementations retained for differential validation are not alternate production backends.

Table A.1: Mapping from thesis concepts to source modules.

Concept	Primary source path	Implementation responsibility
CLI and run orchestration	`star_aware_router_ex/__main__.py`	Parses the release options, selects the backend, invokes J-exact, handles certified and timeout results, and dispatches QASM and CSV output.

Continued on next page

Table A.1 continued

Concept	Primary source path	Implementation responsibility
OpenQASM and dependency model	`qasm_io.py` `dag.py`	Loads QASM 2 input, constructs the dependency DAG and line order, separates terminal measurements, and writes the routed QASM.

Configuration and hardware	`config.cfg`
	`star_aware_router/config.py`
	`star_aware_router/backend.py`
	`star_aware_router/hardware_snapshot.py`
		Handles local and live acquisition, validates normalized metrics, and binds each live snapshot to its hardware signature.

Physical metrics and objective	`metrics.py`
	`circuit_error.py`
	`execution_time.py`
		Normalizes operation metrics and independently evaluates circuit error, ASAP duration, and the reported value of J.

SABRE incumbent	`star_aware_router/routers/sabre_upper_bound_adapter.py`	Builds the deterministic all-initial-mapping candidate and subjects it to restricted-action replay, canonicalization, strict replay, and rescoring.
Authoritative route validation	`star_aware_router/routers/restricted_route_validation.py`	Implements flexible replay of raw restricted actions, canonical route construction, strict general replay, and fast constructive replay of a trusted internal JIT trace.
Python exact coordinator	`star_aware_router/routers/dp_hardware_aware_exact.py`	Builds compact eager or JIT models and the native payload, validates the incumbent, calls the kernel once, rebuilds the trace, and validates the final result.

Continued on next page

Table A.1 continued

Concept	Primary source path	Implementation responsibility
ExactKernel and binding	`star_aware_router/_pareto_frontier_cpp.cpp`	Implements the direct CPython extension entry point `run_exact_kernel`, payload parsing, complete native search loop, deadline status, statistics, and trace return.
Frontier, duplicates, and dominance	`star_aware_router/_pareto_frontier_cpp.cpp`	Stores frontier slot identifiers, uses an indirect open-addressed exact map for binary64 state identity, and applies classical plus guarded Max-Plus dominance.
Label arena, queue, and ancestry	`star_aware_router/_pareto_frontier_cpp.cpp`	Manages chunked hot/cold label storage, stable append-only slot IDs, generation-tagged best-first entries, active-label filtering, and compact parent/action records.
FC and JIT scheduling	`star_aware_router/_pareto_frontier_cpp.cpp` `star_aware_router/routers/dp_hardware_aware_exact.py`	Encodes forced-closure and JIT prerequisite and terminal plans, with equivalent Python reference logic used for validation tests.
Bounds, deadlines, and certificates	`star_aware_router/_pareto_frontier_cpp.cpp`	Applies the off and cheap lower-bound modes, incumbent cutoffs, periodic deadline checks, and the resulting queue-exhausted or timeout certificate state.
Output and telemetry	`csv_logger.py` `star_aware_router/telemetry_contract.py` `star_aware_router_ex/__main__.py`	Writes routed-circuit metadata and statistics, defines telemetry records, and keeps timeout incumbents separate from certified optima.

Continued on next page

Table A.1 continued

Concept	Primary source path	Implementation responsibility

Permutation-based router	`iqm_spark_paper_exact/cli.py`
	`iqm_spark_paper_exact/paper_algorithm.py`
	`iqm_spark_paper_exact/_exact_core.cpp`
		Enumerates feasible orders in Python, evaluates minimum-SWAP center sequences in the pybind11 core, reconstructs the circuit, and emits its own QASM, JSON, and CSV records.

Benchmark execution	`benchmark/runners/run.py`
	`benchmark/runners/common.py`
	`benchmark/runners/external.py`
		Runs or resumes the three internal and five external methods in their recorded environments, validates completed artifacts before reuse, and writes new campaigns outside the frozen `results/` archive.

Statistical analysis and selection	`benchmark/analysis/build_analysis.py`
	`benchmark/analysis/selection_policy.py`
	`benchmark/analysis/evaluator/`
		scrivilo in una box di testo in latex seguendo esattamente e attentamente le regole di wikipedia rimanendo nell'ambito accademico, senza inserire in tcolorbox

Release and artifact validation	`tests/test_j_exact_release.py`
	`tests/test_hardware_and_permutation.py`
	`tests/test_benchmark_artifacts.py`
	`benchmark/analysis/reproduce.py`
	`benchmark/analysis/verify_package.py`
		Tests routing and hardware invariants, verifies all eight stored campaigns, reconstructs Qiskit selections, reproduces the statistical analysis, and checks package inventories and SHA-256 hashes.

Package integrity	`MANIFEST.json`
	`SHA256SUMS`
	`results/manifest.json`
	`results/SHA256SUMS`
	`.gitattributes`
		Defines the public-file inventories, records exact byte hashes and prevents Git line-ending conversion from changing files covered by the manifests.

The mathematical state of Chapters 8 and 9 is compiled once, rather than serialized as a Python object per successor. The guarded comparison of Equation (9.6) is co-located with the exact-key logic it must match.

A.4 Native Implementation Details

The following mechanisms are implementation details rather than independent algorithmic contributions. They are retained here because they realise the exact-state, numerical-safety and reconstruction invariants described abstractly in the main body.

The release builds two C++17 extensions. J-exact's `star_aware_router/_pareto_frontier_cpp.cpp` uses the direct CPython C API; the permutation module `iqm_spark_paper_exact/_exact_core.cpp` uses pybind11. J-exact also uses explicit floating-point compiler options so that fast-math optimisations do not alter the numerical assumptions used in Chapters 9 and 10.

For J-exact, `star_aware_router/routers/dp_hardware_aware_exact.py` prepares the topology, compact node and progress data, dependency and JIT information, lower-bound data, physical metrics, incumbent, and mode codes. A single `run_exact_kernel` call then transfers the search loop to C++. The returned result contains the termination status, search counters, terminal mapping, and the parent/action trace of the selected route.

Table A.2: Correspondence between logical search objects and native storage.

Logical object	**Native representation and role**
Progress and mapping state	Compact progress fields and a permutation rank in a stored label; checked wider representations are used when compact narrowing is not safe.
Exact dynamic state	Binary64 error bits plus center time and leaf deltas, recovered through the label's stable arena slot.
Frontier	A `NativeFrontier` holding active arena-slot identifiers and an indirect exact-map index.
Best-first open set	Deterministically ordered queue entries containing bound, routing counts, progress, mapping, slot, and generation. Stale or inactive entries are discarded on pop.
Label lifetime	Chunked append-only arena storage separates hot search fields from cold trace data while preserving stable indices.
Parent relation	A compact parent slot, node identifier, and packed action record; parent slots always refer to older committed labels.
Terminal result	Queue exhaustion yields an exact certificate; deadline expiry yields the best validated incumbent and an explicit non-certified status.

Table A.3: Native label-storage layouts implemented by the exact kernel. The final thesis benchmark uses the `cheap`-bound layout in the last row; the smaller layouts are eligible only for `bound-mode=off` runs.

Layout	Per committed slot	Principal storage
Compact progress	76 B	64 B hot record plus 12 B reconstruction trace
Retained progress	84 B	72 B hot record plus 12 B reconstruction trace
Retained deltas	100 B	88 B hot record plus 12 B reconstruction trace
Final `cheap`-bound layout	116 B	88 B retained hot record, 16 B lower-bound accumulator sidecar and 12 B reconstruction trace

Labels are allocated in append-only chunks of 65 536 logical slots. Stable 32-bit slot identifiers therefore remain valid for reconstruction even after a label becomes inactive. The final `cheap`-bound configuration retains 32-bit progress storage and 64-bit centre-relative timing coordinates; the checked narrower layouts are selected only when their eligibility conditions hold in bound-off runs.

The indirect exact map stores stable arena-slot identifiers rather than copying complete dynamic keys. Exact equality therefore recovers the authoritative binary64 and scheduling fields from the referenced committed label. Guarded Max-Plus dominance scans only labels with the same discrete frontier key, constructs outward binary64 intervals with `nextafter`, and fails closed whenever a safe ordering cannot be established.

Candidate publication is transactional. Structural checks, exact-duplicate detection, incumbent pruning, dominance, deadline tests and narrowing conversions complete before persistent search structures are modified. Only then are dominated labels deactivated and the new arena, frontier, exact-map and queue entries made visible. Generation-tagged queue records allow stale entries to be discarded lazily when popped.

On return, Python reconstructs physical operations. Eager output receives the general strict dependency-valid replay; an internal JIT trace receives constructive replay of prerequisite events, mappings, ticks, error, routing counts, terminal flush, and measurements. Independent final evaluation then enforces the same observable routing and objective invariants, preserving the certificate interpretation of Section 11.8.

For the comparison router, Python batches feasible topological orders to the pybind11 core, which evaluates center sequences under the deadline. Python then reconstructs the minimum-SWAP route and evaluates J post-hoc; its separate schema prevents that certificate from being confused with J-exact.

A.5 Build and Reproduction Paths

A.5.1 Core router installation

A C++17 compiler and the Python development headers are required to build the two native extensions. From the repository root, the core package can be installed in editable mode with:

```
python -m pip install -e
```

A minimal invocation resolves the circuit from the official `QASM/` directory and uses the default command-line settings:

```
j-exact b01_qaoa_maxcut_cycle5_p2_cz020.qasm
```

The recorded JIT configuration can instead be selected explicitly as:

```
j-exact b01_qaoa_maxcut_cycle5_p2_cz020.qasm --hardware-source local --hardware-
    file Hardware_Configurations/9da78e.txt --upper-bound-strategy sabre-upper-
    bound --exact-kernel-backend native --native-exact-map-backend indirect-slot
     --native-frontier-index linear --nonrouting-schedule-mode jit-exact --bound
    -mode cheap --exact-workers 1 --time-limit-seconds 900
```

Replacing `jit-exact` with `eager` selects the forced-closure ablation. The permutation-based comparison router is exposed separately through the `iqm-spark-paper-exact` console entry point.

A.5.2 Data-only reconstruction of the final analysis

The complete archived analysis can be reproduced without installing a quantum compiler, building either native routing extension, rerunning a router or accessing a QPU:

```
python -m pip install -r benchmark/environments/requirements-analysis.txt
python benchmark/analysis/reproduce.py
```

The reproduction pipeline independently audits the stored routed circuits, reconstructs the common metrics and Qiskit selection policies, rebuilds the derived CSV and JSON analysis, and regenerates the LaTeX tables and final figures in a temporary directory. The regenerated artifacts are required to be byte-identical to the archived copies, and the command does not invoke any router or compiler.

To retain the regenerated files instead of using the temporary verification directory:

```
python benchmark/analysis/reproduce.py --output-dir analysis_rebuilt
```

A.5.3 Executing or resuming a new benchmark

The final benchmark distinguishes the internal and external compiler environments. The provided environment definitions can be recreated and the package installed in each environment as follows:

```
conda env create -f benchmark/environments/environment-internal.yml
conda env create -f benchmark/environments/environment-external.yml

conda run -n quantum-vlsi python -m pip install --no-build-isolation --no-deps -
    e .
conda run -n quantum-vlsi-benchmark python -m pip install --no-build-isolation
    --no-deps -e .
```

The three internal methods and five external methods are run serially:

```
conda run -n quantum-vlsi python benchmark/runners/run.py --methods internal --
    output-dir benchmark_runs/local --resume
conda run -n quantum-vlsi-benchmark python benchmark/runners/run.py --methods
    external --output-dir benchmark_runs/local --resume
```

The `-resume` option preserves complete validated circuit artifacts and reruns incomplete units; for Qiskit a complete ten-seed candidate set constitutes one resumable unit. Fresh executions are written outside the archived `results/` directory.

Direct CLI executions can additionally emit routed QASM, statistics records and optional JSONL telemetry in their configured runtime locations. Exact naming conventions and the complete set of executable options are documented in `J_EXACT_USAGE.md` and `iqm_spark_paper_exact/PERMUTATION_USAGE.md`. The command-line parser provides the definitive specification of these runtime options.

A.6 Frozen Benchmark Artifacts and Validation

The frozen `results/` directory contains the complete benchmark campaign for all eight methods and 70 circuits. Its eight raw subdirectories store the outputs for JIT-exact, forced closure, the permutation-based router, Qiskit opt3, Qiskit SABRE-only, tket, MQT QMAP heuristic, and MQT QMAP exact. The archive contains 560 selected method–circuit records, all 1,400 Qiskit seed candidates, and 420 selections corresponding to the three retained Qiskit policies. JIT and forced closure each contain 70 exact certificates, while the permutation archive contains 49 minimum-SWAP certificates and 21 valid timeout incumbents.

Raw compiler and router outputs are separated from derived analysis products. `results/derived/` contains the common 560-row analysis table, the 1,400-candidate Qiskit table, policy-selection records, paired comparisons against JIT, primary hypothesis tests, family-block and mirror sensitivity analyses, and unified compilation-time summaries. `results/tables/` and `results/figures/` contain the presentation artifacts consumed by the thesis, while `results/audits/` records semantic, numerical, provenance and reproduction checks.

The independent route audit reconstructs topology, five-qubit mappings, primitive-operation counts, full 32×32 unitaries and the common hardware-aware objective directly from the stored routed circuits. Certificate status is not used as a substitute for semantic validity. Separate audit records verify tket intermediate transformations, the MQT CZ-to-CX adjacency-token round trip and exact reconstruction of every Qiskit selection policy.

The data-only reproduction requires byte-identical reconstruction of every derived file, LaTeX table and final figure. The check of input and raw-result hashes is done before and after the analysis. In this way is possible to verify that the reproduction process has not modified the archived campaign.

Git text conversion is disabled globally through the repository-level `.gitattributes`:

```
# Preserve exact bytes recorded by MANIFEST.json and SHA256SUMS.
* -text
```

This rule prevents automatic LF/CRLF conversion from changing files covered by the byte-level manifests, including on Windows systems with `core.autocrlf=true`. `benchmark/analysis/verify_package.py` verifies this global rule, rejects obsolete path-specific patterns, and checks both project-level and result-level inventories and SHA-256 hashes.

36 passing pytest cases, with no failures, errors or skipped tests are recorded by the archived publication audit. The suite combines algorithmic and hardware checks on small fixtures with checks of the official corpus, the completeness of all eight stored campaigns, certificate and timeout counts, exact reconstruction of Qiskit selections, native-extension imports and protection of the frozen archive.

Despite its historical filename, `tools/validate_representative.py` no longer validates only a small representative subset. In the publication release it delegates directly to the complete data-only reconstruction pipeline and therefore validates the entire archived campaign without invoking a router.

A clean publication package can be checked with:

```
python -m pip install -r benchmark/environments/requirements-test.txt
python -m pip install --no-build-isolation --no-deps -e .
```

```
python -m pytest -q
python tools/validate_representative.py
python benchmark/analysis/verify_package.py
```

These checks provide implementation and artifact evidence; the mathematical claim that queue exhaustion certifies the optimum of the selected exact model remains the subject of Chapter 11.

A.7 External Benchmark Configuration Details

A.7.1 tket configuration

The tested configuration uses pytket 2.18.1 with `NoiseAwarePlacement` followed by `RoutingPass` on the five-qubit star [41]. Placement receives the calibrated single-qubit and edge errors with `maximum_matches=1000`, a 1000 ms timeout, `maximum_pattern_gates=300`, and `maximum_pattern_depth=300`. The raw routed output, SWAP-lowered representation and final native circuit are retained separately.

A.7.2 Qiskit configurations

Both Qiskit baselines use Qiskit 2.1.2 and seeds $0, \ldots, 9$. Opt3 invokes the end-to-end level-three transpilation pipeline on the calibrated five-qubit target. SABRE-only instead uses `SabreLayout` with four iterations, one layout trial and one SWAP trial; the subsequent optimisation-level-zero native translation uses the selected physical layout with routing disabled. All ten raw candidates are retained for every circuit and both Qiskit methods.

A.7.3 MQT QMAP adapter

The benchmark uses MQT QMAP 3.9.0 and MQT Core 3.9.2. Heuristic mode uses dynamic initial placement and the `gate_count_max_distance` strategy; exact mode uses QMAP's minimum-SWAP formulation. Pre- and post-mapping CNOT-specific optimisations are disabled in both configurations.

Because this QMAP version expects CNOT interactions, each source CZ is temporarily represented by one CX adjacency token carrying the corresponding CZ duration and error. The token representation is used only to express the same adjacency constraint and is not treated as an executable replacement of the source circuit. After mapping, every routed token is restored one-to-one to CZ before SWAP lowering and common evaluation. Stored interaction counts, layouts and full-unitary checks independently audit this conversion.

Appendix B

Frozen Hardware Configuration

This appendix reports the complete frozen hardware configuration used throughout the routing and benchmarking experiments. The configuration, identified by hardware signature `9da78e`, was generated on 9 June 2026 and describes a five-qubit star topology with q_2 as the centre and q_0, q_1, q_3, and q_4 as leaves.

Coherence times are reported in microseconds and gate durations in nanoseconds. The SWAP quantities are not native backend calibration parameters: they are derived from the corresponding CZ and single-qubit primitive metrics.

Table B.1: Complete frozen hardware configuration with signature `9da78e` used in the routing and benchmarking experiments.

General configuration			
Parameter	Value		
Generated at	`2026-06-09 T 20:34:56`		
Hardware signature	`9da78e`		
Number of qubits	5		
Centre qubit	q_2		
Leaf qubits	q_0, q_1, q_3, q_4		
Native single-qubit gate	r		
Qubit coherence metrics			
Qubit	T_1 [μs]	T_2 [μs]	
q_0	50.66	18.09	
q_1	43.85	10.01	
q_2	26.34	11.54	
q_3	49.92	3.339	
q_4	45.67	12.49	
Single-qubit gate metrics			
Gate	Qubit	Error probability	Duration [ns]
r	q_0	4.051×10^{-4}	20
r	q_1	7.691×10^{-4}	20
r	q_2	4.692×10^{-4}	20
r	q_3	6.739×10^{-4}	20
r	q_4	4.009×10^{-4}	20
CZ gate metrics			
Gate	Edge	Error probability	Duration [ns]
CZ	q_2–q_0	4.217×10^{-3}	80
CZ	q_2–q_1	4.089×10^{-3}	68
CZ	q_2–q_3	5.792×10^{-3}	80
CZ	q_2–q_4	4.004×10^{-3}	64
Derived SWAP metrics			
Gate	Edge	Error probability	Duration [ns]
SWAP	q_2–q_0	1.519×10^{-2}	360
SWAP	q_2–q_1	1.588×10^{-2}	324
SWAP	q_2–q_3	2.064×10^{-2}	360
SWAP	q_2–q_4	1.454×10^{-2}	312

B.1 Interpretation of the effective coherence scale

A physical interpretation of $T_{\text{coh,eff}}$ follows from the short-time process fidelity of a qubit subject to amplitude relaxation and dephasing. Writing the Bloch contraction factors as $e^{-t/T_2}, e^{-t/T_2}, e^{-t/T_1}$, its identity process fidelity is

$$F_{\text{proc},q}(t) = \frac{1 + 2e^{-t/T_{2,q}} + e^{-t/T_{1,q}}}{4} = 1 - t\left(\frac{1}{4T_{1,q}} + \frac{1}{2T_{2,q}}\right) + O(t^2) \tag{B.1}$$

Exponentiating the first-order loss gives the model factor $e^{-t/T_{\text{coh},q}}$. Supposing that all five qubits retain quantum information until the final measurement, multiplying these factors over the full makespan yields

$$\frac{1}{T_{\text{coh,eff}}} = \sum_{q=0}^{4}\left(\frac{1}{4T_{1,q}} + \frac{1}{2T_{2,q}}\right), \qquad P_{\text{model}} = P_{\text{gates}} \prod_{q=0}^{4} e^{-T_{\text{exec}}/T_{\text{coh},q}} \tag{B.2}$$

The logarithmic identity in Equation (12.1) is exact for this constructed model. The exponential continuation of the first-order channel loss is an approximation, especially for durations comparable to or longer than coherence times. It is not the exact finite-time process fidelity of the complete noisy circuit.

Two double-counting issues are distinct. Algebraically, P_{gates}, G, C, J and P_{model} describe the same score and are never recombined as independent quality indicators. Physically, calibrated gate errors contain active-time decoherence, which the makespan penalty charges again. The calibration does not separate control error from decoherence, so this overlap cannot be removed from the recorded results. P_{model} denotes model survival, not measured fidelity or the probability of the correct algorithmic answer.

Appendix C

Supplementary Correctness Proofs and Counterexamples

C.1 Supporting completeness proofs

C.1.1 Initial-state completeness

Lemma 2 (**initial-state generation completeness**).
For each selected scheduling mode

$$M \in \{E, J\}$$

where E denotes the forced-closure mode and J the JIT-exact mode, every permitted initial logical-to-physical mapping is generated and submitted to the common exact-search commit and reduction machinery.

Proof
At width five, every logical-to-physical mapping is one of the $5! = 120$ permutations of the logical qubits onto the physical star. The native kernel iterates the complete ranked permutation table independently of the mapping used by the validated incumbent. Thus neither the SABRE initial placement nor any other upper-bound placement can prevent an initial mapping from being generated by the exact search.

- **Forced-closure case**
  For every mapping m, the kernel first constructs the zero-history state

  $$u_0(m): \qquad G = 0, \qquad \mathbf{t} = \mathbf{0}$$

  with initial logical progress and logical-to-physical mapping m. It then applies the deterministic forced-closure operator $C_E(u_0(m))$ materialising the operations prescribed by the forced-closure semantics until the closure fixed point

is reached. The resulting closed initial candidate is then submitted to the common transactional commit and reduction machinery.

- **JIT-exact case**
  For every mapping m, the JIT kernel constructs

  $$u_0^J(m): \qquad p_{2Q} = 0, \qquad G = 0, \qquad \mathbf{t} = \mathbf{0},$$

  with no routing-relevant two-qubit event consumed and mapping m.

  No initial forced closure is applied. In particular, ready non-routing operations that are not yet required by a selected two-qubit event remain pending. When a first event

  $$g \in R_{2Q}\Big(u_0^J(m)\Big)$$

  is selected, its missing non-routing prerequisites are handled later by the JIT materialisation operator

  $$M\Big(u_0^J(m), g\Big)$$

  This does not alter the preceding fact that the root associated with every permitted mapping m has already been generated and submitted to the exact search machinery.

Hence, in either scheduling mode, all 120 permitted initial mappings are generated and presented to the same exact-search commit and reduction procedure. The present lemma makes no claim about whether every submitted candidate remains active after duplicate detection, dominance reduction, or other exact pruning operations; the safety of those reductions is established separately by the corresponding later results.

Therefore the initialisation procedure is complete at the generation stage with respect to the selected scheduling semantics. □

C.1.2 Forced-closure successor completeness

Lemma 3 (**forced-closure successor completeness**)**.**
For every active non-terminal forced-closure label u, native successor generation returns every legal next state admitted by the forced-closure search space $\mathcal{R}_E$.

Proof
By construction, every active label in the forced-closure mode is a closure fixed point. All currently executable operations that introduce no routing branch have therefore already been materialised according to the forced-closure semantics. Let $g_E(u)$ denote the single routing-relevant two-qubit event exposed by the deterministic forced-closure selector at state u. Once $g_E(u)$ is fixed, the legal physical action set depends only on the current mapping.

If the physical operands of $g_E(u)$ form a native centre–leaf edge, the interaction is directly executable and the forced-closure action set contains exactly one element:

$$|\mathcal{A}_E(u)| = 1$$

The unique successor executes the direct two-qubit interaction and then applies deterministic post-action forced closure.
If both operands occupy leaves, the restricted star action model admits exactly three alternatives:

$$\mathcal{A}_E(u) = \{S_1, S_2, B\}$$

where S_1 and S_2 swap either logical operand with the physical centre before executing the interaction, and B denotes the fixed mapping-preserving BRIDGE construction. Hence

$$|\mathcal{A}_E(u)| = 3$$

For every $a \in \mathcal{A}_E(u)$ the native kernel constructs the corresponding physical transition, updates the mapping when required, advances logical progress, updates the integer resource-availability vector and accumulated gate cost G, and finally applies deterministic forced closure to the resulting state. Thus each committed successor has the form

$$v = C_E(T_E(u, g_E(u), a))$$

where T_E denotes execution of the selected routing action and C_E denotes forced closure.

Since $g_E(u)$ is uniquely determined by the forced-closure semantics and the kernel enumerates every action in $\mathcal{A}_E(u)$, no legal forced-closure successor is omitted. Conversely, no successor outside the restricted action model is generated. Therefore native successor generation is complete with respect to $\mathcal{R}_E$. □

Remark
The lemma is intentionally model-relative. In particular, proactive SWAPs, alternative BRIDGE decompositions, and arbitrary scheduling choices among non-routing operations are not missing implementation branches; they are excluded by the definition of the forced-closure search model $\mathcal{R}_E$. Completeness with respect to a larger physical or scheduling action space would require enlarging the successor relation and establishing a corresponding new completeness result.

Implementation-level differential tests provide an independent check that the native successor generator agrees with the reference implementation on reachable forced-closure states, including primitive operations, mapping updates, progress transitions, scheduling coordinates and post-action closure.

C.1.3 Forced-closure contraction/soundness

Proposition 3 (forced-closure contraction within $\mathcal{R}_E$)
Let u be a forced-closure search state. Replacing a maximal sequence of states having a single successor under the deterministic forced-closure semantics by the corresponding contracted closure transition preserves the set of complete routes represented in $\mathcal{R}_E$ and preserves their objective values.

Equivalently, explicit execution of the deterministic forced transitions and their contraction into a single closure operation represent the same paths of the forced-closure search model.

Proof
Consider a state encountered during forced closure for which the model prescribes exactly one next transition. By definition of the forced-closure semantics, both the identity of that transition and the deterministic order in which such forced events are processed are part of the transition system defining $\mathcal{R}_E$.

Executing the forced transition explicitly produces a unique successor and updates the corresponding search quantities—including projected progress, logical-to-physical mapping where applicable, integer resource-availability times, and accumulated gate cost G—according to the same transition rules used by the uncontracted search.

If the successor again admits exactly one transition under the same semantics, the argument can be repeated. Thus a maximal chain

$$u_0 \to u_1 \to \cdots \to u_k$$

of deterministic forced transitions contains no branching decision within $\mathcal{R}_E$. Replacing this chain by the contracted transition

$$u_0 \Longrightarrow u_k$$

therefore removes only intermediate representations of states, not any route choice admitted by the forced-closure model.

Because the contracted closure applies exactly the same sequence of physical operations and the same deterministic scheduling updates as the corresponding explicit transitions, the resulting mapping, timing state, accumulated gate cost and therefore the objective contribution are unchanged.

Hence deterministic closure contraction preserves both the complete routes represented by $\mathcal{R}_E$ and their objective values. □

Remark
This proposition is deliberately restricted to the forced-closure search space $\mathcal{R}_E$. It does not claim that deterministic closure preserves every dependency-valid schedule of the original circuit.

In particular, if several logically independent non-routing events could be scheduled in different dependency-valid orders and those orders lead to different physical timing or gate-cost contributions, the fixed closure rule selects only the deterministic ordering prescribed by the forced-closure semantics. Such alternative schedules may therefore lie outside $\mathcal{R}_E$.

The proposition states only that, once this deterministic semantics has been fixed, replacing its explicit one-successor transition chains by a contracted closure operation does not further restrict the resulting search space.

C.1.4 JIT prerequisite-materialisation soundness

Consider a committed pre-terminal-flush JIT label u with projected two-qubit progress p, and let $g \in R_{2Q}(u)$ be an eligible two-qubit event selected as the next routing-relevant event.
Using the notation of Section 11.2.1, define the missing non-routing prerequisites of g as

$$P(g, p) = \mathrm{Anc}_{\mathrm{NR}}(g) \setminus N_{\mathrm{done}}(p) \tag{C.1}$$

These are precisely the non-routing predecessors of g that are required by the canonical JIT model but have not yet been materialised. The deterministic prerequisite-materialisation operator introduced earlier is therefore $u_g = M(u, g)$ where M schedules the events in $P(g, p)$ in the canonical dependency-valid order, updates the physical-resource availabilities and accumulated gate cost G, but does not yet consume g.
Lemma 4 (JIT prerequisite-materialisation soundness).
Let u be a committed pre-terminal-flush JIT label with projected progress p_u. Let $g \in R_{2Q}(u)$ and define the canonical JIT pre-action state $u_g = M(u, g)$. By definition, M materialises the events in

$$P(g, p_u) = \mathrm{Anc}_{\mathrm{NR}}(g) \setminus N_{\mathrm{done}}(p_u)$$

in the deterministic dependency-valid order prescribed by the JIT semantics. Then prerequisite materialisation satisfies

$$p_{u_g} = p_u, \qquad m_{u_g} = m_u$$

consumes no two-qubit event, and materialises no pending non-routing event outside $P(g, p_u)$. Consequently,

$$R_{2Q}(u_g) = R_{2Q}(u) \tag{C.2}$$

Thus prerequisite materialisation prepares the selected event g without implicitly making an additional two-qubit-order or routing decision.

Proof
The definition of M fixes which non-routing events are materialised and their internal canonical order. The soundness properties relevant to the subsequent exact search are established as follows.

1. **Necessity of the materialised prerequisites**
   For every $x \in P(g, p_u)$, the definition of P gives

   $$x \in \mathrm{Anc}_{\mathrm{NR}}(g)$$

   and

   $$x \notin N_{\mathrm{done}}(p_u)$$

   Hence x is an unmaterialised non-routing predecessor of g. Every legal JIT continuation that selects g as its next two-qubit event must therefore materialise x before executing g.

   The internal materialisation order of these prerequisites is the deterministic canonical order prescribed by the JIT semantics. It is not an additional two-qubit ordering decision: it is part of the definition of the JIT search space $\mathcal{R}_J$.

2. **Dependency-valid canonical order**
   The operator M emits the elements of $P(g, p_u)$ in the canonical order defined by the JIT materialisation procedure. This order is consistent with the predecessor relations of the original dependency DAG. Hence, whenever $x, y \in P(g, p_u)$ and the circuit requires $x \prec y$, the canonical materialisation order also places x before y.

   The physical operations are scheduled using the same integer-tick ASAP rules as the rest of the exact router. Their execution may update the physical resource-availability vector $\mathbf{t}$ and the accumulated gate cost G, but these updates are precisely the physical consequence prescribed by the selected JIT transition.

3. **Exclusion of independent non-routing work**
   Let $x \in V_{\mathrm{NR}}$ be an unmaterialised non-routing event that is not a required predecessor of g. Then $x \notin \mathrm{Anc}_{\mathrm{NR}}(g)$, and therefore $x \notin P(g, p_u)$. By construction, $M(u, g)$ does not materialise x.

   Thus a non-routing event is not consumed merely because it is ready. Independent non-routing work remains pending until it becomes a prerequisite of a subsequently selected two-qubit event or is materialised by the deterministic terminal flush.

4. **Preservation of two-qubit progress**
   By construction,
   $$P(g, p_u) \subseteq V_{\mathrm{NR}}$$
   Therefore prerequisite materialisation consumes no event in V_{2Q}. In particular, it consumes neither the selected event g nor any other eligible two-qubit event. Hence
   $$C_{2Q}(p_{u_g}) = C_{2Q}(p_u)$$
   and consequently
   $$p_{u_g} = p_u. \tag{C.3}$$
   The explicit decision about which two-qubit event is processed next remains the selection of
   $$g \in R_{2Q}(u)$$
   it is not hidden inside prerequisite materialisation. Since two-qubit eligibility is determined by the projected two-qubit dependency state, and this state is unchanged by M,
   $$R_{2Q}(u_g) = R_{2Q}(u)$$
   Thus every alternative event that was eligible before preparing g remains eligible at the projected logical level. The selected event g is removed from projected progress only by the subsequent two-qubit routing action.

5. **Mapping preservation**
   Every event materialised by $M(u, g)$ belongs to the non-routing class handled by the JIT prerequisite mechanism. Such operations do not perform routing and therefore do not modify the logical-to-physical mapping. Consequently,
   $$m_{u_g} = m_u \tag{C.4}$$
   The physical operands of g, its direct-or-blocked status, and the legal routing-action set available for g are therefore determined by the same mapping before and after prerequisite materialisation. The operator M can modify the physical-resource availability times and the accumulated gate cost. It does not, however, affect which routing actions are physically legal for the selected two-qubit event.

Taken together, these properties show that $M(u, g)$ materialises exactly the non-routing prerequisites required by the JIT semantics. It does not consume any independent non-routing event, any two-qubit event, or any routing action.

As a result, $M(u, g)$ leaves both the projected two-qubit progress and the logical-to-physical mapping unchanged. The set of eligible two-qubit events is therefore preserved. Hence u_g is exactly the canonical JIT pre-action state associated with selecting g. □

Corollary (routing-action preservation)
For every eligible event $g \in R_{2Q}(u)$,

$$m_{M(u,g)} = m_u$$

Hence the legal physical action set for g can be determined from the mapping of the original committed label:

$$\mathcal{A}_J(M(u,g), g) = \mathcal{A}_J(u, g)$$

where the equality concerns physical routing legality. The scheduling cost of those actions is evaluated from the updated availability state produced by $M(u, g)$.

C.1.5 JIT successor completeness

Let u be an active non-terminal JIT-exact search label with projected two-qubit progress p_u. Define $R_{2Q}(p_u)$ as the set of all unconsumed two-qubit events whose predecessors in the projected two-qubit DAG have already been consumed. These are precisely the two-qubit events that the JIT model permits to be selected next. For every $g \in R_{2Q}(p_u)$, the kernel first constructs the canonical pre-action state

$$u_g = M(u, g),$$

where M is the prerequisite-materialisation operator of Section C.1.4. By Lemma 4, this operation materialises exactly the missing non-routing prerequisites of g, consumes no additional two-qubit event and preserves the logical-to-physical mapping. Let $\mathcal{A}_J(g, m_u)$ denote the set of legal physical routing actions for g under mapping m_u. Since $m_{u_g} = m_u$ the same action set applies after prerequisite materialisation. For the restricted star action model,

$$|\mathcal{A}_J(g, m_u)| = \begin{cases} 1 & \text{if } g \text{ occupies a centre–leaf edge} \\ 3 & \text{if both operands of } g \text{ occupy leaves} \end{cases} \tag{C.5}$$

In the direct case,

$$\mathcal{A}_J(g, m_u) = \{D_{a,b}\}$$

where $D_{a,b}$ denotes direct execution of the selected interaction between logical operands a and b. In the blocked leaf–leaf case,

$$\mathcal{A}_J(g, m_u) = \{S_a, S_b, B_{a,b}\}$$

where S_a and S_b move either operand to the physical centre before executing the interaction, while $B_{a,b}$ is the fixed mapping-preserving BRIDGE construction.

Lemma 5 (JIT successor completeness).
For every active non-terminal JIT-exact label u, every eligible event

$$g \in R_{2Q}(p_u)$$

and every physical action

$$a \in \mathcal{A}_J(g, m_u)$$

native successor generation emits the corresponding legal JIT successor. Consequently, every legal next state admitted by the JIT search model $\mathcal{R}_J$ is represented.

Proof
The proof has two distinct completeness levels.

1. **Completeness of two-qubit event selection**
   At a non-terminal JIT label, the kernel constructs the complete eligible set $R_{2Q}(p_u)$. Successor generation iterates over every $g \in R_{2Q}(p_u)$ rather than selecting a single canonical routing event.

   By definition of projected-DAG eligibility, every event in this set is a legal candidate for the next consumed two-qubit event. Conversely, an unconsumed two-qubit event outside $R_{2Q}(p_u)$ still has at least one unconsumed projected predecessor and therefore cannot legally be selected at this state.
   For each eligible g, the kernel constructs

   $$u_g = M(u, g)$$

   By Lemma 4, this prerequisite-materialisation step executes exactly the missing non-routing predecessors required by g, preserves

   $$p_{u_g} = p_u, \qquad m_{u_g} = m_u$$

   and consumes no other event in V_{2Q}. Thus selecting g does not implicitly select or discard another two-qubit event.
   Therefore every two-qubit event that the projected JIT model permits to occur next is explicitly considered by native successor generation.

2. **Completeness of physical-action selection**
   Fix any $g \in R_{2Q}(p_u)$. After constructing u_g, the physical placement of the operands is still determined by m_u.

   If the operands occupy the centre and one leaf, the restricted hardware model admits exactly one physical action: direct execution. The kernel emits that unique successor.

   If both operands occupy leaves, the restricted action model admits exactly three alternatives: S_a, S_b, $B_{a,b}$. The native generator enumerates all three.

Each transition schedules the corresponding physical primitives, updates the resource-availability vector and accumulated gate cost, changes the mapping when required by a SWAP, and marks g as consumed in the projected progress. Hence, for every eligible event g, every action in

$$\mathcal{A}_J(g, m_u)$$

is generated and no legal action of the restricted JIT model is omitted.

Combining event-selection completeness with physical-action completeness, native successor generation enumerates exactly the legal next transitions of $\mathcal{R}_J$. □

Consequence for topological-order freedom
Because the event-selection loop considers every

$$g \in R_{2Q}(p)$$

at every JIT search state, no topological two-qubit order admitted by the projected JIT model is excluded by the event selector. If several independent or partially ordered two-qubit events are simultaneously eligible, each possible next choice is represented, and the same argument applies recursively to the resulting successors.

Thus every dependency-valid two-qubit event order admitted by the projected JIT semantics can, subject to the common restricted physical action model, be represented by a path through the JIT search graph.

Scope of both successor-completeness results
The completeness statements of Section C.1.2 and **??** are relative to their respective exact search spaces $\mathcal{R}_E$ and $\mathcal{R}_J$ Neither model includes proactive SWAPs that are unrelated to the currently selected routing event. Such transitions are not missing implementation branches; they are excluded by the common restricted physical action model. Likewise, alternative routing primitives outside the specified direct, operand-to-centre SWAP and fixed BRIDGE constructions are outside both search spaces. Extending either exact model to include such actions would require an enlarged successor relation and a corresponding new completeness proof.

C.1.6 JIT terminal-flush soundness

Let u be a committed JIT search label whose projected two-qubit progress indicates that no routing-relevant two-qubit event remains:

$$V_{2Q}^{\mathrm{rem}}(p_u) = \varnothing$$

Equivalently,

$$C_{2Q}(p_u) = V_{2Q}$$

At this point the JIT search has consumed every routing-relevant two-qubit event, but some supported non-routing events may still remain pending because they were never required as prerequisites of a later two-qubit event. Let

$$N_{\text{pending}}(p_u)$$

denote this residual set. The deterministic terminal-flush operator

$$u_{\text{term}} = F_{\text{term}}(u)$$

materialises the residual non-routing sub-DAG and produces the complete terminal schedule associated with u.

Lemma 6 (JIT terminal-flush soundness).
Let u be a committed pre-terminal-flush JIT label such that

$$V_{2Q}^{\text{rem}}(p_u) = \varnothing$$

Then $F_{\text{term}}(u)$ consumes every residual supported non-routing event exactly once, in dependency-valid order, introduces no new routing decision, preserves the logical-to-physical mapping, and produces the complete terminal routed schedule represented by u.

Proof
Since

$$V_{2Q}^{\text{rem}}(p_u) = \varnothing,$$

all routing-relevant two-qubit events have already been consumed. The residual logical sub-DAG therefore contains only events in V_{NR}. No further two-qubit routing choice is required because no such event remains in the projected dependency state.

The terminal flush processes the remaining events in $N_{\text{pending}}(p_u)$ according to the canonical dependency-valid order. Each residual event is emitted only after its remaining logical predecessors have been satisfied. Hence the flush respects the original circuit dependency DAG. Moreover, every event in $N_{\text{pending}}(p_u)$ is materialised exactly once. Events already contained in

$$N_{\text{done}}(p_u)$$

are not emitted again, while every residual supported event belongs to $N_{\text{pending}}(p_u)$ and is therefore eventually consumed by the flush. After completion,

$$N_{\text{pending}} = \varnothing.$$

Because the terminal flush contains no routing action and no SWAP-like mapping-changing transition, the logical-to-physical mapping is preserved:

$$m_{\text{term}} = m_u$$

The flush does, however, schedule the remaining physical primitives using the same integer-tick ASAP rules as the rest of the exact router. Consequently, the resource-availability vector changes from $\mathbf{t}_u$ to $\mathbf{t}_{\text{term}}$ and the accumulated gate cost changes from G_u to G_{term}. Thus,

$$u_{\text{term}} = F_{\text{term}}(u)$$

represents a fully materialised routed and scheduled circuit: every supported logical event has been consumed, all physical-resource updates have been applied, and no further routing or non-routing work remains. □

Terminal objective evaluation
A pre-flush state with $V_{2Q}^{\text{rem}}(p) = \varnothing$ is not yet assigned its final objective. After the deterministic terminal flush, the authoritative terminal objective is evaluated :

$$J_{\text{terminal}} = G_{\text{term}} + \kappa \max_i t_{\text{term},i}$$

Equivalently,

$$J_{\text{terminal}} = J(F_{\text{term}}(u))$$

rather than the objective associated with the pre-flush search label u.

Only after all remaining non-routing events have been materialised the incumbent comparison and terminal tie-breaking are performed. In this way, it is possible to avoid comparing terminal candidates before residual non-routing gates have contributed their gate cost and resource-availability updates.

Separation from the search frontier
The post-flush terminal is a completed solution rather than another ordinary JIT frontier label. In particular, the pre-flush state and its post-flush completion may share the same projected two-qubit progress, but they do not represent two alternative frontier states with the same key.
The terminal operator is instead a deterministic completion map

$$F_{\text{term}} : u \longmapsto u_{\text{term}}$$

applied only when the projected two-qubit search is complete. This is consistent with the pre-terminal-flush scope of the JIT progress-determinacy lemma.

C.2 Supporting reduction proofs

C.2.1 Common-continuations equivalence

Lemma 7 (**common-continuation equivalence at equal discrete key**).
Fix a scheduling mode $M \in \{E, J\}$ and let u and v be two committed labels of

that same mode. If

$$K(u) = K(v)$$

then u and v admit the same legal future routing and scheduling decisions under the semantics of mode M. Equivalently,

$$\mathcal{C}_M(u) = \mathcal{C}_M(v), \tag{C.6}$$

where $\mathcal{C}_M(u)$ denotes the set of legal symbolic continuations from u under the selected scheduling mode.

This equality concerns the available continuation structure. It does not state that corresponding continuations have the same objective value unless the dynamic coordinates of the two labels also coincide.

Proof
The argument is mode-specific.

1. **Forced-closure case**
   Suppose $M = E$. Equality of the discrete frontier key gives

   $$p_u = p_v, \qquad m_u = m_v$$

   Committed forced-closure labels are closure fixed points. Equal logical progress therefore identifies the same remaining dependency state under the deterministic forced-closure semantics, while equal mapping places every remaining logical operand on the same physical qubit in the two labels.

   Hence the same next routing-relevant events are available, with the same direct-or-blocked status and the same legal physical routing actions. After choosing corresponding actions, the mapping and logical-progress transitions are identical. The deterministic closure subsequently applied to each successor is also prescribed by the same forced-closure semantics.

   Repeating the argument after every routing decision shows that the two labels admit the same symbolic future continuations:

   $$\mathcal{C}_E(u) = \mathcal{C}_E(v)$$

2. **JIT-exact case**
   Suppose $M = J$. Again,

   $$p_u = p_v, \qquad m_u = m_v$$

   By Lemma 1, equal projected progress implies the same actual non-routing materialisation state. In particular,

   $$N^{\text{impl}}_{\text{done}}(u) = N^{\text{impl}}_{\text{done}}(v)$$

and the same non-routing obligations remain pending. Consequently, the two labels expose the same eligible two-qubit set:

$$R_{2Q}(u) = R_{2Q}(v)$$

For every $g \in R_{2Q}(u)$ the missing prerequisite set is also identical:

$$P(g, p_u) = P(g, p_v)$$

By Lemma 4, prerequisite materialisation preserves projected two-qubit progress and logical-to-physical mapping. Therefore, after preparing the same selected event g, the two labels still have the same mapping and expose the same physical routing situation. Since $m_u = m_v$ each eligible event has the same physical operands, the same direct-or-blocked status, and the same legal direct, SWAP, or BRIDGE action set in the two states. Applying corresponding actions produces the same discrete successor key.

The same argument can therefore be repeated inductively after every selected two-qubit event. When all routing-relevant events have been consumed, equal projected progress also determines the same residual non-routing obligations, so the deterministic terminal flush has the same symbolic completion in both cases. Hence

$$\mathcal{C}_J(u) = \mathcal{C}_J(v)$$

Thus, within either fixed scheduling mode, equality of the discrete frontier key identifies the same future decision structure. □

C.2.2 Exact duplicate removal

Fix a scheduling mode $M \in \{E, J\}$ Let u and v be two committed labels such that

$$K(u) = K(v)$$

and

$$G_u = G_v, \qquad \mathbf{t}_u = \mathbf{t}_v$$

Then u and v have identical attainable continuation costs and identical attainable terminal objective values. One of the two labels may therefore be removed without changing the optimum of the selected exact scheduling model.

Proof
By Lemma 7,

$$\mathcal{C}_M(u) = \mathcal{C}_M(v)$$

Thus every symbolic continuation available from one label has a corresponding continuation from the other with the same sequence of logical event selections,

prerequisite materialisations, routing actions, mapping updates, and deterministic mode-specific completion steps. The additional equalities

$$G_u = G_v, \qquad \mathbf{t}_u = \mathbf{t}_v$$

give identical dynamic initial conditions for every such common continuation. In the native representation, this means the same binary64 accumulated gate cost, the same centre availability tick, and the same four relative availability deltas, and therefore the same reconstructed five absolute physical-resource availabilities.

Because corresponding physical transitions use the same hardware operations from identical dynamic coordinates, they produce identical increments of G and identical updates of $\mathbf{t}$. By induction along any common continuation, the dynamic coordinates remain identical at every corresponding successor.

Hence every terminal continuation reachable from u has a corresponding terminal continuation from v with the same objective value, and vice versa. Therefore the attainable terminal-objective sets are identical.

Removing one exact duplicate cannot change the optimum. When several exact duplicates occur, the implementation may retain the deterministically preferred reconstruction trace according to the established tie policy; this choice affects only which equal-cost representative is reconstructed, not the optimal objective value. □

C.2.3 Incumbent pruning

Lemma 8 (incumbent pruning).
Fix a scheduling mode

$$M \in \{E, J\}$$

Let U be a validated feasible upper bound for the selected exact search model, and let $\mathrm{LB}_M(u)$ be an admissible lower bound satisfying

$$\mathrm{LB}_M(u) \leq J(R)$$

for every complete continuation R of label u admitted by mode M. Then pruning u whenever

$$\mathrm{LB}_M(u) > U$$

cannot remove a solution with objective smaller than the incumbent.

Proof
Because the incumbent is a complete feasible solution admissible for the selected exact model,

$$J_M^\star \leq U, \qquad M \in \{E, J\}$$

Now let u be an active label. By admissibility,

$$\mathrm{LB}_M(u) \leq J(R)$$

for every complete mode-M continuation R reachable from u. Therefore, if

$$\mathrm{LB}_M(u) > U$$

every such continuation satisfies

$$J(R) > U$$

and cannot improve the current incumbent. Removing u is consequently sound. If

$$\mathrm{LB}_M(u) = U,$$

the primary objective alone does not justify removing the label when an equal-J continuation could still be preferred by the deterministic terminal tie policy. Equality pruning is therefore permitted only when the corresponding tie metadata prove that no continuation of u can produce a preferred representative. □

Bound-off lower bound

The validated `bound-mode=off` configuration still uses the accumulated objective of the current label,

$$\mathrm{LB}_M^{\mathrm{off}}(u) = G_u + \kappa \max_i t_{u,i} \tag{C.7}$$

as an admissible lower bound for any continuation.

In forced-closure mode, each future forced operation and routing action adds a non-negative contribution to G, while the physical-resource availability times cannot decrease.

The same property holds for JIT-exact. Prerequisite materialisation, two-qubit routing actions, and the deterministic terminal flush all add non-negative gate-cost contributions, and their updates to the physical-resource availability times are monotone. Hence, for every complete JIT continuation R of u,

$$G_u + \kappa \max_i t_{u,i} \leq J(R)$$

Thus Equation (C.7) is admissible in both scheduling modes.

Validation of the incumbent is part of the soundness requirement. A malformed or incorrectly evaluated heuristic route could supply an artificially small value of U, invalidating the pruning inequality. The authoritative replay and objective recomputation performed before kernel handoff prevent an unverified heuristic score from being used directly as an exact pruning threshold.

C.2.4 Classic Dominance

Lemma 9 (**classic dominance**).
Classic dominance in Equation (8.3) preserves an optimal continuation within the selected scheduling mode.

Proof
By Equation (11.6), the two comparable labels admit the same legal continuation action sequences. Suppose u classically dominates v. Its accumulated gate cost is no larger and its physical-resource availability vector is componentwise no later. Consider any complete continuation $c \in \mathcal{C}_M(v)$ The same continuation belongs to $\mathcal{C}_M(u)$. Because corresponding future logical events are executed under the same mapping evolution and physical action sequence, they incur the same calibrated primitive-cost increments. Monotone ASAP scheduling preserves a componentwise no-later availability vector when the same future operations are applied to an initially no-later vector. Therefore the continuation from u reaches a terminal state with objective no larger than the corresponding continuation from v.

Hence removing the classically dominated label cannot remove a strictly better terminal solution. □

C.2.5 Reconstruction correctness

Lemma 10 (**reconstruction correctness**).
Following the stable parent chain of the selected exact solution and applying the reconstruction rule of the selected scheduling mode reproduces the state and complete routed trace from which the reported solution was derived.

Proof
Every successful non-root commit records a stable parent slot together with the reconstruction metadata required to identify the exact transition that produced the child. Persistent slots are append-only and every parent precedes its descendants, so repeatedly following parent identifiers yields a finite chain terminating at an initial-mapping root.

Reversing the collected descriptors restores chronological transition order. The reconstruction rule applied to those descriptors depends on the selected exact scheduling mode.

- **Forced-closure reconstruction**
  In the forced-closure mode, each committed non-root label records the routing event and physical action associated with the transition from its parent. Starting from the reconstructed initial mapping, replay executes the recorded

routing action and then reapplies the deterministic forced-closure rule:

$$u \longrightarrow T_E(u, g_E(u), a) \longrightarrow C_E(T_E(u, g_E(u), a)) = v$$

Because both the routing action and the subsequent closure are deterministic once the parent state and action descriptor are fixed, replay reconstructs the same progress, mapping, accumulated gate cost and resource availabilities as the committed child: p'_E, m'_E, G'_E, $\mathbf{t}'_E$. Repeating this procedure along the complete parent chain reconstructs the terminal forced-closure state represented by the selected slot.

- **JIT-exact reconstruction**
  In JIT-exact mode, each committed non-root transition records the selected two-qubit event g and the corresponding physical routing action a. The prerequisite-only intermediate state is not stored as a persistent search label. It is reconstructed deterministically from the parent state.

  Let the reconstructed parent have projected progress p. Replay first recomputes

  $$P(g, p) = \mathrm{Anc}_{\mathrm{NR}}(g) \setminus N_{\mathrm{done}}(p)$$

  and applies the canonical prerequisite-materialisation operator

  $$u_g = M(u, g)$$

  The recorded physical action a is then applied to the selected event g:

  $$v = T_J(M(u, g), g, a)$$

  For each reconstructed transition, replay checks that the resulting state matches the committed child in all coordinates relevant to future search:

  $$p_v = p', \qquad m_v = m', \qquad G_v = G', \qquad \mathbf{t}_v = \mathbf{t}'$$

  Replay also checks the selected event identifier, the legality of the routing action, the progress update, and the mapping update.

  After the final parent-chain routing transition, all projected two-qubit events have been consumed but residual non-routing work may still remain. The replay therefore applies the deterministic terminal operator

  $$u_{\mathrm{term}} = F_{\mathrm{term}}(u)$$

  and verifies the resulting terminal mapping, accumulated gate cost, resource-availability vector and complete physical trace.

Thus JIT reconstruction has the constructive form

$$u \longrightarrow M(u,g) \longrightarrow T_J(M(u,g),g,a) \longrightarrow v,$$
$$u_{\text{last}} \longrightarrow F_{\text{term}}(u_{\text{last}}) = u_{\text{term}}$$

Since prerequisite materialisation and terminal completion are deterministic, the parent progress together with the recorded pair (g,a) uniquely determines the reconstructed JIT transition. Repeating this procedure along the complete parent chain reconstructs the terminal state selected by the exact kernel. □

Native JIT constructive replay. Each exact transition selected by the kernel is associated with a record provided by a native JIT dynamic-programming trace. Its parent chain can therefore be replayed as a constructive witness of the search path.

For each parent descriptor, the specialised JIT replay recomputes the prerequisite materialisation, applies the recorded routing action, and checks the resulting progress and mapping. It also verifies the integer-tick resource availabilities, accumulated gate cost G, routing counters, and the final terminal flush.

This provenance-specific replay is a validation of the already constructed native trajectory, not a second search for a semantically compatible route. It does not alter the state graph, successor relation, frontier contents, dominance decisions or terminal selected by the exact kernel.

General semantic replay. The specialised constructive path is used only when native dynamic-programming provenance is available. Heuristic routes, routes produced by the reference IQM Spark algorithm, externally supplied routed QASM and other routes without trusted native parent-chain provenance continue to use the general semantic replay procedure.

The general validator independently checks logical trace equivalence, hardware legality, mapping consistency, physical-operation counts and the authoritative objective. Thus the native fast path changes how an internal JIT witness is verified, not the validation requirements imposed on externally produced routes.

C.3 Queue-exhaustion proofs

C.3.1 Complete forced-closure theorem proof

Initial-state completeness guarantees that every permitted logical-to-physical mapping is introduced into the forced-closure search, or is removed only when an exact duplicate or a sound dominance relation already provides a sufficient representative.

For every surviving non-terminal forced-closure label, forced-closure successor completeness enumerates every legal next transition in $\mathcal{R}_E$. The selected routing-relevant event is the one prescribed by the forced-closure semantics; every legal physical action for that event is generated, and deterministic post-action closure produces the corresponding next closed state. Therefore repeated successor generation represents every route in $\mathcal{R}_E$ unless one of its intermediate labels is safely removed.

Exact duplicate removal preserves an equivalent representative with identical continuation cost. Incumbent pruning removes a label only when an admissible mode-specific lower bound proves that none of its complete continuations can improve the validated incumbent. Classic and guarded Max-Plus dominance remove a label only when a no-worse representative exists for every legal common continuation, with the deterministic tie conditions preserved where required.

Transactional visibility ensures that these abstract removal and insertion arguments correspond to the implemented search: only complete forced-closure successors become visible in the arena, exact map, frontier and priority queue, and an interrupted or failed candidate cannot partially modify the persistent search state.

If the kernel returns `queue_exhausted`, every active representative has therefore been expanded or safely eliminated. Consequently, every feasible terminal route in $\mathcal{R}_E$ has either been explicitly reached or is covered by a retained or recorded representative whose attainable objective is no larger.

The selected terminal route $R_E^\star$ is therefore minimal over $\mathcal{R}_E$:

$$J(R_E^\star) = \min_{R \in \mathcal{R}_E} J(R).$$

Finally, forced-closure reconstruction follows the stable parent chain, reapplies each recorded routing action and deterministic closure, and reproduces the selected terminal state. Successful final validation confirms that the emitted routed QASM realises that reconstructed solution. □

C.3.2 Complete JIT theorem proof

Initial-state completeness guarantees that, for every one of the 120 logical-to-physical mappings, the JIT search introduces the corresponding zero projected-progress root, or safely subsumes it through exact duplicate or sound dominance machinery. No initial forced closure restricts the JIT root set.

JIT progress determinacy establishes that, for every committed pre-terminal-flush search label, projected two-qubit progress p uniquely determines the materialised and pending non-routing obligations:

$$\Phi(p) = (N_{\text{done}}(p), N_{\text{pending}}(p))$$

Hence the compact JIT state does not omit an independent non-routing search coordinate.

Consider now any active non-terminal JIT label u. For every eligible two-qubit event $g \in R_{2Q}(p_u)$, prerequisite-materialisation soundness guarantees that $M(u, g)$ materialises exactly the still-missing non-routing predecessors required by g, in dependency-valid order. It consumes neither an independent non-routing event nor another two-qubit event and does not modify the current mapping.

JIT successor completeness then has two levels. First, native generation considers every event $g \in R_{2Q}(p_u)$ so no two-qubit topological order admitted by the projected JIT model is excluded by the event selector. Second, for each selected g, every legal physical action in $\mathcal{A}_J(g, m_u)$ is generated: one direct action for a centre–leaf interaction, or the two operand-to-centre SWAP actions and the fixed BRIDGE action for a blocked leaf–leaf interaction.

Therefore repeated JIT successor generation represents every transition sequence admitted by $\mathcal{R}_J$, unless an intermediate committed label is safely removed by the common reduction machinery.

Exact duplicate removal preserves an identical continuation representative. Incumbent pruning removes a label only when a lower bound that is admissible for the selected mode shows that no complete JIT continuation can improve the validated incumbent.

Classic dominance and the guarded Max-Plus relation are sound because labels with the same JIT frontier key admit the same legal continuations. Equal projected progress implies the same remaining non-routing and two-qubit events, while equal mappings imply the same set of future physical routing actions.

Transactional visibility ensures that prerequisite-only states such as $M(u, g)$ remain temporary and are not inserted into the persistent search frontier. Only a complete materialisation-plus-routing successor is allowed to become visible in the persistent search structures. Consequently, the implemented queue and frontiers contain exactly the committed state type assumed by the abstract JIT proofs.

When no two-qubit event remains, JIT terminal-flush soundness guarantees that the deterministic operator F_{term} materialises every residual supported non-routing event exactly once and in dependency-valid order. The resulting terminal state preserves the final mapping, correctly updates G and the resource availabilities, and is evaluated only after completion:

$$J_{\text{terminal}} = G_{\text{terminal}} + \kappa \max_i t_{\text{terminal},i}$$

Thus every terminal considered by the incumbent mechanism represents a complete routed and scheduled circuit rather than merely complete projected two-qubit progress.

If the kernel returns `queue_exhausted`, no active unexpanded JIT representative remains. Every route in $\mathcal{R}_J$ has therefore either been represented through the complete successor relation or safely covered by duplicate removal, admissible incumbent pruning or sound dominance. The best recorded complete terminal consequently satisfies

$$J(R_J^\star) = \min_{R \in \mathcal{R}_J} J(R)$$

Finally, JIT reconstruction follows the stable parent chain, recomputes each prerequisite set $P(g, p)$, materialises it, executes the recorded physical action, verifies the resulting progress, mapping, accumulated gate cost and resource availabilities, and applies the deterministic terminal flush after the last two-qubit transition. Successful final validation confirms that the emitted routed QASM realises $R_J^\star$. □

C.4 Counterexamples between forced closure and JIT-exact

C.4.1 Counterexample forced closure better

Consider the five-qubit logical circuit shown in Figure C.1.

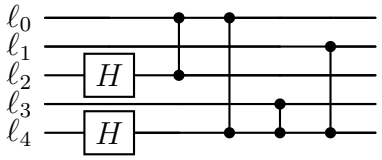


Figure C.1: Logical circuit C_A for which the forced-closure optimum is strictly smaller than the JIT-exact optimum. The first two Hadamard gates are independent and are simultaneously ready before the first CZ.

A complete enumeration of the finite exact search spaces using the hardware configuration with signature 9da78e gives

$$J_E^\star(C_A) = 0.2340850905427162$$

and

$$J_J^\star(C_A) = 0.24091175954517224$$

Hence

$$\boxed{J_E^\star(C_A) < J_J^\star(C_A)} \tag{C.8}$$

The two optima use the same optimal initial mapping and the same physical routing sequence: the first logical CZ is executed by a mapping-preserving BRIDGE and

the remaining three CZs are direct. Their accumulated gate cost G is the same, so the difference between the two cases is temporal. At the initial forced-closure fixed point, both $H(\ell_2)$ and $H(\ell_4)$ are materialised. Under the optimal mapping, they are assigned to different physical resources, allowing the ASAP scheduler to overlap their two 20 ns durations.

In JIT-exact, only $H(\ell_2)$ is materialised at this stage because $H(\ell_4)$ is not a prerequisite of the first selected two-qubit event. The operation $H(\ell_4)$ is not materialised until it is required by the later interaction involving ℓ_4. Only in this case its physical resource is on the routing critical path, and the delayed Hadamard contributes an additional 20 ns to the global makespan. Consequently,

$$T_E(C_A) = 588\,\text{ns}, \qquad T_J(C_A) = 608\,\text{ns}$$

This means that forced closure can be advantageous because early deterministic materialisation may expose parallelism that, instead, the canonical JIT semantics does not reproduce.

C.4.2 Counterexample JIT better

Consider now the circuit shown in Figure C.2.

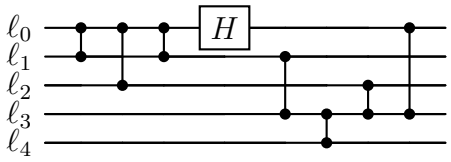


Figure C.2: Logical circuit C_B for which the JIT-exact optimum is strictly smaller than the forced-closure optimum. The projected two-qubit DAG is a total chain, so the improvement cannot be attributed to alternative topological ordering of the CZ events.

Its projected two-qubit dependency DAG is totally ordered:

$$g_1 \prec g_2 \prec g_3 \prec g_4 \prec g_5 \prec g_6 \prec g_7$$

Therefore JIT-exact has no additional freedom in the ordering of the two-qubit events in this example. Both native exact runs terminate with `queue_exhausted`. Using the hardware configuration with signature 9da78e , they certify

$$J_E^\star(C_B) = 0.33240060569933694$$

and

$$J_J^\star(C_B) = 0.3255056270606887$$

hence

$$\boxed{J_J^\star(C_B) < J_E^\star(C_B)} \tag{C.9}$$

The optimal initial mapping places ℓ_0 at the physical centre. After the first three CZ events, $H(\ell_0)$ becomes logically ready. Forced closure immediately materialises it while $\ell_0 \mapsto q_2$. The subsequent routing action swaps the logical identities occupying q_2 and q_4, so that $\ell_0 \mapsto q_4$.
Since $H(\ell_0)$ is not a prerequisite of the next selected two-qubit event, JIT-exact does not execute it, even if it first becomes ready. Therefore, the gate remains pending. Meanwhile, the SWAP changes the mapping and is later materialised on q_4.

This operation changes both the gate-cost and scheduling contributions. The one-qubit error associated with q_4 is lower than that of the physical centre q_2, giving

$$G_J(C_B) < G_E(C_B)$$

Moreover, after the SWAP, the delayed Hadamard can be scheduled in parallel with an independent CZ and, therefore, does not extend the critical path. The certified makespans are

$$T_J(C_B) = 820\,\text{ns} < 840\,\text{ns} = T_E(C_B)$$

Thus the JIT solution is strictly better in both components of the implemented objective:

$$G_J(C_B) < G_E(C_B), \qquad T_J(C_B) < T_E(C_B)$$

C.4.3 Consequence

The two counterexamples establish opposite strict inequalities:

$$\exists\, C_A : \qquad J_E^\star(C_A) < J_J^\star(C_A)$$

while

$$\exists\, C_B : \qquad J_J^\star(C_B) < J_E^\star(C_B)$$

Consequently,

$$\neg\,[\forall C,\ J_J^\star(C) \le J_E^\star(C)] \tag{C.10}$$

and

$$\neg\,[\forall C,\ J_E^\star(C) \le J_J^\star(C)] \tag{C.11}$$

Therefore,

$$\boxed{\text{neither scheduling semantics universally dominates the other with respect to } J}$$

The same counterexamples also exclude a universal nesting relation between the two families of scheduling spaces. If

$$\mathcal{R}_E(C) \subseteq \mathcal{R}_J(C)$$

held for every circuit C, minimisation over the larger JIT set would imply

$$J_J^\star(C) \leq J_E^\star(C)$$

for every C, contradicting Equation (C.8). Hence

$$\neg\,[\forall C,\ \mathcal{R}_E(C) \subseteq \mathcal{R}_J(C)]$$

Conversely, if

$$\mathcal{R}_J(C) \subseteq \mathcal{R}_E(C)$$

held universally, then

$$J_E^\star(C) \leq J_J^\star(C)$$

would hold universally, contradicting Equation (C.9). Therefore

$$\neg\,[\forall C,\ \mathcal{R}_J(C) \subseteq \mathcal{R}_E(C)]$$

This establishes non-nesting of the *families* of scheduling spaces across supported circuits. It does not assert that both set non-inclusions must necessarily hold simultaneously for every individual circuit.
The operational interpretation is equally important. Forced closure may occasionally obtain a lower objective by materialising independent non-routing work early enough for the ASAP scheduler to overlap it with other work. JIT-exact may instead obtain a lower objective by keeping such work pending across mapping changes, thereby changing its physical placement and allowing a more favourable later schedule. The two mechanisms are genuinely different, which is why neither exact scheduling semantics admits a universal objective ordering.

C.5 Counterexamples between forced closure and permutation-based router

C.5.1 Forced closure is better

Consider the circuit

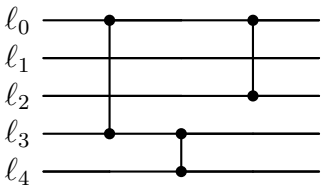


Figure C.3: Logical circuit C_A used in the comparison between forced-closure J-exact and the permutation-based IQM Spark router. For this example, the forced-closure optimum attains a smaller hardware-aware objective J than the canonical minimum-SWAP route returned by the paper algorithm.

The projected two-qubit dependency structure contains only three CZ events. The permutation-based router evaluates the two feasible topological orders and terminates exactly with a minimum inserted-SWAP count of

$$N_{\mathrm{SWAP},P} = 1$$

Using the hardware configuration with signature 9da78e , its selected canonical route has

$$T_P(C_A) = 572\,\mathrm{ns}$$

and post-hoc objective

$$J_P(C_A) = 0.230017536307$$

The forced-closure native search also terminates with `queue_exhausted`, and therefore certifies

$$J_E^\star(C_A) = 0.19290424612819387.$$

The corresponding optimal route uses

$$N_{\mathrm{SWAP},E} = 0, \qquad N_{\mathrm{BRIDGE},E} = 1,$$

with

$$T_E(C_A) = 488\,\mathrm{ns}.$$

Hence

$$\boxed{J_E^\star(C_A) < J_P(C_A)} \tag{C.12}$$

Numerically,

$$J_P(C_A) - J_E^\star(C_A) = 0.03711329017880613.$$

The mechanism is structural. The permutation-based router represents routing changes through centre–leaf SWAPs and minimises their number. Forced-closure J-exact has a larger physical action vocabulary that also includes the mapping-preserving BRIDGE.

For this circuit, the minimum-J route uses one BRIDGE and no SWAPs. It has both a lower accumulated physical gate cost and a shorter ASAP makespan than the canonical one-SWAP route returned by the permutation-based algorithm.

The example also shows that minimum inserted-SWAP count and minimum hardware-aware objective are distinct optimisation criteria. A route that uses fewer SWAPs under one routing model does not necessarily minimise J under another.

C.5.2 Counterexample B: the permutation router is better

Now consider

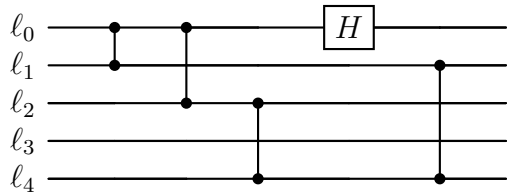


Figure C.4: Logical circuit C_B used in the comparison between forced-closure J-exact and the permutation-based IQM Spark router. In this case, the canonical minimum-SWAP route returned by the paper algorithm attains a smaller post-hoc hardware-aware objective J than the forced-closure optimum.

Let the four CZ events be

$$g_1,\ g_2,\ g_3,\ g_4$$

in circuit order. Their projected dependency DAG is a total chain,

$$g_1 \prec g_2 \prec g_3 \prec g_4$$

Thus the permutation-based router evaluates exactly one two-qubit topological order. Its exact routing search certifies

$$N_{\mathrm{SWAP},P} = 1$$

and, using the hardware configuration with signature 9da78e , the reconstructed route has

$$T_P(C_B) = 608\,\mathrm{ns},$$

$$J_P(C_B) = 0.239224975038$$

Forced-closure J-exact also terminates with `queue_exhausted`. Its exact optimum is

$$J_E^\star(C_B) = 0.24611995367622366$$

with

$$N_{\mathrm{SWAP},E} = 1, \qquad N_{\mathrm{BRIDGE},E} = 0$$

and

$$T_E(C_B) = 628\,\mathrm{ns}$$

Therefore

$$\boxed{J_P(C_B) < J_E^\star(C_B)} \tag{C.13}$$

The objective difference is

$$J_E^\star(C_B) - J_P(C_B) = 0.006894978638223664$$

This example is particularly informative because the two solutions use the same number of inserted SWAPs and no BRIDGE, while the two-qubit order itself is fixed. The difference comes neither from a lower SWAP count nor from a different choice of topological order. After the second CZ, the logical gate $H(\ell_0)$ is ready. Under forced closure, it is materialised immediately while ℓ_0 is still mapped to the physical centre q_2. The subsequent routing SWAP then moves ℓ_0 from q_2 to q_4.

On the other hand, the canonical paper reconstruction does not use the forced-closure rule.

The Hadamard can remain pending while the mapping changes because it is not required before the next selected two-qubit event. It is subsequently placed on q_4, where its physical error is slightly smaller, and its duration can be absorbed by available ASAP parallelism. Consequently, the paper route obtains both

$$G_P(C_B) < G_E(C_B)$$

and

$$T_P(C_B) = 608\,\mathrm{ns} < 628\,\mathrm{ns} = T_E(C_B)$$

Thus a SWAP-optimal paper route can attain a strictly smaller post-hoc hardware-aware objective than the exact optimum of the restricted forced-closure scheduling model.

C.5.3 Consequence

The two certified examples establish opposite strict inequalities:

$$\exists\, C_A : \qquad J_E^\star(C_A) < J_P(C_A)$$

and

$$\exists\, C_B : \qquad J_P(C_B) < J_E^\star(C_B)$$

It follows that

$$\neg\,[\forall C,\ J_E^\star(C) \le J_P(C)] \tag{C.14}$$

and

$$\neg\,[\forall C,\ J_P(C) \leq J_E^\star(C)] \tag{C.15}$$

Therefore,

$$\boxed{\begin{array}{c}\text{the forced-closure optimum and the post-hoc } J \text{ of the} \\ \text{canonical paper route are not universally ordered}\end{array}} \tag{C.16}$$

Equivalently, there exist supported circuits for which forced closure attains a strictly smaller hardware-aware objective and other supported circuits for which the minimum-SWAP route selected by the permutation-based router attains a strictly smaller post-hoc J.

This objective non-comparability must not be confused with the exactness criteria of the two algorithms. Forced-closure J-exact is exact for

$$\min_{R \in \mathcal{R}_E(C)} J(R),$$

whereas the permutation-based router is exact for inserted SWAP count. The quantity $J_P(C)$ is an authoritative post-hoc evaluation of the selected minimum-SWAP route, not the minimum of J over the route family enumerated by the paper algorithm.

For this reason, the opposite objective inequalities above do not by themselves establish a symmetric non-inclusion relation between the two route spaces. In particular, the fact that

$$J_E^\star(C_A) < J_P(C_A)$$

may result from the paper selecting a different route because it optimises SWAP count rather than J. Conversely, the strict inequality

$$J_P(C_B) < J_E^\star(C_B)$$

does imply that the specific canonical paper route $R_P(C_B)$ cannot belong to the forced-closure search space $\mathcal{R}_E(C_B)$, since otherwise the definition of $J_E^\star(C_B)$ would require

$$J_E^\star(C_B) \leq J(R_P(C_B)) = J_P(C_B),$$

contradicting Equation (C.13).

C.6 Detailed complexity bounds

C.6.1 Time complexity

The exact router is output-sensitive: its running time depends not only on the input circuit size, but also on the number of search labels that survive duplicate

rejection, pruning and dominance. The two scheduling modes expose different successor structures and are therefore considered separately.
Let N denote the total number of committed labels and let W_k denote the active width of frontier key k. With a binary heap, queue insertion and removal require $O(\log N)$ time. Under the validated linear-frontier implementation, testing a candidate against an active frontier requires $O(W_k)$ comparisons, with a constant number of classic and transformed dynamic coordinates.

Forced-closure complexity

Let E_x^E denote the set of labels expanded in the forced-closure mode. Every non-terminal closed label exposes one deterministic routing-relevant two-qubit event. Let $b_E(u)$ be the number of legal physical successors of that event. Under the restricted star action model,

$$b_E(u) = \begin{cases} 1 & \text{for a directly executable centre–leaf interaction} \\ 3 & \text{for a blocked leaf–leaf interaction} \end{cases}$$

and therefore

$$b_E(u) \leq 3.$$

Let $C_{\text{closure}}(u, a)$ denote the cost of applying deterministic post-action forced closure after physical action a from state u. A run-dependent description of the native search cost is then

$$O\left(\sum_{u \in E_x^E} \left[\log N + \sum_{a \in \mathcal{A}_E(u)} \left(C_{\text{closure}}(u, a) + \log N + W_{K(v_{u,a})}\right)\right]\right) \tag{C.17}$$

where $v_{u,a}$ is the complete closed candidate generated by action a. The first $\log N$ term accounts for the queue pop, while the remaining terms represent successor construction, frontier processing and possible queue insertion.

A coarse path-count intuition can be obtained by considering a path ρ containing $N_{\text{blk}}(\rho)$ blocked routing decisions. If their choices were independent, the number of corresponding routing-action histories would scale as $5!\,3^{N_{\text{blk}}(\rho)}$ This is not a bound on the number of committed labels, because histories may reconverge and be removed by duplicate detection, dominance or incumbent pruning.

JIT-exact complexity

The JIT-exact mode has a different immediate branching structure. Let E_x^J denote its expanded-label set and define

$$r_u = |R_{2Q}(p_u)|$$

the number of simultaneously eligible two-qubit events at label u. For every $g \in R_{2Q}(p_u)$, let

$$a(g) = \begin{cases} 1 & \text{if } g \text{ is directly executable} \\ 3 & \text{if } g \text{ is blocked} \end{cases}$$

The number of candidate routing transitions exposed by expanding u is therefore

$$b_J(u) = \sum_{g \in R_{2Q}(p_u)} a(g) \tag{C.18}$$

and satisfies

$$b_J(u) \leq 3r_u \tag{C.19}$$

Thus the forced-closure bound $b \leq 3$ does not apply to JIT-exact. Let $C_{\text{mat}}(u, g)$ denote the cost of discovering and materialising the missing prerequisite set $P(g, p_u)$ for selected event g. Since this prerequisite state is constructed once for g and then shared conceptually by all physical actions associated with that event, a run-dependent JIT cost can be written as

$$O\left(\sum_{u \in E_x^J} \left[\log N + \sum_{g \in R_{2Q}(p_u)} \left(C_{\text{mat}}(u, g) + \sum_{a \in \mathcal{A}_J(g, m_u)} \left[\log N + W_{K(v_{u,g,a})} \right] \right) \right] \right) \tag{C.20}$$

where $v_{u,g,a}$ is the complete JIT successor obtained after prerequisite materialisation and physical action a.

The important distinction is that JIT complexity is governed not only by the physical routing alternatives of a selected interaction, but also by topological-order freedom. Whenever $r_u > 1$ several dependency-valid two-qubit events may be selected next. Repeated choices among such eligible events can generate a very large number of valid two-qubit event orders even when the local quantity r_u remains modest.

For example, two independent ordered chains containing m and n two-qubit events admit

$$\binom{m+n}{m}$$

dependency-valid interleavings. Dynamic programming may merge many of the resulting histories, but different interleavings can also produce Pareto-incomparable resource-availability vectors and therefore survive within the same frontier.

Absence of a polynomial input-size guarantee

Neither Equation (C.17) nor Equation (C.20) provides a polynomial bound in the circuit input size. The quantities

$$|E_x^E|, \qquad |E_x^J|, \qquad N, \qquad W_k$$

depend on the search itself and may grow exponentially on difficult instances.

In forced-closure mode, the main explicit source of branching is the repeated choice among routing actions for blocked two-qubit events. In JIT-exact, this routing branching is combined with the different dependency-valid orders of the remaining two-qubit events.

In both modes, trade-offs between gate cost and execution time can also produce large Pareto frontiers. As a result, search histories that reach similar logical states may still remain as distinct labels rather than collapsing to a single representative.

Preprocessing, upper-bound generation and final reconstruction or validation add further costs outside the native search loop. In particular, provenance-specific JIT replay changes the post-search validation cost but does not change either Equation (C.20) or the exact search state graph.

Finally, if device width n were treated as variable, enumeration of all full-width initial bijections already contributes $n!$ roots. The resource-availability dimension also grows with n, while the number and structure of coupling-dependent routing alternatives depend on the hardware graph. The specialised five-qubit implementation should therefore not be interpreted as a scalable general-n exact routing algorithm.

This lack of a polynomial input-size guarantee is consistent with the known hardness of qubit routing even on linear-nearest-neighbour architectures [7].

C.6.2 Space complexity

The arena stores $O(N)$ hot and cold records, where N is the cumulative number of committed labels. In the validated compact layout, each committed slot uses a 64-byte hot record and a 12-byte cold reconstruction record, for a total of 76 bytes of label-related storage per committed state, excluding chunk slack, exact-map nodes, active-frontier containers, priority-queue entries, DAG tables and general container overhead.

Because dominated labels may remain reachable through the parent chains of surviving descendants, their reconstruction records cannot necessarily be released when they leave the active frontier. Memory usage therefore follows the cumulative number of committed labels more closely than the number of currently active labels.

Compact JIT pending-state representation
JIT-exact introduces additional scheduling freedom, but it does not require each search label to store an arbitrary subset of pending non-routing events. By Lemma 1,

the non-routing materialisation state is uniquely determined by the projected two-qubit progress:

$$\Phi(p) = (N_{\text{done}}(p), N_{\text{pending}}(p))$$

Consequently, the pending non-routing state can be reconstructed from p rather than stored as an independent per-label bit set.

This property is important for the compact JIT representation. Without progress determinacy, a label might in principle need to distinguish many different subsets of already materialised non-routing operations at the same two-qubit progress, introducing an additional combinatorial state component. JIT instead retains the common discrete frontier key

$$K(u) = (p_u, m_u)$$

while the materialisation status implied by p_u is recovered deterministically when prerequisite or terminal materialisation is required.

The priority queue may contain stale entries, so its size is not identical to the number of live frontier labels. Exact-map and frontier occupancy likewise reflect current or historical dynamic states according to the maintenance policy of the selected backend. These structures add $O(N)$ storage in the worst run-dependent sense. Overall native search memory is therefore $O(N)$, with a practically significant constant factor. This motivates the compact hot/cold split, stable arena indirection and reconstruction of JIT non-routing obligations from projected progress rather than explicit per-label storage.

C.7 Proof-obligation summary

Table C.1: Proof obligations and the implementation mechanisms that discharge them within the selected exact scheduling model.

Obligation	Mechanism	Residual scope
Initial-state completeness	All 5! = 120 logical-to-physical bijections are offered to the exact state space. Forced-closure roots undergo deterministic initial closure; JIT roots start from zero projected two-qubit progress with no automatic non-routing materialisation.	Exactly five logical and five physical qubits; mode-specific root semantics

Continued on next page

Table C.1 – continued from previous page

Obligation	Mechanism	Residual scope
Forced-closure successor completeness	Every closed non-terminal label exposes the single routing event $g_E(u)$. A centre–leaf event has one direct successor; a blocked leaf–leaf event has exactly the two operand-to-centre SWAP successors and the fixed BRIDGE successor.	Restricted forced-closure search space $\mathcal{R}_E$
Forced-closure soundness	Initial and post-action closure deterministically materialise exactly the operations prescribed by the forced-closure scheduling semantics until a closure fixed point is reached.	Deterministic forced-closure model only
JIT progress determinacy	Projected progress p uniquely determines the consumed two-qubit set and the non-routing materialisation state $\Phi(p) = (N_{\text{done}}(p), N_{\text{pending}}(p))$. No independent pending-event bit set is part of the search state.	Committed pre-terminal-flush JIT labels
JIT prerequisite soundness	For each selected $g \in R_{2Q}(p)$, the kernel materialises exactly $$P(g,p) = \text{Anc}_{\text{NR}}(g) \setminus N_{\text{done}}(p),$$ in dependency-valid order. No independent non-routing event or additional two-qubit event is consumed, and the mapping is preserved.	Canonical JIT prerequisite materialisation
JIT successor completeness	For every eligible event $g \in R_{2Q}(p)$, all legal physical actions are enumerated: one direct action for centre–leaf placement or the two operand-to-centre SWAPs and fixed BRIDGE for leaf–leaf placement.	Restricted JIT search space $\mathcal{R}_J$
JIT terminal completeness	When no two-qubit event remains, the deterministic terminal flush consumes every residual supported non-routing event exactly once in dependency-valid order before terminal J is evaluated.	Post-search completion; no additional routing decision
Duplicate safety	Within a fixed scheduling mode, equal frontier key and identical dynamic coordinates imply identical continuation costs. One deterministic representative can therefore be retained.	No cross-mode duplicate comparison

Continued on next page

Table C.1 – continued from previous page

Obligation	Mechanism	Residual scope
Dominance safety	Dominance is applied only within a fixed mode and common frontier key. Forced-closure common continuations follow from equal closed progress and mapping; JIT common continuations follow from progress determinacy together with equal mapping. Classic and guarded Max-Plus inequalities then preserve a no-worse continuation.	Frozen objective, ASAP model, and mode-specific continuation semantics
Incumbent pruning safety	Only a validated feasible upper bound U for the selected model is used. An admissible $\mathrm{LB}_M(u)$ permits rejection only when it cannot improve U. The bound-off lower bound remains $G_u + \kappa \max_i t_{u,i}$.	Selected mode $M \in \{E, J\}$; authoritative incumbent evaluation
Transactional visibility	Prerequisite materialisation, routing actions and forced closure are candidate-local until a complete successor passes all commit checks. No partial state becomes visible in the arena, exact map, frontier or queue.	Native transactional implementation and deadline handling
Reconstruction safety	Forced-closure replay applies the recorded action followed by deterministic closure. JIT replay recovers the selected event g, recomputes $P(g, p)$, materialises its prerequisites, applies the recorded action and performs the terminal flush after the final two-qubit transition.	Stable parent chain and mode-specific reconstruction metadata
Optimality certificate	`queue_exhausted` certifies the minimum objective over the search space of the selected scheduling mode: $J_E^\star = \min_{R \in \mathcal{R}_E} J(R), \qquad J_J^\star = \min_{R \in \mathcal{R}_J} J(R)$ Timeout or interruption yields only a feasible incumbent, not an optimality certificate.	Exact only within the selected model and restricted physical action set

Bibliography

[1] A. Gnatowski, A. Biskup, K. Berniker, M. Owczarek, and E. Sikora, "Quantum circuit compilation for IQM spark," in *Dependable Computer Systems: Methods, Models, and Applications*, vol. 2014 of *Lecture Notes in Networks and Systems*, pp. 142–152, Springer Nature Switzerland, 2026.

[2] G. Li, Y. Ding, and Y. Xie, "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*, ASPLOS '19, pp. 1001–1014, Association for Computing Machinery, 2019.

[3] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[4] M. Y. Siraichi, V. F. dos Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO '18, pp. 113–125, Association for Computing Machinery, 2018.

[5] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, vol. 135 of *Leibniz International Proceedings in Informatics*, pp. 5:1–5:32, 2019.

[6] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming," *ACM Transactions on Quantum Computing*, vol. 4, no. 1, pp. 1–31, 2023.

[7] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, and Y. Okamoto, "Algorithmic theory of qubit routing in the linear nearest neighbor architectures," *ACM Transactions on Quantum Computing*, vol. 6, no. 3, pp. 1–17, 2025.

[8] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021318, 2019.

[9] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, pp. 369–395, 2020.

[10] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pp. 987–999, Association for Computing Machinery, 2019.

[11] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "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*, ASPLOS '19, pp. 1015–1029, Association for Computing Machinery, 2019.

[12] P. Murali, N. M. Linke, M. Martonosi, A. Javadi-Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: Architectural comparisons and design insights," in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, pp. 527–540, Association for Computing Machinery, 2019.

[13] L. Lao, H. van Someren, I. Ashraf, and C. G. Almudever, "Timing and resource-aware mapping of quantum circuits to superconducting processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 359–371, 2022.

[14] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y.-H. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, pp. 360–374, Association for Computing Machinery, 2021.

[15] IQM Quantum Computers, "IQM Spark." `https://iqm.tech/products/iqm-spark/`. Official product page.

[16] Politecnico di Torino, "The first IQM quantum computer in italy is turned on in turin." `https://www.polito.it/en/polito/communication-and-press-office/poliflash/the-first-iqm-quantum-computer-in-italy-is-turned-on-in`, 2025.

[17] LINKS Foundation, "Lagrange quantum computing service documentation." `https://pages.linksfoundation.com/links-iqm-spark/docs/`, 2026.

[18] P. Viviani, F. Bertone, G. Vitali, E. Dri, F. Stirano, G. Caragnano, F. Lubrano, A. Nespola, O. Terzo, M. Cocuzza, B. Montrucchio, G. Turvani, G. Bertaina, M. Coisson, D. Calonico, F. Pirri, and P. Asinari, "Lagrange: Operating italy's first publicly-accessible quantum computer for research and education," 2026. arXiv:2604.21695.

[19] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 10th anniversary ed., 2010.

[20] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.

[21] T. S. Metodi, D. D. Thaker, A. W. Cross, F. T. Chong, and I. L. Chuang, "Scheduling physical operations in a quantum information processor," in *Quantum Information and Computation IV*, vol. 6244 of *Proceedings of SPIE*, p. 62440T, 2006.

[22] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025004, 2018.

[23] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, vol. 70, pp. 43–50, 2020.

[24] A. Russo, M. Simoni, D. Volpe, G. A. Cirillo, and M. Graziano, "Exploring the advantages of layout procedure with fully-connected quantum computing technologies," *Advanced Quantum Technologies*, vol. 7, no. 1, p. 2300128, 2024.

[25] M. Y. Siraichi, V. F. dos Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 120:1–120:29, 2019.

[26] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.

[27] A. M. Childs, E. Schoute, and C. M. Unsal, "Circuit transformations for quantum architectures," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, vol. 135 of *Leibniz International Proceedings in Informatics*, pp. 3:1–3:24, 2019.

[28] S. Niu, A. Suau, G. Staffelbach, and A. Todri-Sanial, "A hardware-aware heuristic for the qubit mapping problem in the NISQ era," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–14, 2020. Art. no. 3101614.

[29] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pp. 1–6, Association for Computing Machinery, 2019. Article 142.

[30] B. Tan and J. Cong, "Optimal qubit mapping with simultaneous gate absorption," in *2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8, 2021.

[31] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via MaxSAT," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1078–1091, 2022.

[32] L. Burgholzer, S. Schneider, and R. Wille, "Limiting the search space in optimal quantum circuit mapping," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 466–471, IEEE, 2022.

[33] R. Wille and L. Burgholzer, "MQT QMAP: Efficient quantum circuit mapping," in *Proceedings of the 2023 International Symposium on Physical Design*, ISPD '23, pp. 198–204, Association for Computing Machinery, 2023.

[34] F. Wagner, A. Bärmann, F. Liers, and M. Weissenbäck, "Improving quantum computation by optimized qubit routing," *Journal of Optimization Theory and Applications*, vol. 197, no. 3, pp. 1161–1194, 2023.

[35] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 7 2017. arXiv:1707.03429.

[36] A. W. Cross, A. Javadi-Abhari, T. Alexander, N. de Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, and B. R. Johnson, "OpenQASM 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–50, 2022.

[37] Politecnico di Torino, "Quantum computing: Tools from the future." `https://orizzonti.polito.it/en/2025/quantum-computing-tools-from-the-future/`, 2025.

[38] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Chichester: John Wiley & Sons, 1992.

[39] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.

[40] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024. arXiv:2405.08810.

[41] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2021.

[42] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[43] SciPy Community, "`scipy.stats.wilcoxon` — SciPy v1.13.1 documentation." `https://docs.scipy.org/doc/scipy-1.13.1/reference/generated/scipy.stats.wilcoxon.html`, 2024. Documented defaults: `zero_method='wilcox'`, `correction=False`, `method='auto'`.

[44] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[45] S. Li, X. Zhou, and Y. Feng, "Qubit mapping based on subgraph isomorphism and filtered depth-limited search," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1777–1788, 2021.