A SABRE-inspired Circuit-Depth Surrogate for Two-level Parallel Genetic Qubit Mapping

Anonymized

No Institute Given

Abstract. Qubit mapping is a key compilation step for NISQ devices, where sparse qubit connectivity requires additional routing operations. Genetic Algorithms (GAs) can improve mapping quality, but their scalability is limited by repeated calls to costly routing heuristics during fitness evaluation. We build on an existing parallel GA for qubit mapping and introduce PSiD (Parallel SABRE-inspired Depth surrogate), a depth-only surrogate for state-of-the-art SABRE-routing. PSiD preserves SABRE's main routing logic while avoiding routed-circuit materialization, and parallelizes independent randomized trials within each fitness evaluation. Its integration yields PSiD-GA, a two-level parallel framework combining population-level GA evaluation with intra-evaluation surrogate parallelism. Experiments on diverse benchmarks show that PSiD closely estimates SABRE-routing depth while greatly reducing evaluation time. PSiD-GA improves over SABRE in mapping quality and robustness, and achieves up to two orders of magnitude speedup over the original parallel GA.

Keywords: Genetic Algorithms · Qubit Mapping · Parallel Metaheuristics · Quantum Computing · High-Performance Computing · Optimization

1 Introduction

By leveraging properties of quantum mechanics such as superposition and entanglement, quantum computing promises to deliver substantial speedups for a wide range of applications, including search algorithms [3], integer factorization used in cryptography [11], and simulation of physical systems [7]. However, most of these contributions remain theoretical or restricted to quantum simulators, as currently available quantum computers belong to the Noisy Intermediate-Scale Quantum (NISQ) era. NISQ devices feature only a limited number of quantum bits—or *qubits*—and a sparse connectivity between them, with non-negligible error rates that reduce the reliability of their outputs.

To overcome these hardware restrictions, a quantum program—represented as a *quantum circuit*—must undergo a *transpilation* process. Among its multiple steps, the *mapping*—the assignment of logical qubits to physical ones—plays a central role: it is followed by a *routing* step that inserts SWAP gates to satisfy the device's sparse connectivity, and the cost of routing depends critically on

the chosen mapping. Finding the mapping that leads to the least error-prone transformed circuit is known as the *qubit mapping* (or qubit allocation) problem and has been proven NP-hard [12]. It has been addressed by several methods, including the SABRE heuristic [6]—the reference method implemented in IBM's Qiskit transpiler [4]—and specialized metaheuristics such as Genetic Algorithms (GAs). However, GA-based approaches typically suffer from frequent calls to such routing heuristics within their fitness function, leading to substantial execution overheads. In this work, we revisit such approaches by introducing a GA-specific fitness function that accelerates per-solution evaluations, aiming to outperform state-of-the-art heuristics in mapping quality while remaining competitive in execution time.

The contributions of this paper can be summarized as follows:

- We introduce *Parallel SABRE-inspired Depth surrogate* (PSiD), a depth-only variant of Qiskit's SABRE-routing pass designed as a fast objective function for optimization-based qubit mapping methods. PSiD evaluates a candidate mapping at a fraction of SABRE's cost while remaining a faithful estimator of its post-routing depth.
- We integrate PSiD into the parallel GA of Rouzé *et al.* [10], yielding the PSiD-GA framework, which combines two levels of parallelism to amortize the GA's dominant cost across modern multi-core hardware.
- We empirically validate the proposed implementation on a diverse benchmark suite, showing that PSiD-GA outperforms SABRE in mapping quality and robustness while remaining competitive in execution time—thereby addressing the high computational cost limiting previous GA-based approaches.

The remainder of this paper is organized as follows. Section 2 introduces the qubit mapping problem and reviews related work. Section 3 presents the design of PSiD and its integration into the proposed PSiD-GA framework. Section 4 reports our experimental results, including the validation of PSiD, a strong-scaling study, and a comparison of PSiD-GA with SABRE and its parallel-GA predecessor. Finally, Section 5 concludes the study and outlines directions for future research.

2 Background

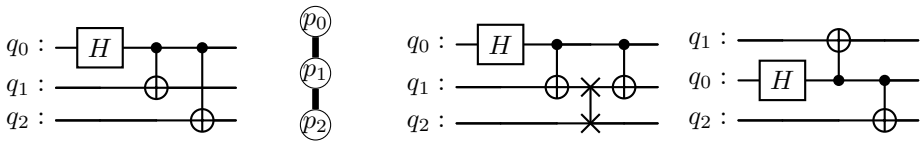
In this section, we first introduce the qubit mapping problem, then review the related work that motivated the present study.

2.1 The Qubit Mapping Problem

Quantum circuits constitute the canonical model of quantum computation, where each horizontal line denotes a *logical qubit* on which gates are applied in left-to-right order (*cf.* Fig. 1a). A gate may target a single qubit (*e.g.*, Hadamard) or a pair of qubits (*e.g.*, the controlled-NOT, or CNOT). Executing such a circuit on a NISQ device introduces two structural limitations: only a small number of

noise-sensitive *physical qubits* are available; and two-qubit gates are restricted to the qubit pairs connected in a fixed coupling graph (*cf.* Fig. 1b). A two-qubit gate whose operands are mapped to non-adjacent physical qubits is therefore not natively executable, and must be enabled by inserting SWAP gates, each of which interchanges the states of two adjacent qubits. Such insertions deepen the circuit and contribute additional sources of error.

These constraints define the *qubit mapping* problem: determining an initial correspondence between the logical qubits of a circuit and the physical qubits of a given NISQ device, with the objective of producing a reliable execution. The mapping step precedes *routing*, which addresses the remaining hardware constraints by inserting SWAP operations so that all two-qubit interactions become executable. While routing strategies attempt to minimize the resulting overhead, their effectiveness strongly depends on the chosen initial mapping. Figure 1 illustrates this on a three-qubit example: starting from the same input circuit (Fig. 1a), two distinct mappings yield noticeably different hardware-compliant variants (Figs. 1c and 1d). We denote a mapping as the ordered list of physical qubits assigned to each logical qubit; for instance, $[1, 0, 2]$ sends q_0 to p_1, q_1 to p_0, and q_2 to p_2. The two routed variants are logically identical, but the latter requires fewer gates and has a lower circuit depth—*i.e.*, the maximum number of gates applied sequentially on any qubit—making it less error-prone and therefore preferable.



(a) Input circuit (b) Coupling (c) Mapping [0,1,2] (d) Mapping [1,0,2]

Fig. 1: Comparison of the input circuit and its routed versions on a linear graph under different qubit mappings.

2.2 Related Work

Qubit mapping has been addressed through both exact approaches [14, 17] and approximate ones, the latter dominating practical use thanks to their scalability and shorter execution times. The most well-established is the SABRE heuristic [6], which tackles both qubit mapping and circuit routing. Its LightSABRE variant [16] is the version currently implemented in Qiskit [4], a leading open-source framework for quantum computing by IBM.

Other contributions include metaheuristic methods such as GAs, most of which rely on SABRE-routing to evaluate candidate mappings. The GA proposed in [1] reports promising results against baseline heuristics, but its evaluation is

limited to a single circuit and its execution time is substantially higher than that of the compared methods. Rouzé *et al.* [10] addressed both issues by parallelizing the GA's evaluation step and extending the benchmark to larger circuits, with a follow-up [9] further extending the approach to a parallel memetic algorithm. Despite these advances, execution time remains considerably higher than that of SABRE.

In this work, we build upon [10] by extending its experimental benchmark and, more importantly, substantially reducing the parallel GA's execution time through a minimalist parallel implementation of the objective function—bringing it within range of SABRE while retaining its mapping-quality advantage.

2.3 The SABRE Heuristic

We close this section with a brief introduction to the SWAP-based BidiREctional (SABRE) algorithm [6] and its enhanced variant LightSABRE [16], which play a central role in our contributions (Sec. 3).

SABRE addresses both qubit mapping (SABRE-mapping) and circuit routing (SABRE-routing). The routing procedure starts by converting the input circuit into a Directed Acyclic Graph (DAG) over its two-qubit gates, where each edge encodes that a gate can only execute once its predecessor has been applied (Fig. 2). The set of gates with no unexecuted predecessors forms the *front layer*; in Fig. 2, this layer starts with g_1 alone.

At each step, two situations arise: (i) if some front-layer gates are executable under the current mapping, they are applied, removed from the layer, and their successors are added; (ii) otherwise, the algorithm inserts a SWAP gate. To guide this choice, SABRE maintains an *extended set* of gates likely to enter the front layer next, and scores each candidate SWAP S by a heuristic function $H(S)$ that combines physical distances over the front and extended sets; the SWAP minimizing H is applied and the mapping is updated accordingly. The process iterates until the front layer is empty, at which point routing is complete.

SABRE-mapping wraps an outer search around routing: starting from a random initial mapping, the circuit is routed forward and then backward (right-to-left), with the mapping updated by the SWAPs inserted in each pass. After a small number of such forward-backward iterations, the final mapping is returned.

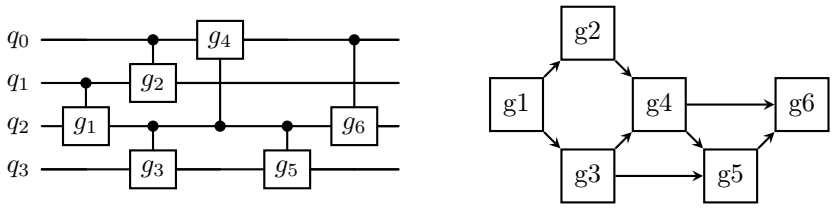


Fig. 2: A 4-qubit quantum circuit simplified to its two-qubit gates only, alongside its dependency DAG.

3 The Proposed Parallel SABRE-inspired Depth Surrogate and Genetic Algorithm Integration

This section presents the design of our parallel GA for qubit mapping. The first subsection details the baseline GA, and the second introduces our main contribution: *Parallel SABRE-inspired Depth surrogate* (PSiD), a more efficient objective function whose integration into the baseline yields the proposed PSiD-GA framework.

3.1 Parallel Genetic Algorithm for Qubit Mapping

Genetic algorithms are a class of optimization algorithms inspired by natural selection [13]. An initial population of P solutions is randomly generated and evolved over a number of generations toward a near-optimal solution. In the qubit mapping context, we adopt the operator choices of [10] for direct comparability; the same operators—excluding the local-search step—appear in the memetic-algorithm extension of [9]. The specific operators are described below.

The population is initially generated by randomly assigning genes to each individual, where a gene represents the mapping of a single qubit. Each generation begins with parent selection via steady-state selection, followed by recombination using a two-point crossover operator. Because crossover can produce infeasible solutions (*i.e.*, multiple logical qubits mapped to the same physical qubit), a repair operator is immediately applied, randomly reassigning one conflicting qubit. Mutation is then applied to the offspring O: each gene has a 0.1 probability of being reassigned to a randomly chosen physical qubit to promote diversity. Finally, offspring fitness is evaluated, and the population is updated using an elitist replacement strategy that retains only the best-performing individuals. The algorithm terminates after a predefined number of generations.

Parallel computing can be incorporated into a GA's structure in several ways [13]. In this work, we adopt *iteration-level parallelism*: the concurrent evaluation of all individuals within a generation. The evaluation step is typically the most time-consuming part of a generation, making it the most critical to accelerate. It is also well-suited for parallelization, since each evaluation is independent and requires no inter-process communication.

Solving the qubit mapping problem with a GA requires defining a fitness function. Ideally, this would be the error rate of the routed circuit, but this quantity is difficult to compute in practice; we therefore use a proxy: the routed-circuit depth. Computing this depth requires a routing algorithm. Among existing options—ranging from basic SWAP-insertion heuristics to state-of-the-art approaches such as SABRE-routing—we follow [10] and adopt SABRE-routing. Our fitness function is therefore defined as the depth of the input circuit after being routed by SABRE.

The GA's workflow is shown in Fig. 3, where blue steps are done in parallel.

This baseline parallel GA, hereafter referred to as **PGA**, suffers from one important limitation: while conceptually straightforward, its practical evaluation

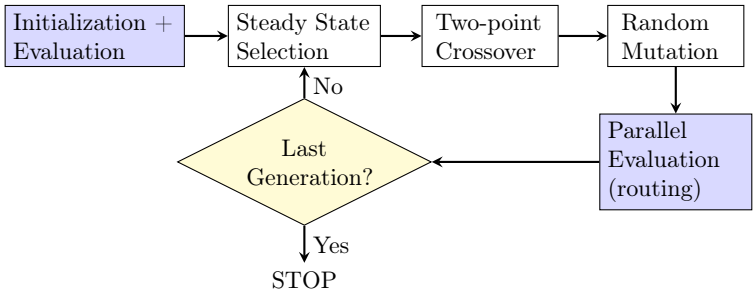


Fig. 3: Workflow of the parallel GA for qubit mapping.

is computationally expensive, as already noted in [10]—invoking a full-featured routing pipeline for each candidate solution introduces significant overhead. This observation motivates the more efficient evaluation strategy introduced in the next subsection.

3.2 Parallel SABRE-inspired Depth Surrogate: A More Efficient Objective Function

To compute the cost of the offspring at each generation, multiple calls to Qiskit's SABRE-routing pass are necessary. However, doing so incurs substantial overhead unrelated to the routing logic itself: each call rebuilds the circuit's DAG, runs the SABRE heuristic, and re-encodes the routed output through a basis-translation pass before the depth can be queried. Across the thousands of evaluations performed during a single GA run, this orchestration cost dominates the GA's wall-clock time, even under parallel evaluation.

We address this by introducing a depth-only variant of SABRE-routing, which we call *Parallel SABRE-inspired Depth surrogate* (PSiD). PSiD retains SABRE-routing's control flow (*cf.* Sec. 2.3): the same front layer, extended set, and SWAP-selection heuristic $H(S)$ are used at each step. The decisive difference is that no routed circuit is ever materialized (*i.e.*, stored in memory). Instead, we maintain an integer array ℓ that records each physical qubit's most recent layer index[1]. Whenever a gate is executed, ℓ is advanced on its operands. If the executed gate is a SWAP, it is treated as three CNOTs—accounting for the standard decomposition of a SWAP into three CNOT basis gates. Once all gates have been routed, the routed depth is read off as the maximum value of this array. Empirically, the depth produced by PSiD closely tracks that of Qiskit's full SABRE pipeline, making it a faithful drop-in replacement for the GA's objective function.

PSiD also exploits an important component of LightSABRE-routing [16]: its best-of-k structure, in which the heuristic is run from k randomized restarts (*trials*) and only the lowest-depth result is retained. Since the k trials are mutually

[1] Not to be confused with the front layer. Here, the layer index of a qubit is the depth at which the most recent gate involving that qubit is placed.

Algorithm 1: Parallel SABRE-inspired Depth surrogate (PSiD)

Input: logical circuit C, coupling graph G, mapping π, trials k
Output: routed depth d^*
$d^* \leftarrow +\infty$
for $t = 1, \ldots, k$ **do in parallel**
  $\pi_t \leftarrow \pi$
  $\ell[q] \leftarrow 0$ for every physical qubit q
  $F \leftarrow$ front layer of C
  while $F \neq \emptyset$ **do**
    if F *contains executable gates under* π_t **then**
      apply every executable gate in F, updating ℓ and advancing F
    else
      $E \leftarrow$ extended set (look-ahead gates beyond F)
      $S^* \leftarrow$ candidate SWAP adjacent to F minimizing $H(S; F, E, \pi_t)$
      apply S^*, update π_t and ℓ
    end
  end
  $d_t \leftarrow \max_q \ell[q]$
  $d^* \leftarrow \min(d^*, d_t)$ `// atomic update`
end
return d^*

independent, they can be executed concurrently with no synchronization apart from the final reduction. Algorithm 1 provides a high-level overview of PSiD.

PSiD's depth-only design comes with one important limitation: since no routed circuit is materialized, gate-type information is lost, and PSiD therefore cannot serve as a faithful estimator of Qiskit's full transpilation pipeline. That pipeline comprises six stages—initialization, mapping, routing, translation, optimization, and scheduling—of which PSiD models only the routing stage and the translation of inserted SWAPs. The initialization stage, being mapping-agnostic, can be applied once as a preprocessing step before the GA loop. The post-routing optimization and scheduling stages, however, fall outside PSiD's scope, and all depth values reported in our experiments accordingly correspond to the pre-optimization, pre-scheduling circuit for every approach. PSiD is not designed to supplant Qiskit's full SABRE pipeline, but rather to serve as a faithful surrogate for optimization-driven settings such as GAs, where the routing step alone is the dominant bottleneck.

In the GA, PSiD substitutes Qiskit's SABRE-routing call as the fitness objective, leading to the proposed variant denoted **PSiD-GA**. Whereas PGA exploits only *iteration-level parallelism*—the concurrent evaluation of the offspring O at each generation—PSiD-GA further introduces *solution-level parallelism* by distributing the k trials within every individual evaluation, raising the effective

degree of parallelism per generation from $|O|$ to $k \times |O|$. Replacing the routing call with a faster yet functionally equivalent counterpart therefore directly translates into PSiD-GA achieving substantial speedups over its PGA counterpart, as confirmed by our subsequent experiments.

4 Experimental Results

4.1 Experimental Setup

In the remainder of this paper, we evaluate the following three qubit mapping implementations:

- **SABRE.** Qiskit's implementation of the SABRE-mapping heuristic, written in Python and Rust.
- **PGA.** The baseline parallel GA from [10] implemented in Python.
- **PSiD-GA.** The proposed parallel GA, implemented using a hybrid approach. The GA is implemented in Python, while PSiD is implemented in C++ and linked through the pybind11 library.

Both GA implementations leverage the pygad [2] Python package. PGA parallelizes evaluation through Python's multiprocessing, whereas PSiD-GA parallelizes both the evaluation loop and PSiD's k trials with OpenMP. To match Qiskit's optimization level 3 (the maximum), PSiD uses $k = 20$ trials per call. All experiments are conducted on an HPC compute node with two 32-core Intel Xeon Platinum 8358 CPUs. Software versions are listed in Table 1.

As benchmarks, we use the six MQTBench [8] circuits used by Rouzé *et al.* [10] (`dj`, `ghz`, and `ghzall` at $n = 80, 120$), targeting the 127-qubit IBM Eagle NISQ device [5], where the `ghzall` instances were generated to match the MQTBench convention. For a more diverse and representative benchmark, we complete the suite with six smaller RevLib [15] circuits of sizes ranging from $n = 19$ to 27, targeting the 27-qubit IBM Falcon NISQ device [5].

The entire code used in this study is publicly available at `https://github.com/anonymized`.

Table 1: Software versions used for the experiments.

name	version	name	version
GCC	13.2.0	Qiskit	1.2.4
Python	3.9.2	pygad	3.6.0
OpenMP	4.5	pybind11	3.0.3

Table 2: Validation and performance evaluation of PSiD.

circuit	n	depth		error (%)	bias (%)	time (ms)		speedup
		SABRE	PSiD			SABRE	PSiD	
add6_196	19	58635	58615	0.62	+0.06	2482	15.1	×164.4
decod_217	21	9675	9746	0.89	+0.65	412	2.6	×159.3
cm150a_210	22	9459	9598	1.85	+1.43	390	2.8	×139.3
arb8_323	24	7934	7912	1.87	−0.05	325	2.5	×130.0
in0_235	26	458659	459357	0.33	+0.21	18626	109	×170.9
sym9_317	27	628	634	4.46	−0.15	40.2	0.5	×80.4
dj_80	80	454	452	3.14	−0.33	67.8	1.3	×52.2
dj_120	120	661	661	3.16	+0.33	96.2	1.8	×53.4
ghz_80	80	902	927	6.70	+1.27	48.0	0.9	×53.3
ghz_120	120	1402	1410	6.88	+0.43	67.3	1.2	×56.1
ghzall_80	80	382	384	3.48	+0.26	50.3	1.1	×45.7
ghzall_120	120	561	567	4.57	+0.96	70.6	1.3	×54.3

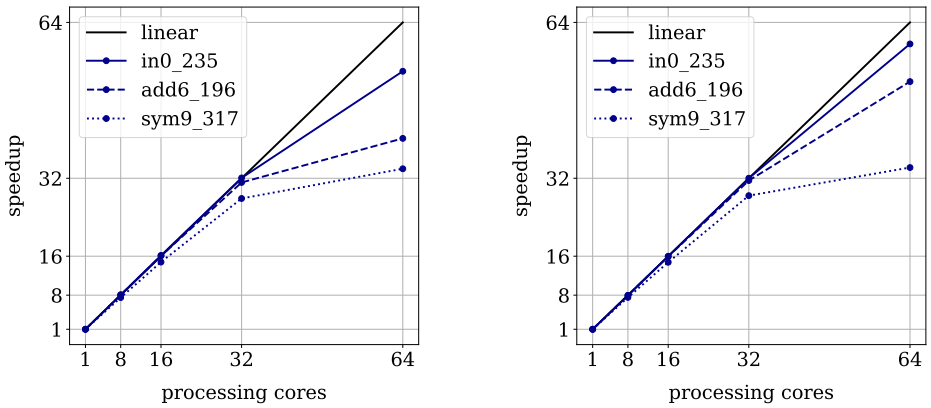
4.2 Empirical Evaluation

We organize the evaluation in three complementary studies: (i) validating PSiD as a faithful but accelerated surrogate for SABRE-routing in isolation from the GA; (ii) a strong-scaling analysis of PSiD-GA's parallel evaluation phase, the dominant cost of every GA generation; and (iii) an end-to-end comparison of PSiD-GA's mapping quality and runtime against the PGA baseline and the SABRE-mapping reference.

We start with the validation of PSiD in isolation from the GA. For each circuit, PSiD and SABRE-routing are executed on 100 randomly generated initial mappings under a fixed seed for reproducibility (both being stochastic by design), with PSiD running on 20 OpenMP threads to match its $k = 20$ trials. Table 2 reports, across the 100 runs, the median post-routing depth obtained by each routine, the average relative error and bias (the latter capturing whether PSiD systematically over- or under-estimates SABRE's depth), and the average wall-clock time (in milliseconds) together with the resulting speedup of PSiD over SABRE. Overall, PSiD emerges as a faithful, low-bias estimator of SABRE-routing depth, while delivering significant performance gains—especially on the deeper circuits—confirming both its efficiency and its suitability within a GA framework.

Next, we conduct the strong-scaling study. The setup mirrors the evaluation phase within a single GA generation: a population of P randomly generated mappings is evaluated in parallel with PSiD ($k = 20$ trials each), totaling $P \times k$ parallel tasks distributed across the OpenMP threads. Each timing measurement is reported as the average of 5 independent runs to mitigate run-to-run system noise. Figure 4 reports the relative speedup as the number of CPU cores increases from 1 to 64, for three representative benchmark circuits at two population sizes:

$P = 100$ (Fig. 4a) and $P = 200$ (Fig. 4b). Results show that the parallel scheme tracks the linear speedup closely up to 32 cores and only departs from it at 64 cores, with the deviation shrinking as the per-circuit workload grows: the challenging `in0_235` instance reaches up to 93% of linear speedup at 64 cores. More broadly, the near-linear scaling demonstrated here makes the proposed parallel scheme an effective lever for shrinking the GA's per-generation cost on modern multi-core hardware, where the fitness evaluation is otherwise the dominant bottleneck.



(a) $P = 100$ (b) $P = 200$

Fig. 4: Strong-scaling of PSiD-GA's parallel evaluation phase on three representative circuits, for two population sizes.

Finally, we compare PSiD-GA against PGA and the SABRE-mapping reference. For a fair comparison, the post-routing depths produced by all three approaches are evaluated using Qiskit's SABRE-routing pass, and each algorithm is run 50 times on every benchmark circuit to mitigate stochasticity from both the GA workflow and the routing heuristic. The GAs use a population of $P = 100$, an offspring of size $|O| = 50$, 30 fixed generations, and leverage 64 CPU cores. Table 3 reports the solution quality of each approach—median depth, standard deviation across runs, and relative depth reduction over SABRE—while Table 4 reports the execution time and the speedup of PSiD-GA over PGA.

As shown in Table 3, PSiD-GA inherits PGA's two key advantages over SABRE: lower median depth and lower run-to-run standard deviation, indicating both better solution quality and greater robustness. Both advantages hold on every benchmark except the `ghz` circuits, a behavior already noted in [10] due to the linear structure of the circuit for which SABRE manages to find an optimal solution. Compared to PGA, PSiD-GA reports slightly larger depths and standard deviations on most circuits; both gaps stem from the same source—the bias of PSiD relative to Qiskit's SABRE (*cf.* Table 2)—since mappings opti-

mal under the approximate objective are not strictly optimal when re-evaluated by the reference pipeline. These gaps are more than offset by the substantial speedups achieved by PSiD-GA over PGA reported in Table 4—approaching two orders of magnitude on the deepest circuits—and by PSiD-GA's ability to beat SABRE at far lower computational cost on dense benchmarks such as `add6_196` and `in0_235`. Overall, these results establish PSiD-GA as a competitive qubit-mapping framework.

Table 3: Median post-routing depth and standard deviation across runs for SABRE-mapping, PGA [10], and PSiD-GA, with the relative depth reduction of PGA and PSiD-GA over SABRE.

circuit	**median depth**			**std**			**reduction (%)**	
	SABRE	**PGA**	**PSiD-GA**	**SABRE**	**PGA**	**PSiD-GA**	**PGA**	**PSiD-GA**
add6_196	58533	57505	57745	322	85.6	147.8	+1.79	+1.76
decod_217	9571	9436	9473	65.4	17.1	30.2	+1.43	+1.03
cm150a_210	9288	9012	9099	139	31.4	49.9	+3.06	+2.08
arb8_323	7723	7504	7547	119	19.7	34.1	+2.92	+2.34
in0_235	457177	454314	455461	946	511	490	+0.63	+0.38
sym9_317	536	517	521	25.5	7.6	8.1	+3.57	+2.88
dj_80	419	341	339	15.5	9.3	9.3	+22.9	+23.4
dj_120	657	527	531	25.0	10.5	9.7	+24.8	+23.8
ghz_80	82	579	592	0	22.8	30.2	−85.8	−86.2
ghz_120	650	974	995	68.5	38.4	36.4	−33.3	−34.7
ghzall_80	310	275	275	13.4	8.2	8.6	+12.9	+12.7
ghzall_120	506	432	435	22.3	9.9	12.8	+17.1	+16.4

5 Conclusions and Perspectives

This paper introduced Parallel SABRE-inspired Depth surrogate (PSiD), an efficient depth-only variant of SABRE-routing, and its integration within a genetic algorithm framework (PSiD-GA) for the qubit mapping problem—a critical step in the transpilation pipeline for NISQ devices. Extensive experiments show that PSiD-GA consistently outperforms the standard SABRE heuristic in both mapping quality and robustness across diverse benchmarks, while maintaining comparable runtimes. Moreover, PSiD-GA achieves up to two orders of magnitude speedup over previous parallel GAs on the largest instances. These results illustrate that carefully designed parallel routing strategies can mitigate the computational overhead of evolutionary approaches—highlighting the broader value of hybridizing domain-specific heuristics with metaheuristic frameworks to balance solution quality and efficiency.

Several directions remain open for future work. The strong scalability of PSiD makes it a natural fit for other optimization algorithms sharing the same

Table 4: Average execution time (in seconds) of SABRE-mapping, PGA [10], and PSiD-GA, with the speedup of PSiD-GA over PGA.

circuit	time (s)			speedup
	SABRE	PGA	PSiD-GA	
add6_196	46.6	643	9.8	×65.6
decod_217	3.5	126	1.9	×66.3
cm150a_210	1.0	118	2.2	×53.6
arb8_323	1.8	103.5	1.9	×54.5
in0_235	292	5015	48.9	×102.6
sym9_317	0.1	31.3	0.8	×39.1
dj_80	0.08	97.5	2.8	×34.8
dj_120	0.1	105.4	5.4	×19.5
ghz_80	17.7	90.6	2.6	×34.8
ghz_120	22.8	96.1	5.1	×18.8
ghzall_80	0.05	91.0	2.7	×33.7
ghzall_120	0.08	95.8	5.2	×18.4

fitness-evaluation bottleneck, such as local search [9]. Within the GA itself, the operators inherited from [10] could be revisited and made more problem-specific—particularly crossover and mutation. Both directions are already under investigation, with promising preliminary results. A further direction is the extension of the depth-only objective to multi-objective criteria that also account for the total gate count or device-specific qubit error rates.

Acknowledgments

Anonymized

References

1. Dahi, Z.A., Chicano, F., Luque, G., Alba, E.: Genetic algorithm for qubits initialisation in noisy intermediate-scale quantum machines: the IBM case study. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1164–1172. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3512290.3528830
2. Gad, A.F.: PyGAD: An intuitive genetic algorithm Python library (2021)
3. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. pp. 212–219. STOC '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866
4. IBM: Qiskit, open source quantum information science kit (2018), https://qiskit.org/

5. IBM Quantum: IBM Quantum processor types (2024), https://docs.quantum.ibm.com/guides/processor-types
6. Li, G., Ding, Y., Xie, Y.: 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. pp. 1001–1014. ASPLOS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3297858.3304023
7. Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.H., Zhou, X.Q., Love, P.J., Aspuru-Guzik, A., O'Brien, J.L.: A variational eigenvalue solver on a photonic quantum processor. Nature Communications **5**(1) (2014). https://doi.org/10.1038/ncomms5213
8. Quetschlich, N., Burgholzer, L., Wille, R.: MQT Bench: Benchmarking software and design automation tools for quantum computing. Quantum **7**(1062), 1–14 (2023)
9. Rouzé, J., Melab, N., Gmys, J., Tuyttens, D.: A parallel memetic algorithm for qubit mapping on noisy intermediate-scale quantum machines. Engineering Applications of Artificial Intelligence **161**, 112081 (2025). https://doi.org/10.1016/j.engappai.2025.112081
10. Rouzé, J., Melab, N., Tuyttens, D.: A parallel genetic algorithm for qubit mapping on noisy intermediate-scale quantum machines. In: Dorronsoro, B., Zagar, M., Talbi, E.G. (eds.) Optimization and Learning. pp. 305–320. Springer Nature Switzerland, Cham (2025)
11. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172
12. Siraichi, M.Y., Santos, V.F.D., Collange, C., Quintão Pereira, F.M.: Qubit allocation. In: Proceedings of the 2018 International Symposium on Code Generation and Optimization. pp. 1–12. CGO 2018, Vienna, Austria (2018). https://doi.org/10.1145/3168822
13. Talbi, E.G.: Metaheuristics: From Design to Implementation. Wiley (2009)
14. Valois, J.P., Helbecque, G., Melab, N.: Efficient and scalable branch-and-bound algorithm for exact qubit allocation. Future Generation Computer Systems **179**, 108342 (2026). https://doi.org/10.1016/j.future.2025.108342
15. Wille, R., Große, D., Teuber, L., Dueck, G.W., Drechsler, R.: RevLib: An online resource for reversible functions and reversible circuits. In: Proceedings of the 38th International Symposium on Multiple Valued Logic. pp. 220–225 (2008)
16. Zou, H., Treinish, M., Hartman, K., Ivrii, A., Lishman, J.: LightSABRE: A lightweight and enhanced SABRE algorithm (2024). https://doi.org/10.48550/arXiv.2409.08368
17. Zulehner, A., Paler, A., Wille, R.: An efficient methodology for mapping quantum circuits to the IBM QX architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(7), 1226–1236 (2019). https://doi.org/10.1109/TCAD.2018.2846658