A Parallel Genetic Algorithm for Qubit Mapping on Noisy Intermediate-Scale Quantum Machines

Jérôme Rouzé, Nouredine Melab, Daniel Tuyttens

A Parallel Genetic Algorithm for Qubit Mapping on Noisy Intermediate-Scale Quantum Machines

Jérôme Rouzé[1], Nouredine Melab[2] and Daniel Tuyttens[1]

[1] Mathematics and Operations Research Department, University of Mons, Belgium
{jerome.rouze,daniel.tuyttens}@umons.ac.be
[2] Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
nouredine.melab@univ-lille.fr

Abstract. Quantum computers are getting increasingly large and available thanks to some major technological advancements, but they remain in the realm of NISQ (Noisy Intermediate Scale Quantum) devices. On such devices, due to the limited connectivity of the physical qubits, most quantum circuit-based programs cannot be executed without transpilation. This latter includes an important step, referred to as qubit mapping, which consists in converting the quantum circuit into another one which best matches the graph of physical qubits taking into account its limited connectivity constraint.
In this paper, we propose a Parallel Genetic Algorithm to Qubit Mapping (PGA-QM). The challenge is to minimize the depth of the transformed circuit and the execution time and error rate consequently. PGA-QM has been experimented using various medium-to-large scale circuit benchmarks. It is compared against the SABRE heuristic currently implemented in Qiskit, the IBM's library for quantum computing. The reported results show that PGA-QM can provide better solutions and with better consistency than its counterpart while parallelism greatly reduces its execution time during the transpilation.

1 Introduction

In recent years, quantum computers have become increasingly available, including a growing number of qubits. Using the principles of quantum physics such as superposition and entanglement, these systems are promised to offer significant speedup over classical ones for a variety of applications. While Grover's search algorithm [9] and Shor's integer factorization [15] can be cited as two significant historical quantum-related contributions, other notable advancements include quantum combinatorial optimization [5,7], quantum machine learning [12], and so forth [10].

However, such promising results, often referred to as the quantum advantage, cannot be easily achieved without quantum error correction, which would require much more qubits than currently supplied. Actually, current quantum computers are noisy, and the results of the computations they return have a non-zero chance of being wrong. To make use of current quantum devices, one needs to minimize the error rate so that the returned results can be trusted. In addition, like in classical computing, a quantum program (circuit in this paper) needs to be complied (or transpiled) to be executable on a device. This transpilation consists in several steps, including gate decomposition and mapping of the quantum (or logical) circuit on a graph of physical qubits composing a quantum machine.

This mapping step has a great impact on the error rate of the executed quantum circuit. In this paper, we investigate this circuit mapping problem referred to in the literature as qubit allocation problem [16], qubit initialization problem [6] or qubit mapping problem [11,4]. Depending on how the logical qubits of the circuit are mapped to the physical ones in the hardware, one may need to add a lot of gates to the circuit to match the hardware connectivity. That will increase the depth of the circuit, increasing the likelihood of decoherence, thus increasing the error rate of the circuit. Given that gates are noisy, adding more gates increases the error rate. For these two reasons, finding a correct mapping can greatly limit the error rate.

The main contributions of this paper are the following:

- We propose a parallel genetic algorithm the solve the qubit mapping problem (PGA-QM). The parallel contribution is done at the evaluation step of the GA as computing the cost of each individual is costly (and doable in parallel).

- We compared the performance of this PGA-QM using various benchmarks, against the currently used SABRE [11] heuristic supplied in IBM's Qiskit [13] library for quantum computing. The reported results highlight the ability of the PGA-QM to outperform SABRE and find better mappings, albeit at a higher computational cost. The added parallelism limits the computational expensiveness of the genetic algorithm.

The rest of this paper is organized as follows. Section 2 starts with a brief introduction to quantum computing followed by a proper formulation of the qubit mapping problem and some related works. In Section 3, the proposed parallel GA approach in presented. The experimental protocol and benchmarks are described in Section 4 and the results are discussed in Section 5. Finally, a conclusion is drawn in Section 6 with some future works.

2 Background, Problem Formulation and Related Works

In this section, we introduce the basic concepts of quantum computing necessary to understand the qubit mapping problem. Then, a proper problem formulation is given. Lastly, related works are presented.

2.1 Background

To introduce the problem at hand in this paper, let us give a brief introduction to the main concepts of quantum computing.

Quantum bits While traditional computing relies on bits as the unit of information, quantum computing relies on quantum bits or qubits. Based on the principles of quantum mechanics, a qubit can be a superposition of the two basis state denoted $|0\rangle$ and $|1\rangle$. That is, bits are deterministic being in either one of those states while qubits are in a state $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ can be normalized such that $|\alpha|^2 + |\beta|^2 = 1$. From a theoretical point of view, the basis state $|0\rangle$ and $|1\rangle$ can be seen as a basis of a vector space, while α and β are the coefficients of the vector $|\Psi\rangle$ in said basis.

Quantum gates An operation on one or more qubit(s) is a quantum gate, similarly to logical gates in classical computing. Mathematically, quantum gates are unitary operations that transform the current quantum state of the system into another. Being unitary operations, they are also reversible. For example, the Hadamard gate, usually denoted as H is a very commonly used single-qubit gate creating a superposition state. The Control-NOT, or CNOT, gate is a two-qubit gate that creates entanglement. It involves a control qubit and a target one applying the NOT operation to the target if the control is in the state $|1\rangle$, else leaving the target unchanged. It has been proven in [3] that the set of all one-qubit gates and the CNOT gate form a universal set of gates. This means that any quantum operation can be decomposed into a sequence of one-qubit and CNOT gates. Actually, only a few carefully selected one-qubit gates and CNOT ones are enough to form a universal set. For that reason, one can consider the CNOT gate as the only multi-qubit gate without loss of generality.

Measurement The only operation that is not a quantum gate is the measurement. Measurement means observing in which state one or more qubit(s) is (are). With the normalization of α and β mentioned before, $|\alpha|^2$ and $|\beta|^2$ are the probabilities of measuring the states $|0\rangle$ and $|1\rangle$ respectively. Due to the properties of quantum mechanics, this measurement operation changes the quantum state in a destructive way[3], meaning that this operation is not reversible and therefore not unitary. Measurement is the way to obtain some results, usually at the end of the computation.

Quantum circuits One of the most commonly used quantum programming paradigms is based on quantum circuits. A quantum circuit represents the sequence of gates applied to its qubit(s). Figure 1 illustrates a four-qubit circuit made of H gates and CNOTs, followed by measurements. On such a representation of a quantum circuit, each line represents a qubit, and the list of gates

[3] One says that the quantum state collapses to the measured state.

on that same line is the list of operations sequentially applied to that qubit.

NISQ Computers In this paper, Noisy Intermediate Scale Quantum (NISQ) computers are considered. On these systems, quantum gates and measurements are noisy, meaning there is a non-zero chance that a gate is wrongly applied or that a measurement does not return the correct state. They also present a limited number of qubits ($< 10^3$) that are scarcely interconnected (see Figure 2). In theory, when designing a quantum circuit, the CNOT(q_i, q_j) between two qubits $q_i \neq q_j$ can always be applied. However, on NISQ hardware, q_i and q_j need to be connected, which is a strong restriction given the scarcity of the connectivity graph. Finally, on these hardware, a limited set of gates is feasible in practice.

Circuit mapping To get around these restrictions, a quantum circuit must be transformed into an equivalent circuit to match the hardware requirement. Gates are decomposed into the basis gates of the system, and extra SWAP operations are added to match the connectivity. A SWAP gate is actually made of three consecutive CNOT gates, and exchanges the quantum states of two qubits. These transformations, and especially the added SWAPs heavily depend on the one-to-one mapping between abstract circuit qubits and physical ones illustrated in Figure 3. Finding the mapping leading to the least noisy circuit is known as the Qubit Mapping Problem.

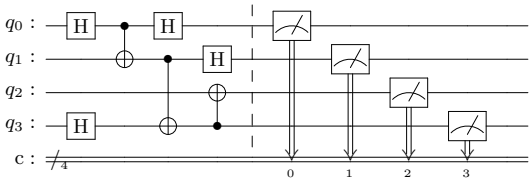


Fig. 1. A 4-qubit quantum circuit made of a some Hadamard and CNOT gates, a barrier (dashed line) followed by measurements.

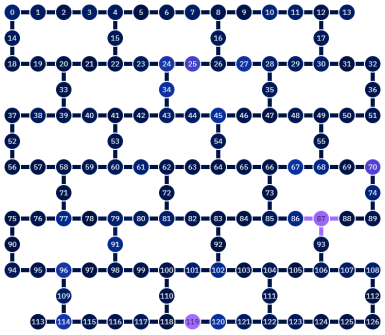


Fig. 2. IBMQ Washington's connectivity graph

2.2 Problem formulation

Formally speaking, the qubit mapping problem can be formulated as follows. The data include a circuit of d virtual qubits $Q = \{q_0, \ldots, q_{d-1}\}$ and a graph representing the connectivity of a physical machine made of $m \geqslant d$ qubits $P = \{p_0, \ldots, p_{m-1}\}$. The objective is to find the best mapping $\pi : Q \to P$, which can be simply represented by an ordered list of d different integers in $\{0, \ldots, m-1\}$. For example, $[1, 3, 0]$ designates the mapping $\pi(q_0) = p_1$, $\pi(q_1) = p_3$ and $\pi(q_2) = p_0$. A solution is then made of d ordered integers among m, leading to $\dfrac{m!}{(m-d)!}$ possible mappings in total. The problem can also be considered as a partial permutation problem.

To each mapping is associated a cost, which is ideally the error rate of the resulting transpiled circuit. However, the error rate is hard to compute. For that reason, it is common to use other metrics that influence the error rate instead. In this paper, we consider the depth of the circuit as the cost function. Essentially, our cost function takes a mapping, i.e. a list of d different integers, and returns the depth of the transpiled circuit to fit the mapping. The *depth* of a quantum circuit is defined as the largest number of gates the quantum device has to execute sequentially to run the whole quantum program. Another way to view this is to represent the quantum circuit by regrouping the gates that can be applied simultaneously in one layer. The depth is then the number of layers.

The depth is chosen among other possible choices, namely, the total cost of circuit transformations as defined in [16] or the number of additional gates [11]. Let us justify the choice made here.

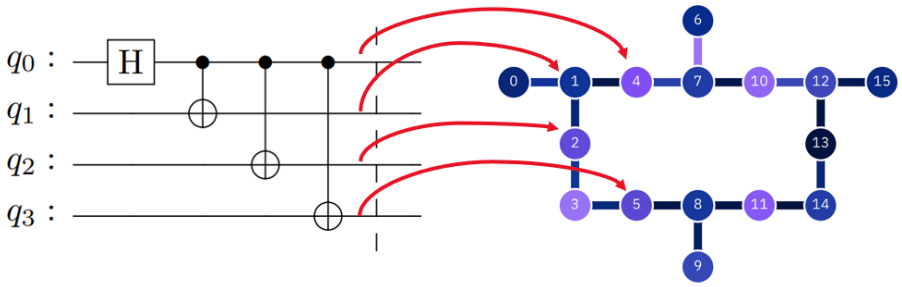


Fig. 3. Visual representation of a mapping π of a circuit on the connectivity graph of some quantum device. Here, $\pi(q_0, q_1, q_2, q_3) = (p_4, p_1, p_2, p_5)$, which can be simply represented by $[4, 1, 2, 5]$.

Recall that the goal is to generate a resulting circuit with the lowest possible error rate. This error rate depends not only on the individual gate error rates but also on the circuit's execution time, due to decoherence. Although negligible at our scale, as it takes a microsecond at worst, longer execution times increase the likelihood of decoherence, leading to an inaccurate quantum state of the outcome. Minimizing the circuit depth reduces the execution time since each layer is executed in parallel, which consequently diminishes the risk of decoherence. A lower depth also translates to fewer SWAP gates as they increase the circuit depth by 3. Moreover, since SWAP gates have a high error rate[4], their number has to be minimized. Hence, a reduced depth addresses both gate-related errors and decoherence likelihood.

The other previously mentioned choices primarily focus on minimizing added CNOT gates to lower the circuit error rate given their higher individual error rates compared to 1-qubit gates ($7.524e^{-3}$ for CNOT, $2.332e^{-4}$ for 1-qubit gates)[5]. However, these approaches do not directly reduce circuit depth, and consequently, the risk of decoherence.

2.3 Related works

The qubit mapping problem has been tackled using different approaches in the recent literature. In [16], an exact algorithm is proposed, but its complexity is too large to be applied to practical problems. A heuristic search is also proposed to deal with this issue. However, the formulation of the problem is quite different as the authors considered unidirectional CNOT, meaning that if the CNOT(q_i, q_j) is feasible on the hardware, CNOT(q_j, q_i) is not. To be exact, it is feasible but demands a "reversal" transformation. Unidirectional CNOTs were the norm at the time the paper was written (in 2018). On current quantum systems, CNOTs are bidirectional, so the added difficulty of unidirectionality is no more.

Other heuristic approaches to this problem have been studied. For instance, the SABRE (SWAP-based BidiRectional) heuristic search [11] is currently implemented in Qiskit [13], a Python library developed by IBM for quantum computing. This heuristic approach has been developed with the goal of being efficient while remaining fast to run. This is achieved through greatly reducing the search space and an efficient initial mapping finder. Indeed, heuristic approaches in the literature have shown to be significantly dependent on the initial mapping they start with.

Duostra (Dual-source Dijkstra) [4] is another fairly recent heuristic contribution designed with large circuits in mind. Their result compared fairly well with SABRE, but the article mentions using the optimization level 0 of Qiskit, i.e. the lowest one.

Metaheuristic approaches have also been developed, and coupled with heuristic search operator. The mapping is found using the metaheuristic while a heuristic search adds the necessary gates to the circuit according to the previously found mapping. Indeed, in the literature, heuristics often try to simultaneously find a mapping and, given that mapping, they find the necessary SWAPs to be performed on the circuit. On the contrary, metaheuristic approaches focus on finding the mapping, leaving the task of adding gates to a heuristic operator. Namely [17] proposed a simulated annealing algorithm to search for the mapping as well as as specific look-ahead heuristic search to add the necessary SWAPs.

[4] Recall that SWAP gates are made of three consecutive CNOTs, which have high error rate.
[5] https://quantum-computing.ibm.com/

A genetic algorithm has also been developed in [6]. In this work, the GA finds a good mapping and the SABRE heuristic adds the necessary gates to match the requirement. Our approach shares this principle but is actually quite different. While [6] focuses on one type of circuit of different sizes on different hardware, our study focuses on testing different large scale circuits on one hardware. We also study the potential improvement parallel computing could provide. Indeed, [6] shows the promising results of the genetic algorithm but the execution gets much longer than the other approaches, and our contribution aims to lower that time.

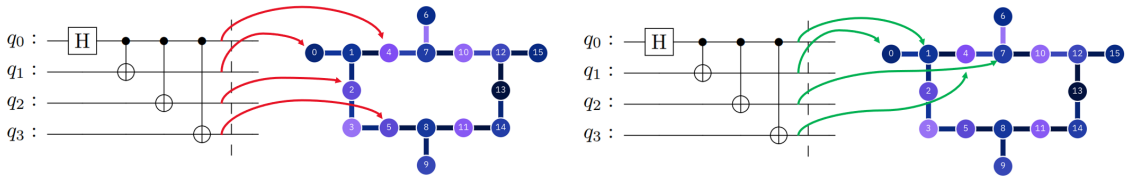
Another completely different approach using Deep Neural Network (DNN) has been developed in [1]. In this paper, the qubit mapping problem is modeled as a classification task instead of an optimization problem. An important contribution of this work is that the execution time does not increase with the depth of the input circuit. However, the study was conducted on a 5-qubit hardware and fairly limited circuit sizes. Moreover, generating data to train the DNN for other architectures relies on an additional heuristic search.

3 The Proposed Parallel Genetic Algorithm : PGA-QM

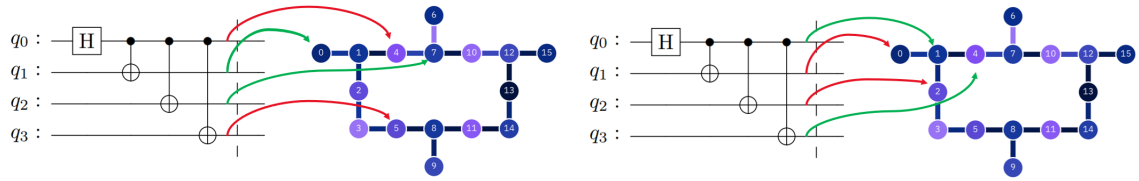
GAs are a type of optimization algorithms inspired by nature, where a population of initially randomly selected individuals evolves throughout a number of generations to find a near-optimal solution to the problem. A pseudo-code of the parallel GA is given in Algorithm 1 and described below.

The population P is initialized with randomly selected genes for each individual. In our application, a gene is simply the mapping of one qubit, i.e. there are d genes which simply consist of one integer each. A generation starts with the selection of parents, which consists of randomly selecting individuals that will be subsequently recombined by a two-point crossover operator (see Figure 4). A mutation operator is then applied to the newly created population $P2$, mimicking natural gene mutations. Here, each gene has a 0.1 probability of being changed to another randomly chosen value. Finally, the cost of each individual of $P2$ is evaluated and based on that evaluation, the population is updated for the next generation. Here, an elitist replacement is considered, that is, only the best individuals are kept. Note that an early stopping criterion may be used, in particular, if for several generations, the best solution has not been improved. In this paper, we considered either no early stopping criterion or a stop after 10 generations without improvement.

Other variations of parameters (crossover type, mutation probability) have been tested to try and get the best possible results, but no significant differences have been observed.



(a) Two parent mappings : $[4, 0, 2, 5]$ (left) and $[1, 0, 7, 4]$ (right)



(b) Their two offspring mappings after crossover : $[4, 0, 7, 5]$ (left) and $[1, 0, 2, 4]$ (right)

Fig. 4. Illustration of the two-point crossover operator

Algorithm 1: Pseudo-code of a parallel genetic algorithm

```
P ← Initialize_Population(m, d, pop_size) ;
v ← Parallel_Evaluation(P);
for i=0, ..., N do
    S ← Random_Parents_Selection(P, nb_new_individuals);
    P2 ← Two_Point_Crossover(S);
    P2 ← Random_Mutation(P2);
    v2 ← Parallel_Evaluation(P2);
    P, v ← Elitist_Replacement(P, P2, v, v2);
    if Early_Stopping_Criterion (v,v2) then
        break
    end
end
```

To select the best individuals for the next generation, one needs to know the cost of each individual, i.e. the depth of the quantum circuit transpiled to match the mapping. This evaluation phase is done in parallel and can significantly reduce the time required for this step, as shown in Section 5.4.

4 Performance Evaluation

4.1 Quantum hardware and circuits

In this section, we describe the benchmarks used in the experiments as well as the experimental protocol and goals. The operation that transforms the abstract circuit into one that fits the hardware requirements is known as the transpilation. Qiskit's transpilation function relies on six steps[6] : initialization, layout, routing, translation, optimization and scheduling. The layout step is the one responsible to find the best mapping, and our proposed PGA-QM aims to replace that step, leaving the other steps unchanged for a fair comparison[7].

An instance of the problem is partly defined by the architecture of the used hardware. Given our aim to consider large circuits, and the available resources built-in within Qiskit, we focused our study on a single architecture, namely, the IBMQ Washington system, whose connectivity graph is given in Figure 2. On such graph, each vertex designates one qubit, and an edge is present between two vertices if a CNOT gate can be applied between these two qubits.

The other element to choose to get one instance of the qubit mapping problem is the quantum circuit to efficiently and effectively deal with. We focused our study on using large circuits of 80 and 120 qubits. These sizes were not chosen randomly. Given that the largest quantum hardware currently provided in Qiskit is, at the time these experiments are performed, made of 127 qubits, we considered the mapping of a "medium-sized" circuit ($\sim 2/3$ of all 127 available qubits are used) and a "full" one ($\sim$ all 127 available qubits are used). A medium mapping problem leads to a partial permutation problem while a full one leads to a true permutation problem (partial permutation problem with $d = m$).

We considered three different quantum circuits, two of which being taken from [14], the Greenberger–Horne–Zeilinger (GHZ) and Deutsch-Jozsa (DJ) circuits. The third one we used will be referred to as GHZALL. Both GHZ and GHZALL lead to the same quantum state at the end, precisely a state where qubits are either all in state $|0\rangle$ or all in state $|1\rangle$, with probability 0.5 for each state. However, they achieve this state through two different means. GHZALL connects all qubits to the same one through CNOT gates while GHZ makes a cascade of CNOT connections as shown in Figure 5. As we will see in Section 5, this leads to very different results.

[6] https://docs.quantum.ibm.com/api/qiskit/transpiler
[7] In this paper, we worked using Qiskit 0.43.3

The three circuits studied have been chosen for their scalability. Actually, since we aimed to study large circuits, we considered those from [14]. The scalability of the circuits limits the type of circuits one can use as not all circuits can be extended to large numbers of qubits.

(a) GHZALL with 3 qubits **(b)** GHZ with 3 qubits

Fig. 5. GHZALL and GHZ circuits

4.2 Objectives of the performance evaluation

The first goal is to evaluate the efficiency and effectiveness of the PGA-QM compared to SABRE. Given the stochastic nature of the algorithms, each execution is done 30 times and the median of the best results found was considered. The median is favored over the average since it is not as sensible to extreme values.

To evaluate the efficiency of our approach compared to SABRE, we considered the execution times of both approaches and their ratio. As shown in Table 1, three variants of the PGA-QM are considered varying their three parameters (population size, number of offsprings per generation and maximum number of generations). These three variants have also been tested with two different stopping criteria: either all the generations are run, or the algorithm is allowed to stop earlier if the best solution did not evolve for 10 consecutive generations. The idea behind those various sets of parameters was to evaluate how impactful they are on both the execution time and quality of the solutions.

A third comparison one can make is based on the robustness of the approaches. Given their stochastic nature, they do not always lead to the same result from one execution to another one. To study the robustness, we considered two statistical metrics: the Inter Quartile Range (IQR) and Median Average Deviation (MAD). The lower these two values are, the less scattered the results are and the more robust the algorithm is.

Finally, we studied the speedup provided by the parallel computing part of the PGA-QM, questioning how much the parallelization actually improved the execution time and how does it scale.

Table 1. Different set of parameters of the GAs.

	# Individuals	# Offsprings per generation	# Total generation
PGA 1	40	20	30
PGA 2	30	20	30
PGA 3	20	15	35

In this paper, we used the parallel GA implementation provided in the PYGAD [8] Python library, and compared its performances with SABRE heuristic [11] currently used in Qiskit. Experiments were run on the GRID5000 testbed [2], using a processing node of 2 AMD EPYC 7301 CPUs. Their characteristics are the following: 16 cores/CPU, 2.2GHz base frequency, 64MB total L3 memory. All codes are available at https://github.com/Jrouze/PGA-QM.

5 Results and Discussion

In this section, we report some experimental results and their discussion. Note that in the following tables the stop10 rows refer to the variants of PGA-QM with an early stop after 10 generations

without improvement while the others refer to the variants without an early stopping criterion.

5.1 Depth analysis

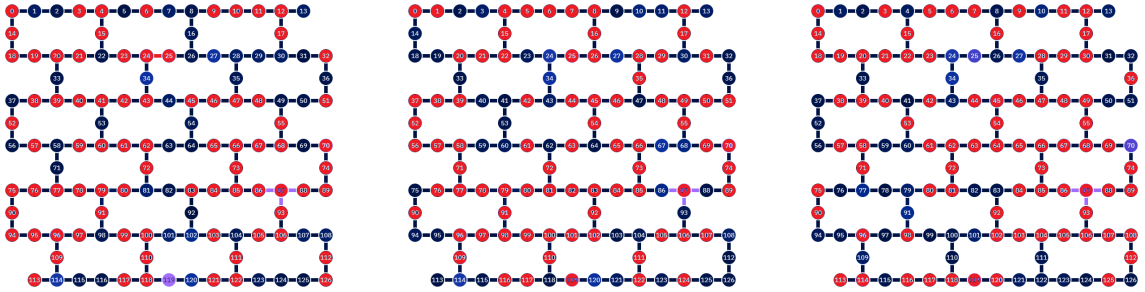
Table 2. Median best found depth for all studied quantum circuits and algorithms

Solver	GHZALL 80	GHZALL 120	DJ 80	DJ 120	GHZ 80	GHZ 120
PGA1	292	**446.5**	**362**	**564.5**	790	1335.5
PGA2	**290.5**	451.5	363.5	565	797	1346
PGA3	292	457.5	365	569.5	814.5	1340
SABRE	344	537.5	389.5	618.5	**83**	**844**
PGA1 stop10	292.5	459.5	370	568	808.5	1326
PGA2 stop10	297	462	367	568	803.5	1355.5
PGA3 stop10	298	465	375.5	570.5	814	1356.5

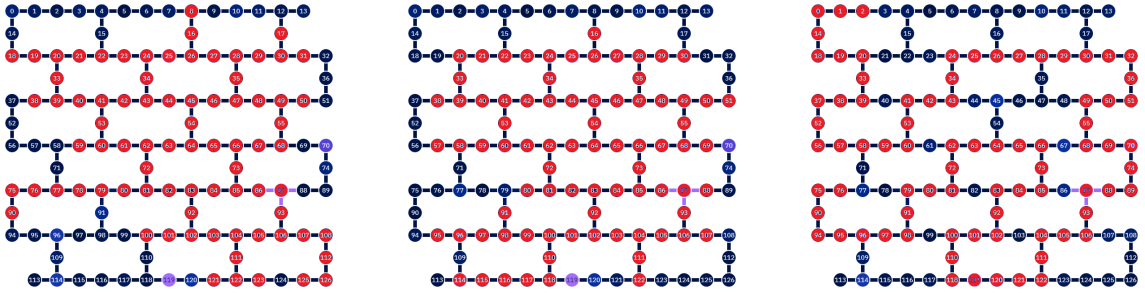
Our first analysis is the study of the best mapping each algorithm has found. According to Table 2 one can observe that the PGA-QM variants can outperform SABRE and find a better mapping, but not for every type of quantum circuit. Indeed, the PGA-QM variants found better solutions for the GHZALL and DJ problems, but a worse one for the GHZ instances. One may also notice that the PGA-QM variants present rather similar results. That can be easily explained by the fact that they all rely on the same operators at each step. Recall that the differences between the three variants of PGA-QM are the population sizes and numbers of generations. The similarity in terms of best result found by each of these variants leads us to conclude that one can lower those parameters to improve the computational time (see next section) while maintaining the quality of the found solution. A study of the best parameters of the GAs (crossover, selection and mutation operators) could likely lead to improving those results.

One thing we have yet to mention is the differences between PGA-QM variant and SABRE and why one is better than the other depending on the circuit. To explain the differences, we studied the solutions they returned. Figure 6 illustrates the connectivity graph of the used hardware, where the red-colored qubits are the ones selected to be part of the solution found by either PGA1 or SABRE for each 80-qubit problems. One can notice that the PGA-QM's solutions are scattered while SABRE ones are connected subgraphs of the connectivity graph. This is due to the SABRE routine that actively searches for connected solutions. It seems that such solutions are more adapted to the GHZ instances. Therefore, the PGA-QM struggles to outperform SABRE for those instances because it does not enforce connected solutions. However, the GHZALL and DJ instances seem to favor scattered solutions and therefore, the PGA-QM finds better solutions for those instances. The GHZALL circuit requires that all qubits are connected to one particular qubit (see Figure 5). That means there is a CNOT between qubit j and qubit 0 for all $j > 0$ while the GHZ circuit does not present such "overloaded" qubit. Our assumption to explain why the GHZALL seems to prefer scattered solutions is that the overloaded qubit is easier to connect to all the others in a scattered mapping, i.e. it required fewer gates to do so. On the other hand, for the GHZ circuits, a connected mapping works best.

It is especially true for the GHZ 80-qubit circuit since the displayed solution is the global minimum: a 80-qubit linear subgraph is found. For that particular instance, the PGA-QM leads to much worse result (10 times SABRE's one). The reason is, for that instance, there actually is a solution that requires no circuit transformation and SABRE is able to find that solution because it looks for it first. Our metaheuristic approach struggles to find such a perfect solution. Our understanding of this is that the PGA-QM explores the search space reasonably well and finds a promising region but struggles to find the local minimum of said region. An hybridization of the PGA-QM with a local search operator could likely help in that regard.



(a) 80-qubit GHZALL — PGA1 **(b)** 80-qubit DJ — PGA1 **(c)** 80-qubit GHZ — PGA1



(d) 80-qubit GHZALL — SABRE **(e)** 80-qubit DJ — SABRE **(f)** 80-qubit GHZ — SABRE

Fig. 6. Selected qubits of the solutions found (in red) by either GA1 (a) to (c) or SABRE (d) to (f) of for each 80 qubits problem.

5.2 Time analysis

From a computational time point of view, the execution of the PGA-QM is up to 65 times slower than SABRE for the larger (120-qubit) circuits (see Table 3). However, one can notice that reducing the population size and total number of generations with the early stopping criteria do not lead to significant worsening of the quality of the produced solutions while it greatly decreases the execution time. In particular, one can get similar results with a PGA-QM just 35 times slower than SABRE in the worst studied case. It is still a lot, but the execution time remains reasonable (a few minutes), and with finer tuned parameters, these results could likely be further improved. Note that in this section, we only compared PGA-QM with enough parallel threads to compute the cost of all individuals in parallel. The conclusion one can draw up here is that PGA-QM can outperform SABRE and find better mappings, and despite being quite slower, remains computationally acceptable.

5.3 Robustness analysis

Another comparison that can be made concerns the robustness of both approaches, defined as their sensibility to randomness. An algorithm is said to be more robust that another one if several executions return similar results despite stochastic operations. Looking at Table 4, one can notice that the PGA-QM variants are more robust. It is an important factor to take into account because a high sensibility to randomness means one can get an “unlucky” bad mapping, leading to a higher error rate of the corresponding quantum program. PGA-QM tends to fix that by being more robust. One can also notice that all PGA-QM variants are more robust than SABRE, but our fastest variant (PGA3 with an early stopping criterion of 10 generations without improvement) is less robust than our slowest one (PGA1 with a stopping after 30 generations). While the median result of both sets of parameters is very similar (see Section 5.1), the results with PGA3 are more split around that median, meaning that the time saved by the faster versions is lost dealing with robustness. However PGA3, even with early stopping after 10 generations without improvement, remains more robust than SABRE. Regarding the robustness scale, PGAs exhibit a better performance.

Table 3. Median time and Ratio (=PGA time/SABRE time) for all studied circuits and algorithms

Solver	GHZALL 80		GHZALL 120		DJ 80	
	Time (s)	Ratio	Time (s)	Ratio	Time (s)	Ratio
PGA1	182.841	49.074	285.422	64.203	187.749	48.690
PGA2	143.695	38.568	225.280	50.674	133.018	34.496
PGA3	147.355	39.550	232.732	52.351	136.895	35.502
SABRE	3.725		4.445		3.856	
PGA1 stop10	110.17	29.570	163.822	36.850	98.008	25.417
PGA2 stop10	78.825	21.156	111.972	25.187	**68.503**	**17.765**
PGA3 stop10	**72.459**	**19.448**	**98.426**	**22.140**	71.798	18.619
Solver	DJ 120		GHZ 80		GHZ 120	
	Time (s)	Ratio	Time (s)	Ratio	Time (s)	Ratio
PGA1	289.968	61.093	339.5609	1.164	588.809	13.139
PGA2	212.788	44.832	274.653	0.941	496.930	11.089
PGA3	212.439	44.759	284.734	0.976	508.984	11.358
SABRE	4.746		291.703		44.812	
PGA1 stop10	151.263	31.869	189.291	0.648	357.661	7.981
PGA2 stop10	115.103	24.251	**137.801**	**0.472**	290.116	6.473
PGA3 stop10	**96.942**	**20.425**	152.789	0.523	**248.505**	**5.545**

Table 4. Inter Quartile Range and Median Average Difference for all studied circuits and algorithms

Solver	GHZALL 80		GHZALL 120		DJ 80		DJ 120		GHZ 80		GHZ 120	
	IQR	MAD	IQR	MAD	IQR	MAD	IQR	MAD	IQR	MAD	IQR	MAD
PGA1	11.5	5.5	14.75	8.5	8.75	6	**12**	**6**	**26.75**	16.5	49.25	27
PGA2	**9.75**	**4.5**	16.5	7.5	**6.75**	**3.5**	17.75	8.5	45	25	**35**	**19.5**
PGA3	10.25	5	14	7.5	10.25	5.5	15.25	8	32.75	17.5	57.5	34.5
SABRE	21	9	27.5	14.5	22.5	11	27.5	15	0	0	63	34
PGA1 stop10	9.75	5	18.5	9.5	13.75	6.5	22.5	12	27.25	**13.5**	38.5	**19.5**
PGA2 stop10	16.5	7.5	13	**7**	16.5	9	15.75	6.5	43	22	55.5	24.5
PGA3 stop10	15.5	8.5	**11.75**	**7**	13.75	7	25.75		32.75	18	49.25	24.5

5.4 Parallel scalability analysis

Lastly, let us discuss the scalability obtained with the parallelization of the PGA-QM for two different quantum circuits, the 80-qubit GHZ and the 40-qubit Quantum Fourier Transform (QFT) [14]. As one can notice in Figure 7, the speedup for GHZ reaches only 12.5 over 20 parallel threads. The two reasons behind this limited scalability are the following. First, the GA is not entirely parallel. As only the evaluation phase is done in parallel, the serial part negatively impacts the speedup. Secondly, the evaluation of the cost function is quite fast for the GHZ circuit (∼3 seconds). Therefore, parallel evaluations speed up the execution moderately as the serial parts remain important. Another important factor is the irregularity of the application. Given that the parallelization is done using the Master-Worker model, the workload is likely to be poorly balanced because of the irregularity of the application.

The QFT circuit is included as an additional problem instance to confirm the irregularity of the application. While using fewer qubits, this circuit is made of a lot more gates (80 for 80-qubit GHZ *versus* 840 for 40-qubit QFT), making the cost function more computationally expensive (∼40 seconds). Actually we can observe on the right side of Figure 7, the speedup is much better, reaching 16 over 20 parallel threads. Indeed, with an objective function more computationally expensive, the parallel part becomes significantly more important than the serial one, leading to a better speedup. This result indicates that the parallel GA is better suited for larger circuits (with many gates).

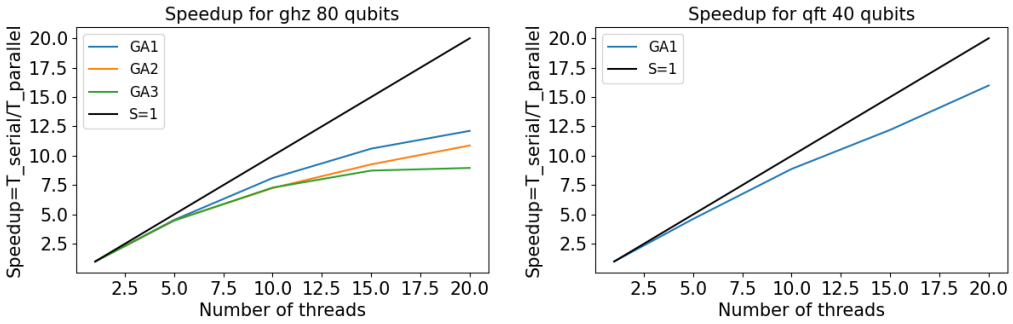


Fig. 7. Speedup for the PGA-QM for either the GHZ (left figure) or QFT (right figure).

6 Conclusions and Future Works

In this paper, we have proposed a Parallel Genetic Algorithm to solve the Qubit Mapping problem (PGA-QM). This latter consists in mapping a quantum circuit on a graph of physical qubits composing a NISQ machine. This mapping is an important step in the transpilation of circuit-based quantum programs in NISQ computers. The objective is to transform the quantum circuit into another one which best matches the graph of physical qubits taking into account its limited connectivity constraint. The challenge is therefore to minimize the depth of the transformed circuit and the execution time and error rate consequently. The parallel GA has been integrated into the IBM Qiskit framework, experimented and compared to the SABRE heuristic provided in this latter. Medium-to-large circuits are considered as benchmarks in the experiments.

The reported results show that our proposed PGA-QM outperforms SABRE on some circuits in terms of error rate, while it is quite slower. Parallelism played a major role in speeding up the execution. The parallel scalability depends on the circuit and its size (number of gates). In addition, while for medium circuits, the parallel GA does not outperform SABRE, it is always more robust, meaning it more often leads to good mappings while it is not as heavily impacted by unlucky draws.

In the future, we plan to combine PGA-QM with single-solution metaheuristics in a two-level approach. At the high level, the hybridization will consist in applying a multi-start local search to the final population produced by the PGA-QM parallel algorithm. At a low level, a local search will be used as a mutation operator in the GA. A second perspective of this work will consist in extending the problem formulation to a multi-objective one considering additional cost functions than the circuit depth.

References

1. Acampora, G., Schiattarella, R.: Deep neural networks for quantum circuit mapping. Neural Comput. Appl. **33**(20), 13723–13743 (oct 2021). https://doi.org/10.1007/s00521-021-06009-3, https://doi.org/10.1007/s00521-021-06009-3
2. Balouek, Daniel, e.a.: Adding virtualization capabilities to the Grid'5000 testbed. In: Ivanov, I.I., van Sinderen, M., Leymann, F., Shan, T. (eds.) Cloud Computing and Services Science, Communications in Computer and Information Science, vol. 367, pp. 3–20. Springer International Publishing (2013). https://doi.org/10.1007/978-3-319-04519-1_1
3. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Physical Review A **52**(5), 3457–3467 (nov 1995). https://doi.org/10.1103/physreva.52.3457, https://doi.org/10.1103%2Fphysreva.52.3457
4. Cheng, C.Y., Yang, C.Y., Kuo, Y.H., Wang, R.C., Cheng, H.C., Huang, C.Y.R.: Robust qubit mapping algorithm *via* double-source optimal routing on large quantum circuits (2023)
5. Dahi, Z.A., Alba, E.: Metaheuristics on quantum computers: Inspiration, simulation and real execution. Future Generation Computer Systems **130**, 164–180 (2022). https://doi.org/https://doi.org/10.1016/j.future.2021.12.015
6. 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. p. 1164–1172. GECCO '22, Association for

Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3512290.3528830, https://doi.org/10.1145/3512290.3528830
7. Farhi, E., Goldstone, J., Gutmann, S.: Quantum adiabatic evolution algorithms *versus* simulated annealing (2002)
8. Gad, A.F.: Pygad: An intuitive genetic algorithm python library (2021)
9. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. p. 212–219. STOC '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866
10. Gyongyosi, L., Imre, S.: A survey on quantum computing technology. Computer Science Review **31**, 51–71 (2019). https://doi.org/https://doi.org/10.1016/j.cosrev.2018.11.002, https://www.sciencedirect.com/science/article/pii/S1574013718301709
11. 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. p. 1001–1014. ASPLOS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3297858.3304023
12. Li, Y., Tian, M., Liu, G., Peng, C., Jiao, L.: Quantum optimization and quantum learning: A survey. IEEE Access **8**, 23568–23593 (2020). https://doi.org/10.1109/ACCESS.2020.2970105
13. Qiskit contributors: Qiskit: An open-source framework for quantum computing (2023). https://doi.org/10.5281/zenodo.2573505
14. Quetschlich, N., Burgholzer, L., Wille, R.: MQT Bench: Benchmarking software and design automation tools for quantum computing (2022), MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/
15. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (Oct 1997). https://doi.org/10.1137/s0097539795293172
16. Siraichi, M.Y., Santos, V.F.D., Collange, C., Quintão Pereira, F.M.: Qubit Allocation. In: CGO 2018 - International Symposium on Code Generation and Optimization. pp. 1–12. Vienna, Austria (Feb 2018). https://doi.org/10.1145/3168822, https://hal.science/hal-01655951
17. Zhou, X., Li, S., Feng, Y.: Quantum circuit transformation based on simulated annealing and heuristic search. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(12), 4683–4694 (Dec 2020). https://doi.org/10.1109/tcad.2020.2969647, http://dx.doi.org/10.1109/TCAD.2020.2969647