Natural Computing (2023) 22:631–644
https://doi.org/10.1007/s11047-023-09955-0

Solving quantum circuit compilation problem variants through genetic algorithms

Lis Arufe[1] · Riccardo Rasconi[2] · Angelo Oddi[2] · Ramiro Varela[1] · Miguel Ángel González[1]

Accepted: 1 August 2023 / Published online: 17 August 2023

Abstract
The gate-based model is one of the leading quantum computing paradigms for representing quantum circuits. Within this paradigm, a quantum algorithm is expressed in terms of a set of quantum gates that are executed on the quantum hardware over time, subject to a number of constraints whose satisfaction must be guaranteed before running the circuit, to allow for feasible execution. The need to guarantee the previous feasibility condition gives rise to the Quantum Circuit Compilation Problem (QCCP). The QCCP has been demonstrated to be NP-Complete, and can be considered as a Planning and Scheduling problem. In this paper, we consider quantum compilation instances deriving from the general Quantum Approximation Optimization Algorithm (QAOA), applied to the MaxCut problem, devised to be executed on Noisy Intermediate Scale Quantum (NISQ) hardware architectures. More specifically, in addition to the basic QCCP version, we also tackle other variants of the same problem such as the QCCP-X (QCCP with crosstalk constraints), the QCCP-V (QCCP with variable qubit state initialization), as well as the QCCP-VX that includes both previous variants. All problem variants are solved using genetic algorithms. We perform an experimental study across a conventional set of instances taken from the literature, and show that the proposed genetic algorithm, termed GA_{VX}, outperforms previous approaches in the literature.

Keywords Quantum circuit compilation · Scheduling · Makespan · Optimization · Genetic algorithm

1 Introduction

In this paper, we explore the utilization of a Genetic Algorithm to optimize the resolution of the Quantum Circuit Compilation Problem (QCCP). Indeed, the problem of efficiently compiling quantum circuits to emerging quantum hardware is of primary importance, for a variety of reasons. At the current stage of development, quantum computing technology is based on mainly two architectures: quantum annealers and gate-model processors (Aharonov et al. 2004; Preskill 2018). However, while quantum annealers are restricted to the resolution of quadratic unconstrained binary optimization (QUBO) problems (one of the most representative technology of this type is the one used in the D-Wave Solvers[1]) gate-model processors are universal and, once properly scaled up, can be applied to any quantum algorithm. In this work, we will

✉ Miguel Ángel González
mig@uniovi.es

Lis Arufe
arufelis@uniovi.es

Riccardo Rasconi
riccardo.rasconi@istc.cnr.it

Angelo Oddi
angelo.oddi@istc.cnr.it

Ramiro Varela
ramiro@uniovi.es

[1] Department of Computer Science, University of Oviedo, Campus of Gijón, 33204 Gijón, Spain

[2] Istituto di Scienze e Tecnologie della Cognizione, Consiglio Nazionale delle Ricerche (ISTC-CNR), Via S. Martino della Battaglia, 44, 00185 Rome, Italy

[1] https://www.dwavesys.com.

focus on the resolution of combinatorial optimization problems using the gate-model approach.

In more details, we consider the hardware technology termed Noisy Intermediate Scale Quantum (NISQ) processors (Preskill 2018; Venturelli et al. 2019). Figure 1 shows six NISQ quantum chip designs featuring different number of qubits ($N = 4, 8, 21, 40, 72, 127$). The four smaller chips are inspired by Rigetti Computing Inc. Sete et al. (2016), while the 72-qubit architecture is the Google Bristlecone and the 127-qubit is the IBM Eagle. Each qubit is represented by a node in the architectures depicted in Fig. 1, and is identified by an integer. If an edge exists between two nodes (i.e., the two nodes are adjacent) then a binary quantum gate can be executed between the two qubits represented by those nodes. The type of connection, either dashed or continuous in the figure, is representative of different processing times in gate execution. In general, the duration of a quantum gate depends on the specific physical implementation in terms of primitive gates provided by the specific quantum hardware.

A quantum circuit (or quantum algorithm) may be viewed as a series of quantum gates that are applied on the qubits over time (see Fig. 2b) where nearest-neighbor restrictions exist on the nodes (i.e., the qubits) the gates are applied to.

Being the NISQ processors characterized by a number of qubits in the [50, 150] range, they are not large enough to demonstrate generalized quantum supremacy, nor are they sufficiently advanced for implementing continuous quantum error correction (QEC) strategies. Additionally, NISQ hardware is prone to *decoherence*, which deteriorates the reliability of the quantum circuit over time, hence the need to produce circuits whose execution terminates before they start to decohere. It is however important to recognize that minimizing the length of a quantum circuit in terms of makespan or depth is not the sole desirable objective for the attainment of a high-quality quantum circuit. Besides decoherence, quantum circuits running on NISQ quantum processors are inherently affected by noise stemming from the NISQ technology itself. Consequently, one may consider additional quality metrics that focus on the synthesis of quantum circuits that minimize the utilization of particularly error-prone quantum gates, for instance the binary gates like the CNOT gates.

In the context of this paper however, we will focus on the quality aspect related to the production of circuits that are as shallow as possible (i.e., compiled circuits characterized by minimum depth).

The compiled circuits considered in this work aim at resolving MaxCut problem instances (see Sect. 3). In general quantum terms, the resolution of these problem instances may be naturally viewed in terms of an energy minimization problems. More specifically, the main difficulty is to build the Hamiltonian H_p related to the objective function of the problem of interest; once H_p is established, its ground state will encode a solution to the problem.

According to the quantum annealing model, an initial Hamiltonian H_i is created whose ground state can be easily prepared; then, the initial Hamiltonian is transformed towards H_p through small perturbations. Finally, following the adiabatic theorem, we have a ground state of H_p. In the circuit model (the model considered in this paper) a set of quantum logic gates operate on a set of quantum bits (qubits), where each qubit finds itself on a determined quantum state (qstate), i.e. a superposition of the two pure states denoted $|0\rangle$ and $|1\rangle$ respectively. As in the adiabatic model, the goal is to prepare a ground state of H_p.

From all of the above, executing a quantum algorithm on a quantum hardware entails evaluating a set of quantum gates on qubits (in this study, unary or binary gates) subject to some constraints originated by both the algorithm and the hardware. To guarantee that the adjacency condition is satisfied prior to the execution of every binary gate, the pair of interested qstates must be frequently moved across the hardware's qubits. This is achieved by means of *swap* gates. Swap gates are binary gates that exchange the qstates resting on two adjacent qubits. Another constraint that should always be satisfied is that two gates cannot be concurrently applied on the same qubit.

The problem of distributing the calculations across the specific quantum device, due to the previous conditions and constraints, is known in the literature as the Quantum Circuit Compilation Problem (QCCP) (Oddi and Rasconi 2018). In this work, we formulate this problem within the planning and scheduling framework, and we explore the use of genetic algorithms to solve the original QCCP version as well as some QCCP variants that will be described in Sect. 4. To this aim, we will focus on the class of Quantum Approximate Optimization Algorithm (QAOA) applied to the MaxCut problem (Farhi et al. 2014). The QAOA paradigm has been widely used in the literature, see for example (Oddi and Rasconi 2018; Rasconi and Oddi 2019; Chand et al. 2019; Aruf et al. 2022), where the authors present results based on the classical benchmark proposed in Venturelli et al. (2017). QAOA is based on the application of a set of quantum gates (i.e., the quantum circuit) over a number p of rounds, though in this paper we restrict our analysis to circuits composed on $p = 1$ rounds only.

The remainder of the paper is organized as follows. The next two sections are dedicated to the general presentation of the QAOA, and to the description of how it is applied to the MaxCut problem, respectively. Section 4 is dedicated to the formal definition of the QCCP for MaxCut, and its

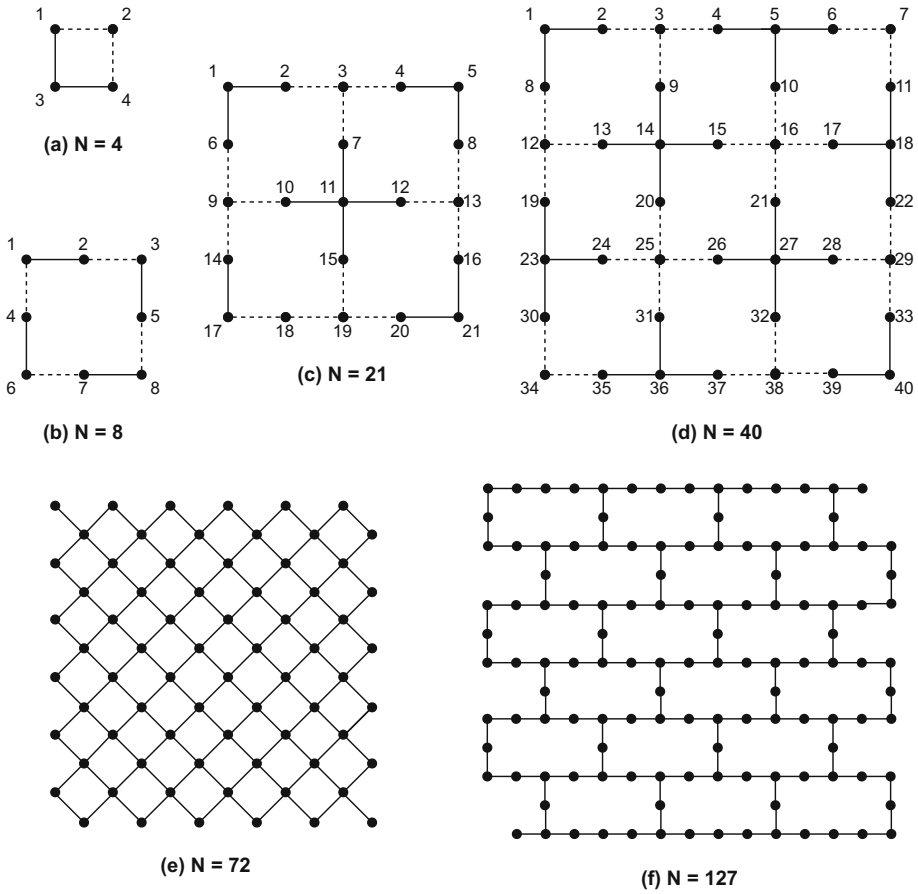


Fig. 1 Six quantum chip designs with different number of qubits

variants QCCP-V, QCCP-X and QCCP-VX. Section 5 describes the Genetic Algorithm (GA_{VX}) we propose. Section 6 is dedicated to the description of the experimental evaluation and to the presentation of the obtained results. Finally, Sect. 7 summarizes the main conclusions, and provides some ideas for future research.

2 The quantum approximate optimization algorithm

The Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al. 2014) is a "hybrid" heuristic algorithm, as it combines both classic and quantum computations to solve combinatorial problems expressed as

$$\text{optimize: } \sum_{\alpha=1}^{m} C_\alpha(\mathbf{z}) \tag{1}$$

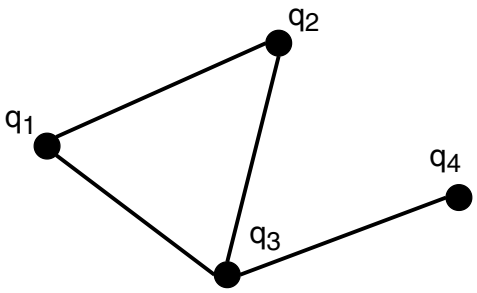
where $C_\alpha(\mathbf{z})$ are clauses on a vector of decision binary variables $\mathbf{z} = (z_1, \ldots, z_n)$, and the goal is to find the assignment of $z_i \in \{0, 1\}, 1 \le i \le n$, that optimizes the number of satisfied clauses.

In order to use the QAOA to solve problems like the MaxCut, it is first necessary to translate the clauses $C_\alpha(\mathbf{z})$ into their equivalent quantum Hamiltonians $\mathbf{C}_\alpha$, which is typically done by interpreting each variable z_i as a quantum spin, hence physically realizable with a qubit. Secondly, it is necessary to choose the number of rounds p the quantum circuit (i.e., the quantum part of the QAOA) will be composed of, as well as two vectors γ, β of p angles γ_i and β_i with $0 \le \beta_i \le \pi$, $0 \le \gamma_i \le 2\pi$, $1 \le i \le p$. Generally, the higher the number p of rounds, the more accurate is the solution returned by the QAOA; however, it is of great importance to select a value of p small enough to keep the quantum circuit depth limited, so as not to cause undesired decoherence effects.

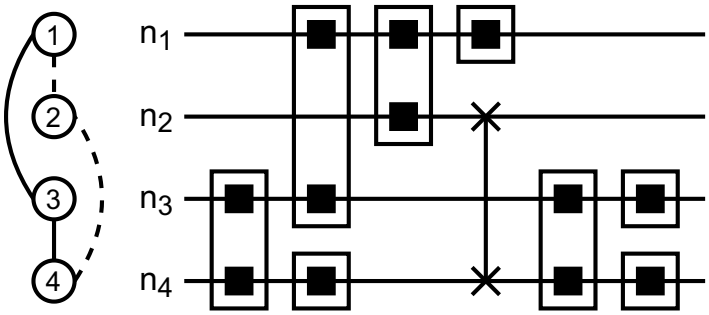
Then, starting from the n qubits in the *uniform superposition* qstate $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, the following state is prepared:

$$|\psi_p(\gamma, \beta)\rangle = \prod_{r=1}^{p} e^{-i\beta_r H_B} e^{-i\gamma_r H_C} |+\rangle^{\otimes n} \tag{2}$$

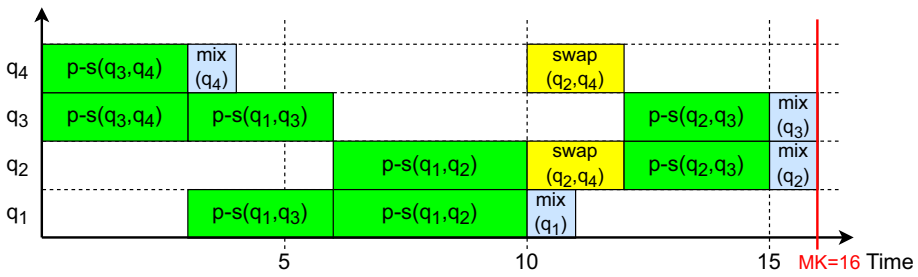
where H_C is the *problem Hamiltonian* defined as $H_C = \sum_{\alpha=1}^{m} \mathbf{C}_\alpha$, and H_B is the *mix Hamiltonian* defined as $H_B = \sum_{j=1}^{n} \mathbf{X_j}$, where $\mathbf{X_j}$ is the $\mathbf{X}$ Pauli matrix applied to qubit j.



(a) MaxCut instance.



(b) Quantum circuit for the MaxCut instance in Figure 2(a).



(c) Gantt chart, representing the operations on each qstate over time.

Fig. 2 Example of MaxCut instance (**a**) and one possible solution, considering only one round (i.e. $P = 1$), represented by a quantum circuit (**b**) and a Gantt chart (**c**)

By repeatedly applying the quantum circuit as per equation 2, the physical system realized by our quantum circuit is slowly brought close to its minimum energy level, which corresponds to a solution of the problem defined in Eq. (1), which therefore should be close to the optimum. The final obtained state is finally measured to get the expected value of our solution, given by

$$\langle \psi_p(\gamma, \beta) | H_C | \psi_p(\gamma, \beta) \rangle \tag{3}$$

If the values of γ and β are well chosen, the state of the qubits after this transformation will represent a good solution to the problem defined in Eq. (1) with high probability. In the QAOA optimization loop, the selection of γ and β is carried out by means of a classic optimization (e.g., simplex based or gradient based optimization) with the exception of the initial pair (γ_0, β_0) which must be provided as input. For each candidate (γ, β), the state of Eq. (2) is prepared and measured in the quantum computer.

3 The MaxCut problem in quantum terms

Given an undirected graph $G = (V, E)$, where V is the set of nodes ($|V| = n$) and E is the set of arcs ($|E| = m$), the goal of the MaxCut problem is to compute a partition of the set V into two subsets V_{+1} and V_{-1} so that the number of arcs in E connecting nodes of the two subsets is maximized. Formally, the goal can be expressed as follows:[2]

[2] Note that a single transformation $z = (\sigma + 1)/2$ converts the variables from the $\sigma \in \{-1, +1\}$ space to the $z \in \{0, 1\}$ space.

$$\text{maximize: } \sum_{(i,j)\in E} \frac{1}{2}(-\sigma_i\sigma_j), \quad \sigma_k = \begin{cases} -1 & \text{if } k \in V_{-1} \\ 1 & \text{if } k \in V_{+1} \end{cases} \tag{4}$$

Obviously, the problem can be transformed into a minimization problem by multiplying the expression above by -1. In this case, we obtain a Hamiltonian $\mathbf{C}_\alpha$ for each arc (i, j) which depends on just these two variables; hence, it can be expressed as

$$\mathbf{C_{i,j}} = \frac{1}{2}(\mathbf{Z}_i \otimes \mathbf{Z}_j) \tag{5}$$

where $\mathbf{Z_i}$ and $\mathbf{Z_j}$ are the Pauli matrix $\mathbf{Z}$ applied to qubits i and j respectively. Therefore, each of the m components of the problem Hamiltonian H_C corresponding to the terms in equation 5 can be formulated in matrix form, obtaining the following operator:

$$e^{-i\gamma_r \mathbf{C_{i,j}}} = \begin{pmatrix} e^{-i\gamma_r/2} & 0 & 0 & 0 \\ 0 & e^{i\gamma_r/2} & 0 & 0 \\ 0 & 0 & e^{i\gamma_r/2} & 0 \\ 0 & 0 & 0 & e^{-i\gamma_r/2} \end{pmatrix} \tag{6}$$

which is in fact an $R_{ZZ}(\gamma)$ gate. As clearly visible, the operator corresponding to the $R_{ZZ}(\gamma)$ gate is diagonal; therefore, any two of such operators can commute each other. As a direct consequence, all the $R_{ZZ}(\gamma)$ gates present in the quantum circuit can be executed in any order on every round, thus making the quantum compilation problem very interesting.

Similarly, each component of the mix Hamiltonian H_B corresponds to the following unitary operator:

$$e^{-i\beta_r \mathbf{X_j}} = \begin{pmatrix} \cos(\beta_r) & -i\sin(\beta_r) \\ -i\sin(\beta_r) & \cos(\beta_r) \end{pmatrix} \tag{7}$$

which is obviously the matrix representation of the $R_X(2\beta)$ gate.

In the following sections, operators (6) and (7) will be referred to as $p - s(q_i, q_j)$ and $mix(q_j)$ respectively.

4 Definition of the Quantum Circuit Compilation Problem

The QCCP can be formalized with the tuple $P = \langle C_0, L_0, QM \rangle$, where:

- C_0 is the input quantum circuit, which represents the quantum algorithm that solves the problem of interest;
- L_0 is the circuit *initial mapping*, i.e., the initial allocation of every qstate q_i on every qubit;
- QM is the quantum hardware whose topology is represented by an undirected graph $QM = \langle V, E \rangle$, where V is a set of nodes that represent physical qubits, and E is a set of edges that represent the physical connections existing on the quantum chip. In order to be feasibly executed, the qstates q_i and q_j of every $p - s(q_i, q_j)$ or $swap(q_i, q_j)$ gate must reside on qubits connected by an edge (adjacent qubits). The durations of the gates depend both on the gate and on the type of the arc (depicted as continuous or dashed).

Each MaxCut quantum circuit instance is characterized by a determined number of $p - s(q_i, q_j)$ (*P-S* set) and $mix(q_i)$ gates (*MIX* set). In general, the operations in *P-S* and *MIX* are applied sequentially over a number of p rounds, but in this work we are considering just one round ($p = 1$).

Figure 2a shows an example of MaxCut instance for a graph with 4 nodes and the $p - s$ and *mix* gates that must be executed over a number of rounds. There are some precedence constraints that all solutions must satisfy; for instance, a $mix(q_i)$ gate can only be executed after all $p - s$ gates in the same round involving the qstate q_i are executed. However, $p - s$ gates may be executed in any order within each round, provided that no two $p - s$ gates operates on the same qstate at a time (i.e., they are commutative).

For a $p - s(q_i, q_j)$ gate to be executed, the involved qstates q_i and q_j must be allocated on a pair of adjacent qubits $(n_k, n_l) \in E$. Every time this adjacency condition is not satisfied, it is necessary to extend the quantum circuit by adding a number of *swap* gates whose task is to move the qstates of interest towards a pair of adjacent qubits. Therefore, solving the circuit compilation problem basically entails the following steps: (i) we start from an initial allocation of qstates on the qubits (decided in L_0); (ii) we insert a number of *swap* gates in the circuit, to ensure that the adjacency conditions are satisfied for all $p - s$ gates. The overall objective is the production of compiled circuits of minimum makespan.

The right side of Fig. 2b shows a compiled quantum circuit that represents a solution for the problem given in Fig. 2a. Given the simplicity of the circuit, only one *swap* gate was necessary to guarantee the adjacency conditions for all the $p - s$ gates. This was achieved by swapping the qstates on qubits n_2 and n_4 so as to move the qstates q_3 and q_2 on the adjacent qubits n_3 and n_4, thus making the execution of the $p - s(q_3, q_2)$ feasible. In the figure, the rectangles represent *p-s* gates, the squares represent *mix* gates, and the x-ended line represents a *swap* gate. In this example, the initial allocation defined by L_0 establishes that each qstate q_i is on qubit n_i. The quantum hardware is represented on the left side of Fig. 2b. Each horizontal line represents the operations on the n_i qubit over time.

As shown in Fig. 2C, a compiled quantum circuit may also be viewed as a Gantt chart. Clearly, the makespan of this particular solution is equal to 16, assuming that a $p-s$ gate takes 3 time units on continuous edges and 4 time units on dashed edges, that the *swap* gate requires 2 time units on all edges, and the *mix* gates require 1 time unit.

Besides the original QCCP version, in this work we consider three more variants, described in the following.

QCCP-V Unlike the original QCCP variant, in which the initial qstate assignment L_0 was equal for every instance and fixed (i.e., it could not be modified by the optimization algorithm), in the QCCP-V version, the initial state is variable and must be decided by the compilation procedure. This naturally allows for the production of compiled circuits characterized by shorter makespans.

QCCP-X The QCCP-X variant assumes that the quantum hardware is affected by *crosstalk* noise, which adds up the following further constraint to our circuits (*crosstalk* constraint): no two gates can operate simultaneously on adjacent qubits. Obviously, the consequence of satisfying the crosstalk constraint leads to the production of compiled circuits characterized by longer makespans.

QCCP-VX In this variant, both the variable initial state computation and the crosstalk constraint satisfaction are taken into account.

5 The genetic algorithm

The problem variants described in Sect. 4 will be solved by means of a genetic algorithm denoted GA_{VX}. In particular, we adapt here the genetic algorithm proposed in Arufe et al. (2022) for the QCCP-X which, in turn, is adapted from the DBGA (Decomposition Based Genetic Algorithm) described in Aruf et al. (2022) for the plain QCCP.

Chromosome codification is a triplet of chains ($chr1$, $chr2$, $chr3$), where $chr1$ is a permutation of the set of the *p-s* gates, $chr2$ is a sequence of connections in the quantum circuit *QM* of the same length as $chr1$. The idea is that gate $p-s(q_i,q_j)$ in position i of $chr1$ must be executed on qubits $\{n_k,n_l\}$ indicated in position i of $chr2$. The decoding algorithm inserts the minimum number of *swap* gates necessary to move the qstates (q_i,q_j) from their current qubits towards $\{n_k,n_l\}$ (see (Aruf et al. 2022) for details on the procedure). Additionally, $chr3$ is a permutation of the N qubits of the quantum hardware considered, each position indicating the qubit in which each qstate starts, and so it will be used for the QCCP-V problem variant.

The genetic algorithm starts by creating a random initial population of *popSize* chromosomes and evaluating them. It then iterates a number of generations until a stop condition is met, in particular a maximum number of consecutive generations without improving the best solution found so far. When the algorithm ends, the best solution in the population is returned.

In each generation, all chromosomes of the population are shuffled in random pairs, and an adapted version of the well-known partial mapping crossover (PMX) operator is applied to each pair with probability P_c in order to produce two offspring solutions. Regarding the $chr3$ chain of the chromosomes, in the crossover step one offspring inherits a complete $chr3$ of one parent and the other offspring inherits the complete $chr3$ of the other parent (an example is shown in Fig. 3).

Then, a mutation operator is applied to each offspring with probability P_m. We consider two mutation operators that are chosen with equal probability when a chromosome is mutated. One swaps two random positions of the $chr1$ and $chr2$ chains, and the other randomly modifies a position of $chr2$.

All offsprings are evaluated, and the two best chromosomes from each pair of parents and its two offspring are inserted in the population for the next generation.

A diversification step is performed after *noImpr* consecutive generations without improvement. Its purpose is dealing with premature convergence by introducing diversity in the population, and consists in mutating *nMutDiv* times all but one chromosomes with the same fitness.

5.1 Variable qubit state initialization (QCCP-V)

As already described, the Quantum Circuit Compilation Problem with variable qubit state initialization (QCCP-V) is an extension of the problem where the initial mapping of qstates over qubits is variable. Varying the initial mapping allows to build circuits with possibly less swap gates and lower makespan, however it makes the problem more complex as we have extra variables to optimize.

In this paper we propose to tackle this extension combining two methods: a heuristic taken from the literature (Alam et al. 2020) and a novel enhancement to further improve the resulting circuits when decoding a chromosome, which we will denote dynamic reallocation. We describe those ideas in the following.

Integrated qubit allocation and initial mapping (QAIM)

In Alam et al. (2020) the authors propose a very interesting heuristic method to perform this initialization, denoted QAIM. In summary, given an instance, the QAIM

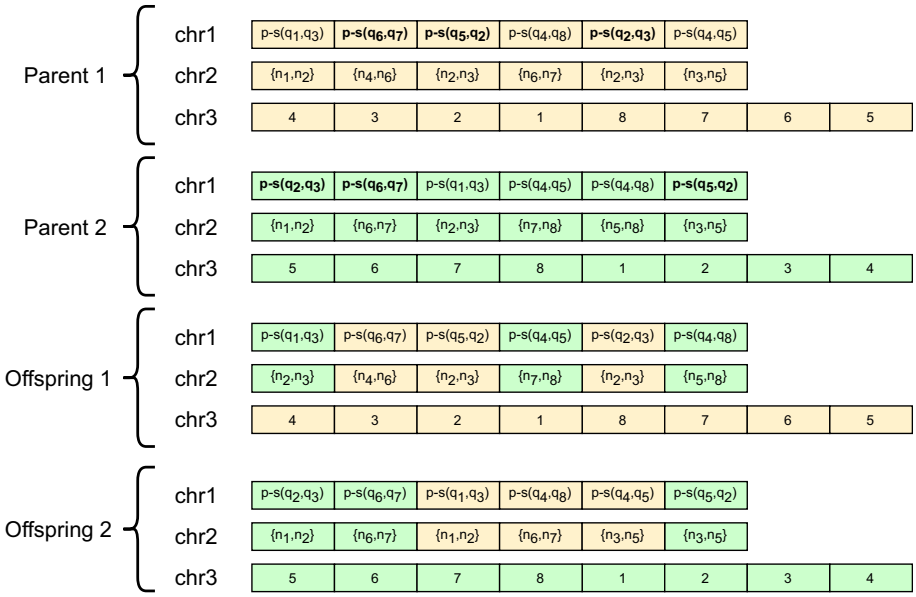


Fig. 3 Illustration of crossover with the proposed three-chain chromosome

heuristic tries to detect which qstates are more heavily used and tries to place them near each other, and favoring those qubits with many connections in the quantum chip. In this way it is expected that the number of swap gates is reduced. We refer the interested reader to Alam et al. (2020) for full details of the QAIM heuristic.

This heuristic is only applied at the beginning of the execution, as its result only depend on the particular problem instance and on the quantum chip used. However, different initializations might be created using QAIM, because in some decisions made in the heuristic there might be a tie between several options (for example if several qubits have the same cost metric—qubit connectivity strength/cumulative distance from the placed neighbors).

In our experiments we try two possibilities: (1) using a single output of the QAIM heuristic and using it in all chromosomes, or (2) leverage the randomness of the method and create a number (to be determined in the experiments) of initial profiles, assigning one to each chromosome.

Dynamic reallocation The so called dynamic reallocation allows to further improve the solutions. Whenever we are decoding a chromosome, if we need to insert a swap gate between two qstates, but that swap gate would be the first gate processed by those two qstates, then it is easy to see that we can omit the swap gate and instead swap the initial mapping of those qstates.

There are two ways of implementing this idea, depending if we consider the modifications made to the initial mapping inheritable or non-inheritable. These two ideas correspond to the classical evolution strategies of Lamarckian and Baldwinian evolution, respectively. In the literature it is usually the case that Lamarckian evolution (i.e. the modifications are inheritable) performs better, and we will confirm it in the experimental study.

In Fig. 4 we show an example of the efficiency of these approaches. In particular we show how we decode a chromosome in three different scenarios: default qstate initialization (makespan 28), QAIM initialization (makespan 16), and QAIM with dynamic reallocation (makespan 12).

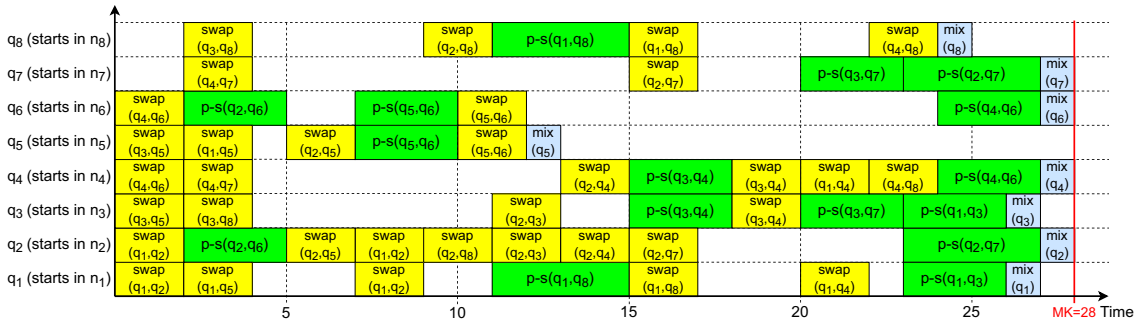
6 Experimental study

Our genetic algorithm GA_{VX} is implemented in C++ and runs in a 64-bit Windows10 OS on top of a Intel Core i5-7400 CPU at 3.00 GHz with 16 GB RAM. The algorithm runs 10 times in each experiment in order to obtain statistically significant results.

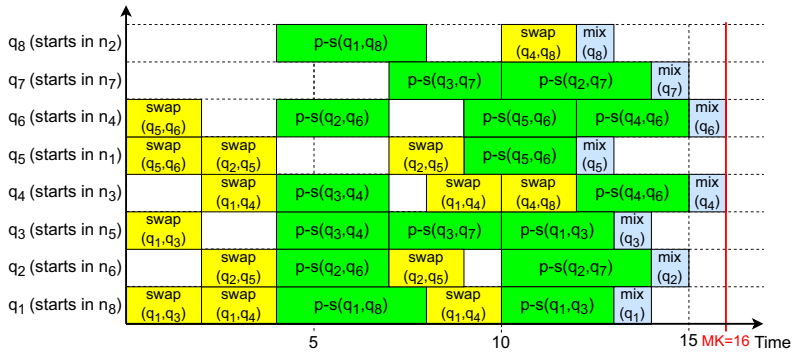
We consider a well-known benchmark initially proposed in Venturelli et al. (2017) which is publicly available[3]. It includes instances for four different quantum architectures $N = \{4, 8, 21, 40\}$, two different utilization levels (i.e. maximum percentage of qstates of the quantum chip that are used in the instance) $u = \{90\%, 100\%\}$ and two different number of compilation passes $p = \{1, 2\}$. Each combination $\{N, p, u\}$ has 50 different instances. In this paper we focus on those instances with $u = 100\%$ and $p = 1$.

We also consider additional instances in recent larger quantum architectures (sizes $N = 72$ and $N = 127$). There

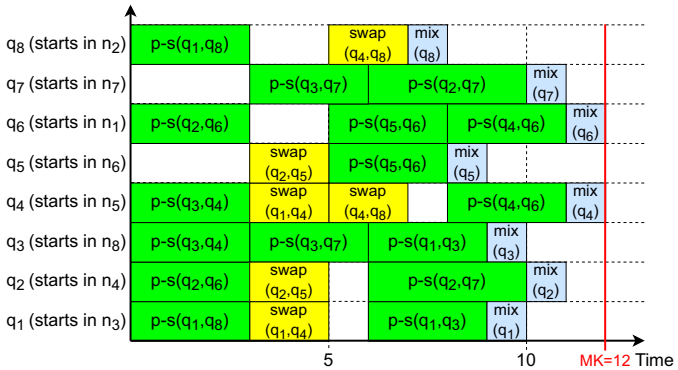
[3] https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_data.zip.



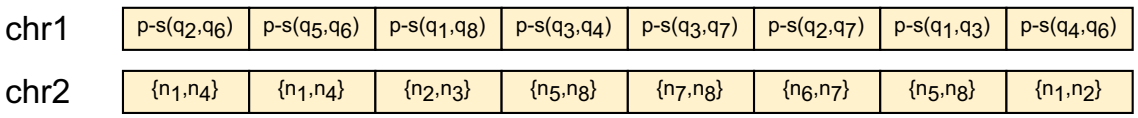
(a) Gantt chart using default initial mapping.



(b) Gantt chart using QAIM heuristic.



(c) Gantt chart using QAIM heuristic and Dynamic Reallocation.



(d) Chromosome.

Fig. 4 Example of MaxCut instance number 5 of size $N = 8$ and one possible chromosome. We show three possible solutions, represented by Gantt charts, obtained when decoding the chromosome considering different initial mapping strategies

instances are built using the same methodology as those of Venturelli et al. (2017).

In an initial parameter analysis we have concluded that the following parameters are reasonable for GA_{VX}: $popSize = 1000$, $P_c = 100\%$, $P_m = 5\%$, $noImpr = 10$, $nMutDiv = 5$. We set a stop condition of 800 consecutive generations without improving the best solution found so far. In this way we obtain computational times no larger

than those reported in Rasconi and Oddi (2019), so we can provide a relatively fair comparison of both methods in section 6.2.

6.1 Comparison of qstate initialization strategies

Table 1 shows a comparison of the QAIM variants described in Sect. 5.1 with different number of created profiles. We show for each strategy and each instance, the average result obtained in 10 runs and the standard deviation. We only consider here the largest instances, i.e. those with $N = 72$ and $N = 127$, as those are the most difficult and interesting. We consider here the problem variant QCCP-V, with no crosstalk constraints.

The first thing we conclude is that ignoring the possibility of variable initialization is very costly, as the makespan is approximately twice than the makespan using any of the QAIM versions.

In particular, QAIM(100) is the variant that obtains the best average result more often: 6 times out of 10 in $N = 72$ instances and 7 times out of 10 in $N = 127$ instances, although the difference with respect to QAIM(10) seems to be reduced. We performed statistical analysis to check if QAIM(100) is in fact better, and we see that paired Wilcoxon signed rank tests return a p-value of 0.0000009537 versus no QAIM, 0.006293 versus QAIM(1) and 0.00006676 versus QAIM(1000), showing the differences are statistically significant. On the other hand, differences with respect to QAIM(10) are not statistically significant. We choose anyway QAIM(100) for the remaining experiments, as it obtained a better average more often than QAIM(10) in these experiments.

Now that we have seen that QAIM(100) might be the best alternative, we will see the effect of the dynamic reallocation and compare its inheritable (Lamarckian evolution) and non-inheritable (Baldwinian evolution) versions. Table 2 shows the results of this comparison.

The inheritable version of the dynamic reallocation seem to perform better, obtaining an average makespan 8.9% lower than that obtained with no dynamic reallocation, and obtaining the best average result in most instances. A paired Wilcoxon signed rank test confirms that the

Table 1 Average makespan and standard deviation in 10 runs, using different QAIM strategies to tackle initialization of qstates in qubits, considering the largest instance sizes $N = \{72, 127\}$

	Default		QAIM(1)		QAIM(10)		QAIM(100)		QAIM(1000)	
Inst	Avg	S.D	Avg	S.D	Avg	S.D	Avg	S.D	Avg	S.D
$N = 72$ **Instances**										
1	55.1	4.36	30.3	2.45	**27.3**	2.21	28.9	2.38	30.3	3.2
2	59.6	5.87	32.5	2.68	**32.8**	4.54	33.5	3.17	34.7	3.83
3	58.8	4.85	30.5	2.46	**29.5**	3.6	30.1	2.56	30.9	1.91
4	54.5	3.89	29.3	2.36	**27.6**	2.5	29.0	3.46	28.1	2.77
5	60.2	3.82	34.4	2.76	34.6	3.6	**32.8**	3.22	33.8	1.23
6	55.5	4.12	31.7	3.71	**30.4**	2.22	31.7	3.23	33.7	2.67
7	56.2	6.23	30.4	3.6	30.8	3.08	**29.8**	3.08	32.2	2.94
8	55.4	5.36	31.5	2.37	**28.2**	2.35	**28.2**	1.93	30.9	2.42
9	55.0	4.14	33.0	3.3	30.5	3.14	**30.2**	2.53	32.4	2.5
10	52.2	4.1	31.5	2.8	30.5	2.01	**29.8**	4.24	30.4	2.01
#best	0		0		6		5		0	
$N = 127$ **Instances**										
1	145.8	13.43	60.5	5.4	**61.8**	5.12	63.9	5.57	66.6	7.59
2	148.7	14.1	65.3	10.06	55.1	6.59	**51.7**	5.56	56.7	3.53
3	161.5	12.43	66.1	9.22	**61.5**	8.87	65.4	10.81	63.7	5.52
4	154.6	14.05	67.6	9.18	57.6	9.3	**55.5**	6.38	61.7	6.93
5	150.7	8.43	75.0	9.52	**72.3**	8.9	72.4	7.29	75.1	7.98
6	135.6	10.86	62.9	9.29	**62.1**	9.23	66.9	5.97	67.1	5.99
7	161.1	12.19	58.1	5.2	**53.7**	6.41	57.1	7.36	62.4	6.43
8	141.9	8.09	62.4	7.82	**56.3**	6.06	57.8	4.89	60.7	7.42
9	156.0	17.75	69.3	7.85	64.3	6.62	**58.9**	4.86	67.5	6.19
10	154.1	12.21	70.4	6.92	**62.1**	4.18	64.8	6.78	69.4	9.34
#best	0		0		7		3		0	

We mark in bold the lowest average result in each instance

Table 2 Average makespan and standard deviation in 10 runs, using QAIM(100) with different dynamic reallocation strategies to tackle initialization of qstates in qubits, considering the largest instance sizes $N = \{72, 127\}$

	No DR		DR(Non-Inhe)		DR(Inhe)	
Instance	Avg	S.D	Avg	S.D	Avg	S.D
$N = 72$ **Instances**						
1	27.3	2.21	28.3	1.64	**26.5**	0.53
2	32.8	4.54	29.9	2.56	**29.7**	1.95
3	29.5	3.6	**28.1**	2.85	28.6	2.41
4	27.6	2.5	25.8	2.25	**24.7**	2.06
5	34.6	3.6	30.7	2.79	**30.5**	2.37
6	30.4	2.22	29.7	1.57	**28.3**	2.63
7	30.8	3.08	29.2	2.7	**27.7**	1.95
8	28.2	2.35	26.4	2.46	**25.2**	1.87
9	30.5	3.14	27.7	2.11	**27.5**	2.17
10	30.5	2.01	28.9	1.2	**27.4**	2.17
#best	0		1		9	
$N = 127$ **Instances**						
1	61.8	5.12	**57.4**	3.2	58.5	6.64
2	55.1	6.59	52.3	4.19	**49.7**	3.68
3	61.5	8.87	59.2	5.61	**53.1**	5.67
4	57.6	9.3	51.7	4.19	**49.7**	5.31
5	72.3	8.9	62.3	8.71	**61.8**	7.47
6	62.1	9.23	57.5	5.76	**54.0**	5.01
7	53.7	6.41	**52.8**	4.96	53.1	3.98
8	56.3	6.06	55.0	6.53	**52.7**	4.32
9	64.3	6.62	59.2	5.2	**56.3**	4.08
10	62.1	4.18	**56.7**	5.85	60.9	11.19
#best	0		3		7	

We mark in bold the lowest average result in each instance

differences are statistically significant, with a p-value of 0.00004771 versus not using dynamic reallocation, and a p-value of 0.005251 versus the non-inheritable version of dynamic reallocation.

We conclude that the best approach for variable initialization is QAIM(100) with inheritable dynamic reallocation, and so this is the configuration used by GA_{VX} in the following section.

6.2 Comparison with state of the art

In this Section we compare GA_{VX} results in different versions of the problem with those of the state of the art, which as far as we know is the genetic algorithm proposed in Rasconi and Oddi (2019), as it obtains better results than those previously reported in Booth et al. (2018). The maximum cpu time used in Rasconi and Oddi (2019) is 5, 30 and 90 s for instances of size $N = 8, 21, 40$ respectively, and so slightly larger than that used by GA_{VX}, although they are not directly comparable due to differences in target machine and programming language.

The benchmark set we consider in this Section is restricted to quantum chips with $N = \{8, 21, 40\}$ qubits, as no results with larger quantum architectures are given in Rasconi and Oddi (2019). We will perform the comparison in several versions of the problem: the standard QCCP, adding variable initialization of qstates (QCCP-V), adding crosstalk constraints (QCCP-X), and adding both (QCCP-VX).

Figures 5a, 5b and 5c show results depending on the number of qubits of the considered architecture. In particular they show the percentage reduction in makespan of the average results of GA_{VX} (in 10 runs) with respect to the method proposed in Rasconi and Oddi (2019).

We see that GA_{VX} seems to perform better, and the makespan reduction it achieves with respect to the method of Rasconi and Oddi (2019) varies depending on chip size. In particular, an average makespan reduction of 5.76% in $N = 8$ instances, of 19.00% in $N = 21$ instances, and of 32.08% in $N = 40$ instances. This makes sense, as in smaller and easier instances it should be relatively easy to obtain good solutions and so there is less room to further optimize them.

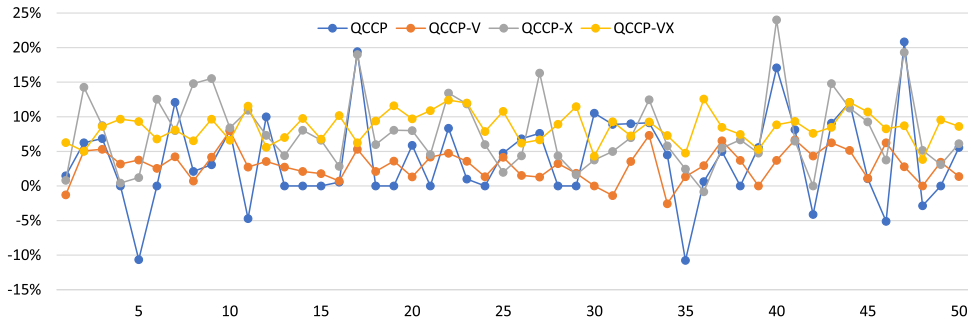
The makespan reduction also heavily depends on the problem variant. In the standard QCCP it ranges from an average of 3.86% for $N = 8$ instances to an average of 12.94% for $N = 40$ instances. For QCCP-V it ranges from 3.00% to 31.53%. For the QCCP-X version it ranges from 7.72% to 38.13%. Finally, for QCCP-VX it ranges from 8.47% to 47.71%. Therefore, the consideration of additional characteristics favors our method over that of Rasconi and Oddi (2019).

Regarding the best solutions reached, the proposal of Rasconi and Oddi (2019) obtains a better best than GA_{VX} in only 10 instances out of the total 600 tested instances (50 instances $\times$ 3 chip sizes $\times$ 4 problem versions). Similarly, it obtains a better average than GA_{VX} in only 13 out of 600 instances.

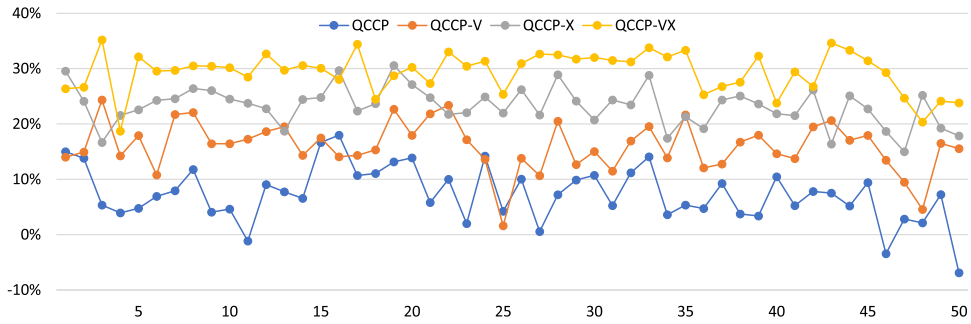
We confirmed the superiority of GA_{VX} through paired Wilcoxon signed rank tests, whose results show that the differences between both methods are statistically significant.

6.3 Influence of the problem variant and quantum chip topology

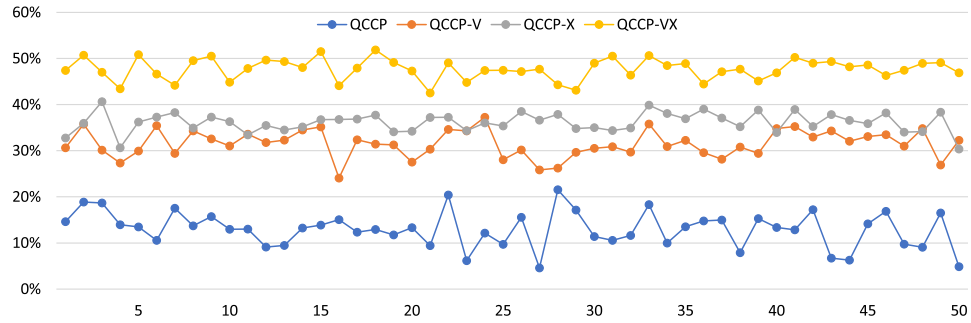
As we performed experiments considering four different problem variants and five different quantum chips, we are able to extract some insights about how much the variable initialization and the crosstalk constraint influence the



(a) N=8 instances



(b) N=21 instances



(c) N=40 instances

Fig. 5 Percentage improvement in makespan of GA_{VX} with respect to GA (Rasconi and Oddi 2019) in all 50 instances of each of the **a** N = 8, **b** N = 21 and **c** N = 40 benchmarks, considering four problem variants

Table 3 Comparison of average makespan increase between different problem variants, considering different chip sizes

	$N = 8$ (%)	$N = 21$ (%)	$N = 40$ (%)	$N = 72$ (%)	$N = 127$ (%)
QCCP → QCCP-V	− 16.99	− 23.90	− 31.72	− 52.68	− 64.41
QCCP-X → QCCP-VX	− 14.15	− 23.87	− 30.70	− 46.66	− 61.05
QCCP → QCCP-X	31.05	34.72	36.82	68.80	27.33
QCCP-V → QCCP-VX	35.47	34.93	38.89	82.71	39.53
QCCP → QCCP-VX	12.28	2.55	− 5.32	− 10.03	− 50.45

makespan. We show in table 3 the makespan increase (or decrease when the shown percentage is negative) between different problem variants depending on chip size.

When comparing QCCP to QCCP-V and QCCP-X to QCCP-VX we can conclude that variable initialization greatly improves the makespan. The improvement is larger as the chip size increases. It was expected, because in smaller chips two qstates cannot be very far away, and so even without a good initialization we will probably not need many *swap* gates anyways.

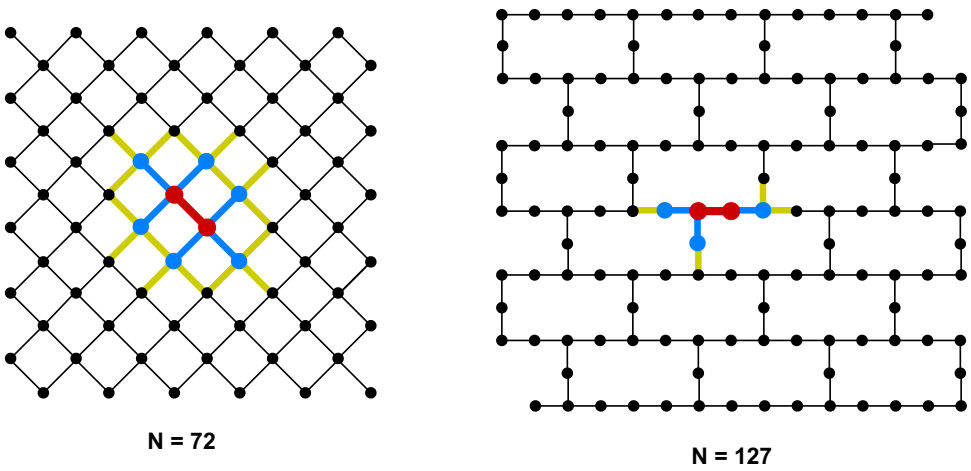


Fig. 6 Comparing how crosstalk constraints affect the $N = 72$ and $N = 127$ quantum chips. We mark in red the connection in which a quantum gate is executing, in blue the connections that are blocked due to sharing a qstate with the aforementioned quantum gate, and in yellow the connections that are blocked due to crosstalk constraints

On the other hand, when we compare QCCP to QCCP-X and QCCP-V to QCCP-VX we see that crosstalk constraints lead to worse makespans. The most outstanding issue here is the terrible performance of the $N = 72$ quantum chip, and the reason is that its topology is not ideal to tackle crosstalk constraints, because the qubits are more densely connected than in other architectures. Notice that if one quantum gate is executed on the $N = 72$ chip, as much as 6 neighboring connections can be blocked because they share a qubit with the gate, and as much as 16 additional connections are blocked around that quantum gate due to the crosstalk constraints (i.e. 19.01% of the total chip connections become unusable). Compare that with the $N = 127$ quantum chip, where if we execute a quantum gate it can block at most 3 neighboring connections due to qubit sharing and at most an additional 4 connections due to crosstalk constraints (i.e. 5.56% of the total chip connections become unusable). Therefore, the chip topology can explain the differences in performance when adding crosstalk constraints. Figure 6 illustrates this point.

Finally, we can compare QCCP with QCCP-VX in order to see how both considerations at the same time can affect the makespan. It seems that when the chip gets larger the makespan reduction of the variable qubit initialization outweighs the makespan increase of the crosstalk constraints. Although the difference is much more noticeable in the $N = 127$ chip than in the $N = 72$ chip, as in the former the crosstalk constraints are less severe, as we have seen before.

The full results and detailed schedules of GA_{VX} in the four variants of the problem and in all instances with size $N = \{8, 21, 40, 72, 127\}$, and also the definition of the instances with size $N = \{72, 127\}$ are publicly available on the web.[4].

[4] Repository section in http://di002.edv.uniovi.es/iscop/. In particular, in the "Detailed Results from Papers" subsection.

7 Conclusions

In this paper we study the application of a genetic approach to solve the quantum circuit compilation problem applied to a class of QAOA circuits that include crosstalk constraints (QCCP-X) and/or variable qubit initialization (QCCP-V). The objective is to obtain quantum gate execution plans that successfully compile idealized circuits to quantum hardware architectures, and we aim to minimize its makespan. We build upon the work presented in Arufe et al. (2022) that only tackles the QCCP-X version of the problem.

The main contribution of this work is a genetic algorithm, denoted GA_{VX}, that leverages a triple-chain chromosome encoding, such that the first chain encodes the sequence of gate insertion of the output solution, the second chain encodes the location where the gates must be executed in the considered quantum chip, and finally the third chain encodes the initial location of qstates in qubits. Another contribution is the method used to tackle the variable initialization issue that combines the heuristic proposed in Alam et al. (2020) with a novel dynamic reallocation mechanism that can modify the initialization at the same time as the corresponding chromosome is decoded. Additionally, in the performed experimental study we have considered recent larger quantum architectures with $N = 72$ and $N = 127$ qubits, which are not considered in previous QCCP-V or QCCP-X studies, and provide results for them in order to promote future research.

We have compared GA_{VX} with the approach proposed in Rasconi and Oddi (2019), which is a genetic algorithm where each gene controls the iterative selection of a quantum gate to be inserted in the solution, over a lexicographic double-key quantum gate ranking returned by a heuristic function. The comparison is performed in several variants of the problem (QCCP, QCCP-V, QCCP-X, QCCP-VX) in a well-known benchmark of the literature

(Booth et al. 2018; Venturelli et al. 2017) that considers quantum architectures with $N = 8$, $N = 21$ and $N = 40$ qubits. The results prove that GA_{VX} exhibits efficient performance, and we have also presented some insights about the solution quality when we consider different variants of the problem and different chip topologies.

We plan several avenues for future work:

- Consider the application of more than one compilation pass. Indeed, choosing the best number p of compilation passes is a problem on its own, as while increasing p improves the accuracy of the quantum circuit, it may eventually increase the circuit's depth to a point where the decoherence effect makes the circuit totally unreliable.
- Apply QAOA to different problems, as the Minimum Vertex Cover Problem (Zhang et al. 2022) or the Graph Coloring Problem (Do et al. 2020).
- Consider multi-objective optimization, for example minimizing both the makespan and the number of gates, as in general, while reducing the makespan decreases the circuit's decoherence, reducing the number of gates (especially the number of the noisy binary quantum gates) may also help increasing the circuit's stability. Both objectives are interesting to minimize but might be mutually conflicting objectives (Li et al. 2019). For the previous reasons, devising a multi-objective optimization approach aimed at producing a Pareto set of solutions with circuit's makespan and number of binary quantum gates (e.g., CNOTs) as objective functions may allow the human decision maker to select the solutions from the Pareto set that best fit the current trade-off requirements.
- Update the study with new hardware architectures, as the recent IBM Osprey (433 qubits), or the yet-to-come IBM Condor processor (1121 qubits).

Author Contributions Lis Arufe performed the implementation and source code and provided ideas about effects of crosstalk constraints depending on chip topology. Miguel Ángel González designed the experimental study and wrote the methods and experimental study parts of the manuscript. Angelo Oddi, Riccardo Rasconi and Ramiro Varela provided the theoretical basis of the paper and wrote those parts of the manuscript. All authors reviewed the final manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This research was supported by the Spanish Government under project PID2019-106263RB-I00 and by ESA Contract No. 4000112300/14/D/MRP Mars Express Data Planning Tool MEXAR2 Maintenance.

Data availibility statement The data that supports the findings of this study is available online, in the Repository Section of http://di002.edv.uniovi.es/iscop/. In particular, in the "Detailed Results from Papers" subsection.

Declarations

Conflict of interest Ramiro Varela is editor of the "Bio-inspired Computing Approaches for Problem Solving" special issue of the *Natural Computing* journal. Besides, there are no more relevant financial or non-financial interests to disclose.

Ethical approval Not applicable.

References

Aharonov D, van Dam W, Kempe J, Landau Z, Lloyd S, Regev O (2004) Adiabatic quantum computation is equivalent to standard quantum computation

Alam M, Ash-Saki A, Ghosh S (2020) Circuit compilation methodologies for quantum approximate optimization algorithm. In: 2020 53rd annual IEEE/ACM international symposium on microarchitecture (MICRO), pp 215–228

Aruf, L, González MA, Oddi A, Rasconi R, Varela R (2022) Quantum circuit compilation by genetic algorithm for quantum approximate optimization algorithm applied to maxcut problem. Swarm Evol Comput 101030

Arufe L, Rasconi R, Oddi A, Varela R, González MÁ (2022) Compiling single round qccp-x quantum circuits by genetic algorithm. In: Ferrández Vicente JM, Álvarez-Sánchez JR, de la Paz López F, Adeli H (eds) Bio-inspired systems and applications: from robotics to ambient intelligence. Springer, Cham, pp 88–97

Booth KE, Do M, Beck JC, Rieffel E, Venturelli D, Frank J (2018) Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In: Twenty-eighth international conference on automated planning and scheduling (ICAPS 2018), pp 366–374

Chand S, Singh HK, Ray T, Ryan M (2019) Rollout based heuristics for the quantum circuit compilation problem. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp 974–981

Do M, Wang Z, O'Gorman B, Venturelli D, Rieffel E, Frank J (2020) Planning for compilation of a quantum algorithm for graph coloring

Farhi E, Goldstone J, Gutmann S (2014) A quantum approximate optimization algorithm

Li G, Ding Y, Xie Y (2019) 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. Association for Computing Machinery, New York, NY, USA, pp 1001–1014

Oddi A, Rasconi R (2018) Greedy randomized search for scalable compilation of quantum circuits. In: van Hoeve W-J (ed) CPAIOR 2018: integration of constraint programming, artificial

intelligence, and operations research. Springer, Cham, pp 446–461
Preskill J (2018) Quantum Computing in the NISQ era and beyond. Quantum 2:79
Rasconi R, Oddi A (2019) An innovative genetic algorithm for the quantum circuit compilation problem. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, vol. 33, pp. 7707–7714
Sete EA, Zeng WJ, Rigetti CT (2016) A functional architecture for scalable quantum computing. In: 2016 IEEE international conference on rebooting computing (ICRC). IEEE, pp 1–6
Venturelli D, Do M, O'Gorman B, Frank J, Rieffel E, Booth KE, Nguyen T, Narayan P, Nanda S (2019) Quantum circuit compilation: an emerging application for automated reasoning. In: Bernardini S, Talamadupula K, Yorke-Smith N (eds) Proceedings of the 12th international scheduling and planning applications workshop (SPARK 2019), pp 95–103
Venturelli D, Do M, Rieffel EG, Frank J (2017) Temporal planning for compilation of quantum approximate optimization circuits. In: Proceedings of the twenty-sixth international joint conference on artificial intelligence (IJCAI 2017), pp 4440–4446
Zhang YJ, Mu XD, Liu XW, Wang XY, Zhang X, Li K, Wu TY, Zhao D, Dong C (2022) Applying the quantum approximate optimization algorithm to the minimum vertex cover problem. Appl Soft Comput 118:108554