Applied Soft Computing 144 (2023) 110456

Contents lists available at ScienceDirect

Applied Soft Computing

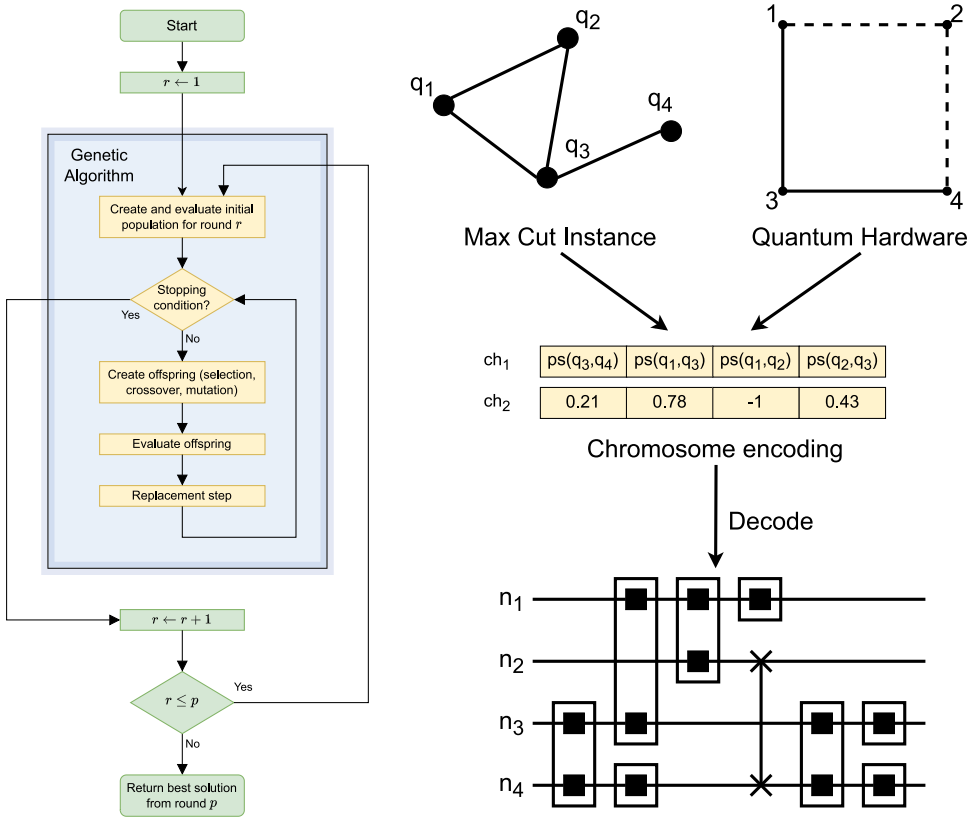
journal homepage: www.elsevier.com/locate/asoc

New coding scheme to compile circuits for Quantum Approximate Optimization Algorithm by genetic evolution

Lis Arufe [a], Riccardo Rasconi [b], Angelo Oddi [b], Ramiro Varela [a,*], Miguel A. González [a]

[a] Department of Computing, University of Oviedo, Campus of Gijón, 33204, Gijón, Spain
[b] Istituto di Scienze e Tecnologie della Congnizione, Consiglio Nazionale delle Ricerche, Via S. Martino della Battaglia, 44, Rome, Italy

GRAPHICAL ABSTRACT



ARTICLE INFO

Article history:
Received 12 February 2023
Received in revised form 29 April 2023

ABSTRACT

Compiling quantum circuits on target quantum hardware architectures is one of the key issues in the development of quantum algorithms, and the related problem is known as the Quantum Circuit Compilation Problem (QCCP). This paper presents a genetic algorithm for solving QCCP instances for Quantum Approximate Optimization Algorithms (QAOA). In particular, such instances represent

* Corresponding author.
E-mail addresses: arufelis@uniovi.es (L. Arufe), riccardo.rasconi@istc.cnr.it (R. Rasconi), angelo.oddi@istc.cnr.it (A. Oddi), ramiro@uniovi.es (R. Varela), mig@uniovi.es (M.A. González).

https://doi.org/10.1016/j.asoc.2023.110456

Accepted 19 May 2023
Available online 25 May 2023

Keywords:
Quantum circuit compilation
Scheduling
Makespan
Optimization
Genetic algorithm

quantum circuits for the resolution of both MaxCut and Graph-Coloring combinatorial problems. The presented algorithm represents a significant improvement over an already existing genetic algorithm called Decomposition Based Genetic Algorithm (DBGA), and is characterized by a completely new coding scheme that allows to reduce the number of *SWAP* gates introduced in the decoding step, consequently reducing the circuit depth. After providing a description of the problem, this paper presents the newly produced genetic algorithm (termed DBGA-X) in detail, especially focusing on the new coding/decoding scheme. Subsequently, a set of results will be presented that demonstrate the superior performance of the new method compared with the results obtained from recent literature against the same benchmark. In addition, new benchmarks characterized by larger quantum architectures and by a higher number of compilation passes are proposed in this paper, to the aim of testing the scalability of the proposed method in more realistic scenarios.

1. Introduction

Quantum computing is a promising technology based on the laws of quantum mechanics, which allows to tackle problems that are extremely hard for classical computers. Together with the development of this technology, the researchers are devising new quantum algorithms that try to outperform their classical counterparts. For example, in [1] a quantum algorithm to solve the problem of logical equivalence is proposed, and the authors claim that it may take exponentially less time than the classical deterministic computation. Just as classical computing entails the execution of classical algorithms, quantum computing entails the execution of quantum algorithms (i.e., quantum *circuits*) for the resolution of the problems.

This work focuses on *gate-based* quantum circuits, i.e., quantum algorithms that are composed of elementary quantum operations (called *quantum gates* or *qgates*) that can be considered as the quantum counterpart of the logical gates in classical computing. As the classical logical gates operate on bits, the quantum gates operate on *quantum bits* (*qubits*). At every instant during the computation, each qubit finds itself in a given *quantum state (qstate*), which may be basically seen as the qubit's value. A quantum circuit is in fact a series of qgates that are applied on qubits over time, whose qstates are consequently modified during the circuit's execution.

In particular, the paper revolves around the Noisy Intermediate Scale Quantum (NISQ) processors [2], which represent the current state of the art in quantum hardware technology; from the topological perspective, NISQ architectures can be described as undirected weighted graphs whose nodes represent the quantum processor's qubits and the edges represent its physical connectivity.

Fig. 1 shows six quantum chip designs, each characterized by a different number of qubits ($N = 4, 8, 21, 40, 72, 127$)[1]. Each qubit is located in a node and identified by an integer. In order to guarantee reliable computation, each quantum circuit must satisfy a number of constraints, imposed by both the hardware and the algorithm. For the purpose of this work, one of the most important constraints to be satisfied is the following: *every 2-qubit qgate (binary quantum gate) may be executed in a pair of qubits only if they are adjacent, i.e. they are connected by an edge*. Since binary qgates need to be applied to adjacent qubits (i.e., *nearest neighbors*), every time the circuit requires the application of a binary qgate to non-adjacent qubits it is necessary to move those qstates so as to satisfy the nearest neighborhood condition. This objective is achieved by planning for, and introducing in the circuit, a proper number of *swap* gates, i.e., particular quantum binary operations whose purpose is to swap the qstates of two adjacent qubits. In other words, the swap gates are used to *move* the qstates around the quantum chip, thus eventually satisfying the nearest neighborhood condition every time it is necessary for binary gate execution. The problem of "distributing" the qstates over a specific architecture to satisfy the aforementioned constraints is known as Quantum Circuit Compilation Problem (QCCP).

The goal of this work is to solve the QCCP by planning and scheduling in the quantum circuit a number of swap gates by means of a genetic algorithm characterized by a novel and very efficient coding/decoding scheme. The produced solutions (i.e., the *compiled* circuits) will distribute the qstates along the physical qubits during the execution, thus ensuring the adjacency of any pair of qubits for all binary quantum gates, before gate application. Unary gates can be applied to any given qubit at any given time, unlike binary gates.

Specifically, this work builds upon the genetic algorithm proposed in [3] and presents the following novel contributions:

1. introduces a completely new coding/decoding scheme that significantly improves the results with respect to the state of the art;
2. conducts a comprehensive experimental study, extending previous results in the literature, by:
   - integrating a set of experiments based on two recent large architectures, the Google Bristlecone (72 qubits) and the IBM Eagle (127 qubits);
   - analyzing quantum circuits composed of up to 5 compilation passes whereas previous literature considered at most 2 passes, thus testing the scalability properties of the method.

Relatively to the last contribution listed above, it is known that in QAOA algorithms the accuracy of the obtained results is directly proportional to the number of compilation passes. Unfortunately though, in the realm of NISQ processors, increasing the number of passes (i.e., increasing the circuit's *depth*) eventually makes the computation unreliable because of decoherence, thus forcing the developer to find the correct balance between the circuit's depth and the required accuracy. In general however, the capability to efficiently compile circuits characterized by many passes remains of high value, in view of less noisy and more reliable future NISQ processors. Also, in [4] it is suggested that the number of compilation passes p should grow with the system size N (e.g., $p \geq \log N$) in order to outperform the best classical algorithms.

The remainder of this paper is structured as follows. In the next Section, an overview of the related literature on the tackled problem is presented. Next, the basics of the QAOA, as well as the description of how it is applied to the MaxCut problem is presented in Section 3. Section 4 is then dedicated to the formal

[1] The four smaller topologies are inspired by Rigetti Computing Inc., whereas the largest two are the Google Bristlecone (72 qubits) and IBM Eagle (127 qubits) architectures.

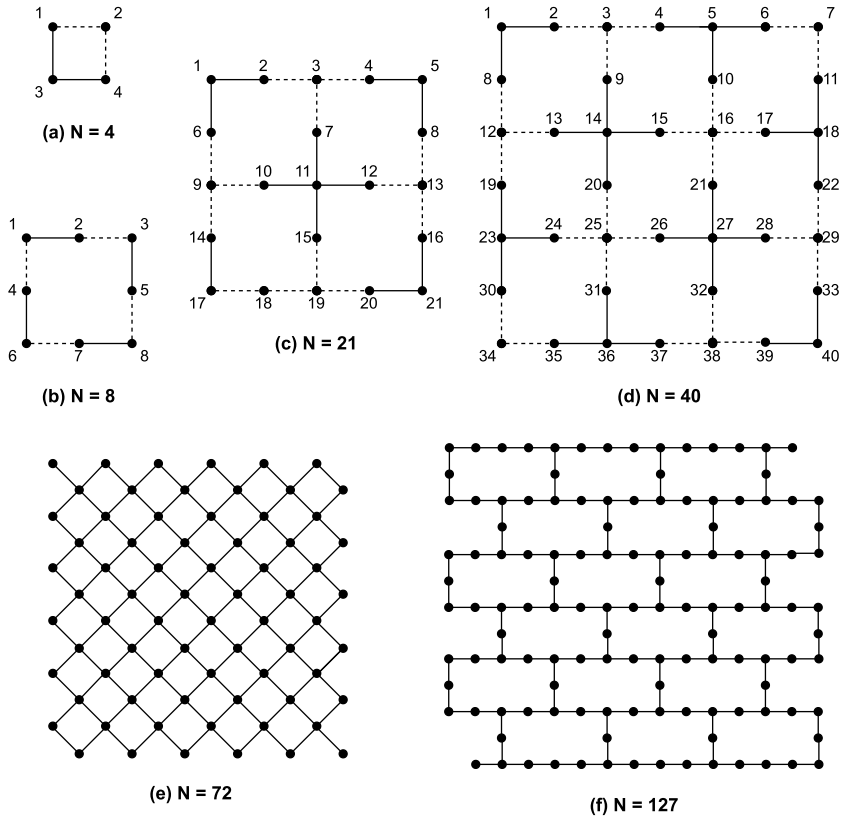


Fig. 1. Six quantum chip designs with different number of qubits.

definition of the QCCP for the MaxCut problem, and the subsequent Section 5 is dedicated to the proposed genetic algorithm. The empirical analysis is described in Section 6, in which the obtained results are compared with those related to the previous state of the art. Finally, in Section 7 the main conclusions are drawn, and some ideas for future research work are advanced.

2. Background literature

Even though quantum circuit compilation has been considered in the literature over more than one decade, the QCCP considered here was initially formulated in [5] and more thoroughly described in [6], where the authors explored the utilization of temporal planners for its resolution, and published the first results against a benchmark set of MaxCut problem instances created on purpose — here referred to as *reference benchmark*. The model described in [5,6] applies the Quantum Approximate Optimization Algorithm (QAOA) to the MaxCut problem [7] on quantum architectures inspired by Rigetti Computing Inc. [8].

Those results were outperformed in [9] by leveraging a heuristic-based greedy randomized search procedure. The same heuristic was subsequently embedded in a genetic algorithm in [10], producing further improvements. In [11], the authors proposed an iterative procedure that utilizes a priority rule-based rollout heuristic, obtaining results that further improved the previous state of the art. Afterwards, the results reported in [11] were further improved in [3], where the authors present a genetic algorithm based on a decomposition of the problem into a number of rounds. The best results on the reference benchmark are currently reported in [12], where the authors develop an ant colony optimization algorithm, introducing a novel pheromone model and leveraging a heuristic-based priority rule to control the iterative selection of the quantum gates to be inserted in the solution.

Moving away from the works that revolve around the reference benchmark, the application of QAOA to the MaxCut problem is also tackled in [13], where the authors study how different graph characteristics correlate with QAOA performance, or in [14], where local classical MaxCut algorithms are compared with QAOA. In [15] the authors prove that for three-regular random graphs, QAOA performance shows improvement by up to two orders of magnitude compared to previous estimates, strongly reducing the performance gap with classical alternatives. Different approaches, such as lookahead heuristics combined with simulated annealing [16], or temporal planning combined with constraint programming [17,18], have also been applied to the QCCP with success.

In this study, the QCCP is tackled using Genetic Algorithms (GA). GAs are a computational technique inspired by the process of natural selection observed in biology, which mimics the biological process of evolution, including selection, crossover, and mutation. GAs are widely used in various fields such as engineering, finance, computer science, and biology, to name a few, as they have proven to be highly effective in solving problems such as resource allocation [19,20], scheduling [21,22], and optimization [23,24]. The empirical evaluation carried out in this paper will prove that, if efficiently encoded, the depth-optimization version of the QCCP can be solved very effectively using GAs, compared to the results in the current literature.

3. QAOA and MaxCut

The Quantum Approximate Optimization Algorithm (QAOA) combines both quantum and classical computation to solve combinatorial optimization problems of the form

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

where $\mathbf{z} = (z_1, \ldots, z_n)$ is a vector of binary decision variables and $C_\alpha(\mathbf{z})$ its clauses. The goal of the problem is finding an assignment of $z_i \in \{+1, -1\}$, $1 \leq i \leq n$, optimizing the number of satisfied clauses.

By promoting each variable z_i to a qubit, clauses $C_\alpha(\mathbf{z})$ are translated into equivalent quantum Hamiltonians $\mathbf{C}_\alpha$. A number of rounds p and two vectors of p angles $\vec{\gamma}, \vec{\beta}$, $0 \leq \beta_i \leq \pi$, $0 \leq \gamma_i \leq 2\pi$, $0 \leq i \leq p$ are then selected. Given qstate $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, the following state is considered starting from its n qubits:

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

where $H_C = \sum_{\alpha=1}^{m} \mathbf{C}_\alpha$ is the problem Hamiltonian and H_B is a mix Hamiltonian of the form $H_B = \sum_{j=1}^{n} \mathbf{X_j}$, where $\mathbf{X_j}$ is the $\mathbf{X}$ Pauli matrix applied to qubit j. An approximate solution to the problem given in Eq. (1) is obtained by measuring the state defined in Eq. (2), whose expected value is

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

The state of the qubits after the transformation will represent a good solution to the problem with high probability in case that the values of $\vec{\beta}$, $\vec{\gamma}$ and p are appropriately selected. An increase in the number of rounds produces an increase in the quality of the solution (with the caveat expressed at the end of Section 1). Classic optimization is used to select $\vec{\beta}$ and $\vec{\gamma}$, for example applying simplex or gradient based optimization [25]. Then, for each candidate $(\vec{\gamma}, \vec{\beta})$, the quantum computer calculates and measures the state of Eq. (2).

3.1. MaxCut problem

Starting from an undirected graph $G = (V, E)$ with a set of nodes $V = \{1, \ldots, n\}$ and a set of arcs E, the objective of the MaxCut problem is to partition V into subsets V_{+1} and V_{-1} so that the number of arcs in E connecting nodes of the two subsets is maximized:

$$\text{minimize:} \sum_{(i,j)\in E} \frac{1}{2}(z_i z_j),\ z_k = \begin{cases} -1 & \text{if } k \in V_{-1} \\ 1 & \text{if } k \in V_{+1} \end{cases} \tag{4}$$

Therefore, for each arc (i, j) there is a Hamiltonian $\mathbf{C}_\alpha$, and the arc only depends on these two variables. It is defined as

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

where $\mathbf{Z_i}$ is the Pauli matrix $\mathbf{Z}$ applied to qubit i (analogously for $\mathbf{Z_j}$). Hence, each of the m components of the problem Hamiltonian H_C corresponds to 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 the $R_{ZZ}(\gamma)$ gate. Every two of these operators sharing a qubit commute and so they may operate in any order, which makes it hard to obtain the overall best schedule.

Given the mix Hamiltonian H_B, each of its components correspond to the unitary operator

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

Operators (6) and (7) will be denoted by $p - s(q_i, q_j)$ and $mix(q_j)$ in the following.

4. The quantum circuit compilation problem

The QCCP is defined by a tuple $P = \langle C_0, L_0, QM \rangle$, where:

- C_0 is the input quantum circuit;
- L_0 is the initial allocation of the qstates on the qubits;
- QM is the quantum hardware topology, represented as a graph.

The input quantum circuit (C_0). The quantum circuit $C_0 = \langle Q, P-S, MIX, \{g_{start}, g_{end}\}, TC_0 \rangle$ contains all the necessary information to solve the MaxCut problem by means of the QAOA paradigm, where: (i) $Q = \{q_1, \ldots, q_N\}$ is the set of qstates, which represent the resources necessary for each gate's execution; (ii) *P-S* and *MIX* are, respectively, the set of *p-s* (phase-separation) and *mix* gate operations, whose mathematical formulation was provided in the previous section; (iii) g_{start} and g_{end} are respectively the initial and final fictitious gates that do not operate on any qstate, and (iv) TC_0 is the set containing the initial precedence constraints acting on the $p-s$, *mix*, g_{start} and g_{end} gates. It should be noted that *p-s* and *mix* gates act on two and one qstates respectively, and therefore the notation $p - s(q_i, q_j)$ and $mix(q_i)$ is used to represent them.

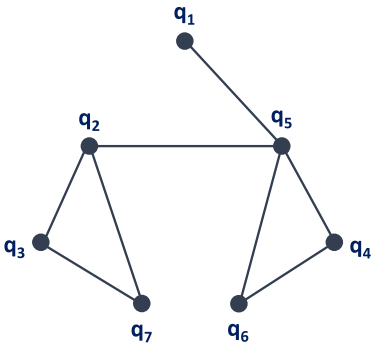
The quantum hardware (QM). The quantum hardware is represented by an undirected graph $QM = \langle V_N, E_c, E_d, \tau_{p-s}, \tau_{mix}, \tau_{swap} \rangle$, where:

- $V_N = \{n_1 \ldots n_N\}$ is the set of qubits belonging to the quantum device;
- $E = E_c \cup E_d$ is a set of undirected edges. Each edge establishes a pair of *adjacent* qubits, thus defining the set of binary gates (*p-s* or a *swap* gates) executable on the quantum machine. The sets E_c (continuous) and E_d (dashed) determine two different durations of the *p-s* gates executed on those edges;
- τ_{p-s} is the duration of each *p-s* gate, and in this study it is equal to 3 when the gate is executed on a continuous edge and equal to 4 when it is executed on a dashed edge;
- τ_{mix} is the duration of a *mix* gate, and in this study it is always equal to 1;
- τ_{swap} is the duration of a *swap* gate, and in this study it is always equal to 2;

The QCCP constraints. Importantly, the QCCP is characterized by the following rules/constraints:

1. each qstate q_i can be processed by at most one gate at any given time;
2. every quantum gate requires the uninterrupted use of all the involved qstates for the gate's whole duration;
3. all gates in *P-S* and *MIX* must follow g_{start} and precede g_{end};
4. the *P-S* and *MIX* sets must be organized in a number of steps that must obey the following ordering: $P - S_1$, MIX_1, $\ldots$, $P - S_p$, MIX_p, where p is the number of rounds (or compilation passes) that are to be executed in the QAOA algorithm;
5. for every round $r = 1, 2, \ldots, p$, all the *p-s* gates belonging to the $P - S_r$ set that require a specific qstate q_i must be processed before all the *mix* gates belonging to the MIX_r set that require the same qstate q_i;
6. all *p-s* gates $\in P - S_r$ may be executed in any order, as they are commutative;
7. for every round $r = 1, 2, \ldots, p-1$ all the *mix* gates belonging to the MIX_r set that require a specific qstate q_i must be processed before all the *p-s* gates belonging to the $P - S_{r+1}$ set that require the same qstate q_i.

Fig. 2 (left) depicts an example graph corresponding to the problem instance no. 1 taken from the $\langle N = 8, u = 90\%,$



P-S_1	MIX_1	P-S_2	MIX_2
p-s(q_1,q_5)	mix(q_1)	p-s(q_1,q_5)	mix(q_1)
p-s(q_2,q_3)	mix(q_2)	p-s(q_2,q_3)	mix(q_2)
p-s(q_2,q_5)	mix(q_3)	p-s(q_2,q_5)	mix(q_3)
p-s(q_2,q_7)	mix(q_4)	p-s(q_2,q_7)	mix(q_4)
p-s(q_3,q_7)	mix(q_5)	p-s(q_3,q_7)	mix(q_5)
p-s(q_4,q_5)	mix(q_6)	p-s(q_4,q_5)	mix(q_6)
p-s(q_4,q_6)	mix(q_7)	p-s(q_4,q_6)	mix(q_7)
p-s(q_5,q_6)		p-s(q_5,q_6)	

Fig. 2. A graph with 7 nodes representing an example MaxCut problem instance. Each particular qstate q_i is associated with a node. If $p = 2$ compilation passes are considered, the *p-s* and *mix* quantum gates that have to be executed are listed in the right side of the figure.

$p = 2\rangle$ subset of the reference benchmark (see Section 6.1). As clearly shown in the picture, the presented instance entails two compilation passes, each characterized by a list of $p - s(q_i, q_j)$ gates and a list of $mix(q_i)$ gates.

A solution to the problem is represented by a tuple $S = \langle SWAP, TC\rangle$ that extends the initial circuit C_0, where:

- *SWAP* is a set of *swap* gates that must be added to ensure the adjacency constraints necessary for executing the *p-s* gates;
- *TC* contains a number of additional precedence constraints for each qstate q_i such that a total order is imposed among the set of gates requiring q_i. These precedence constraints must enforce that qstates (q_i, q_j) of all *p-s* and *swap* gates in the circuit are allocated on adjacent qubits.

A *swap* gate swaps the qstates of two adjacent qubits by implementing the following operator:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

The makespan of a solution is defined as the maximum completion time of all gates in S.

Alternatively, a solution to the problem can be represented using an *activity-on-the-node* graph $G_S = (V_S, E_S)$, where $V_S = \{g_{start}, g_{end}\} \cup P - S \cup SWAP \cup MIX$ and E_S determines the partial order of the operations in V_S according to $TC_0 \cup TC$. In this case, the makespan of the solution is equal to the length of the *critical path*, i.e. the longest path from the g_{start} node to the g_{end} node. Fig. 5(a) depicts an example of a solution graph, where the cost of the arcs corresponds to the duration of the operation the arc originates from.

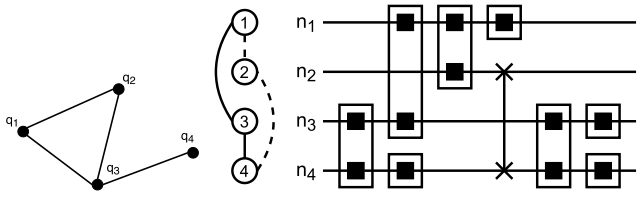
The objective of the QCCP is finding a feasible schedule with minimum makespan, which is an NP-hard problem [26].

QCCP example. The following example illustrates the previous definitions. Consider the quantum chip with $N = 4$ (see Fig. 1(a)) and the graph of Fig. 3(a). To solve this problem the following gates have to be executed: $p - s(q_1, q_2), p - s(q_1, q_3), p - s(q_2, q_3)$ and $p - s(q_3, q_4)$. Note that, for the problem to be properly mapped on a quantum device, the device must contain a number of qubits greater than or equal to the number of nodes of the graph to be cut.

Fig. 3(b) depicts a compiled quantum circuit (i.e., a solution) considering only one compilation pass, i.e. $p = 1$. Note that all adjacency constraints are satisfied by adding only one *swap* gate to the solution.

The corresponding solution graph is shown in Fig. 5(a), characterized by a critical path equal to 16. In order to better understand the solution graph, note that every $p - s(q_i, q_j)$ and $swap(q_i, q_j)$ gate has two successor nodes and two predecessor nodes, which correspond to the previous or to the subsequent gates executed on the qstates q_i and q_j, respectively. In case a gate utilizes the same two qstates as its successor or predecessor, a double arc is depicted. Since the *mix* gates only operate on one qstate, they always have only one successor and one predecessor. Lastly, the first gate executed on a qstate has the node g_{start} as predecessor, whereas the last gate executed on a qstate has the node g_{end} as successor.

Fig. 5(b) depicts the Gantt chart of the solution with the start and end times of all the circuit gates, and showing a makespan equal to 16, which corresponds to the highest end time of all gates. Fig. 4, representing a chromosome example of the genetic algorithm, will be described in Sections 5.1 and 5.2.



(a) MaxCut instance. (b) Quantum circuit (right) for the MaxCut instance of Figure 3(a) on the quantum hardware with 4 qubits on the left.

Fig. 3. Example of a MaxCut instance (a) and a possible solution for it considering $p = 1$, represented by a quantum circuit (b). It is assumed that each qstate q_i is initialized on qubit n_i. The *p-s* gates are represented with rectangles, *mix* gates with squares and *swap* gates with single lines. Operations on each qubit over time are represented with the horizontal lines. Depicted on the left is the considered quantum device.

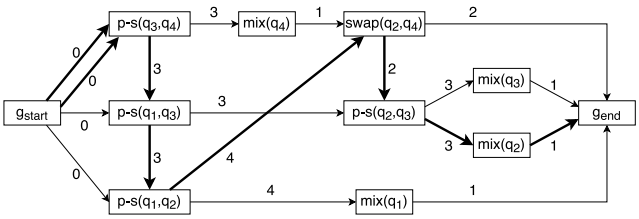
ch_1	ps(q_3,q_4)	ps(q_1,q_3)	ps(q_1,q_2)	ps(q_2,q_3)
ch_2	0.21	0.78	-1	0.43

Fig. 4. A chromosome encoding the solution in Fig. 3(b).

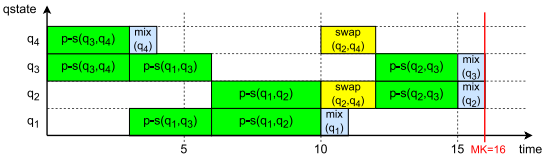
5. The decomposition-based genetic algorithm

This paper borrows the Decomposition-Based Genetic Algorithm (DBGA) proposed in [3], which follows an incremental procedure such that in each iteration $r \in \{1, \ldots, p\}$ it solves the subproblem defined by rounds $1, \ldots, r$. In particular, in each iteration r it starts from solutions to the subproblem defined by rounds $1, \ldots, r-1$ and adds the $p-s$ and *mix* gates corresponding to round r, adding *swap* gates whenever necessary.

The main drawback of the proposal of [3] is that, as the problem complexity is so high, the search space is too large and so it



(a) Solution qraph. Its makespan corresponds to the length of the critical path (16), indicated by bold arcs. When a binary qgate takes both inputs from the same node it is depicted a double arc.



(b) Gantt chart that represents the start and end times of operations on each qstate.

Fig. 5. Representation of the solution in Fig. 3(b), the related solution graph (a) and its Gantt chart (b).

is remarkably difficult to find efficient solutions. It can be argued that the drawback comes from the chosen coding scheme, which allows a *p-s* gate to be scheduled in any part of the quantum chip. In this paper each *p-s* gate is restricted to be scheduled in a shortest path between the two corresponding qstates. Therefore there is less flexibility as a lower number of solutions are allowed, in fact the method may even lose the possibility of finding the optimal solution. On the other hand solutions are in average more efficient, and that allows the final performance of the algorithm to highly improve.

Another advantage with respect to the DBGA of [3] is that the heuristic initialization of the chromosomes proposed in that paper is not needed anymore. The search space reduction produced by the new coding and decoding scheme results in every chromosome having similar quality as those heuristically initialized, thus overall simplifying the genetic algorithm.

The following subsections detail the components of the proposed method, termed DBGA-X.

5.1. The coding scheme

A chromosome for round r is a triplet (sg_{r-1}, ch_1, ch_2), where sg_{r-1} is a solution graph to subproblem $1, \ldots, r-1$, ch_1 is a permutation of the set of *p-s* gates of round r, and ch_2 contains the swap insertion strategy for each *p-s* gate of ch_1. The main difference with respect to the algorithm proposed in [3] lies in ch_2, in which now each position can be either a float number in the interval $[0.0, 1.0)$ (when using strategy *MP*), or a -1 (when using strategy *EST*). Both strategies will be detailed in Section 5.2. Some example chromosomes can be seen in Fig. 8 or in Fig. 4.

5.2. The decoding algorithm

In order to build the schedule represented by a chromosome, the decoding algorithm iterates over the chromosome genes (see Algorithm 1).

For a gate $p-s(q_i, q_j)$ in the position k in ch_1 ($ch_1(k)$), whose qstates are in the qubits $\{n(q_i), n(q_j)\}$, the call to *SelectSwaps* calculates the swap gates determined by the value contained in $ch_2(k)$ to move the qstates towards the pair of adjacent qubits $\{d(q_i), d(q_j)\}$, where the gate will be scheduled. Details of the *SelectSwaps* procedure are described in the following.

Require: A chromosome (sg_{r-1}, ch_1, ch_2)
Ensure: Solution graph sg_r for subproblem defined by rounds $1, \ldots, r$
  $sg_r \leftarrow sg_{r-1}$;
  for k = 1 to $|ch_1|$ **do**
    // $ch_1(k) = p\text{-}s(q_i, q_j)$,
    // $ch_2(k) =$ *X-value (swap insertion strategy)*
    // q_i and q_j are in the qubits $n(q_i)$ and $n(q_j)$)
    $Swaps \leftarrow SelectSwaps(sg_r, ch_1(k), ch_2(k))$;
    for each $sw \in Swaps$ **do**
      Insert swap gate sw in sg_r;
    end for
    // q_i and q_j are in adjacent qubits $d(q_i)$ and $d(q_j)$)
    Insert gate $p\text{-}s(q_i, q_j)$ on qubits $(d(q_i), d(q_j))$ in sg_r;
  end for
  Insert one *mix* gate on all qubits that hold a qstate;
  return Solution graph sg_r;

Algorithm 1: Decoding algorithm.

Given the inputs $p-s(q_i, q_j) = ch_1(k)$ and $X = ch_2(k)$, a set of swap gates are introduced to move concurrently q_j towards q_i and q_i towards q_j though one of the shortest paths on QM between $n(q_i)$ and $n(q_j)$. Two swap insertion strategies are proposed in this work:

1. If $X \in [0.0, 1.0)$ strategy *MP* (Meeting Point) is considered. Let $dist_{ij}$ be the distance between $n(q_i)$ and $n(q_j)$ through a shortest path; clearly, there exist exactly $dist_{ij}$ possible ways the qstates q_i and q_j can become adjacent along such path, each corresponding to a different swap insertion strategy. Fig. 7 shows the swap insertions that would be produced by *X*-values in different subintervals for an example with $dist_{ij} = 9$. In the figure, the available swap insertion strategies are numbered from 1 to $dist_{ij}$. Hence, as $X \in [0.0, 1.0)$, the related swap insertion strategy is $z = floor(X \cdot dist_{ij}) + 1$, which corresponds to making $dist_{ij} - z$ moves from $n(q_i)$ towards $n(q_j)$, and $z-1$ moves from $n(q_j)$ towards $n(q_i)$.
2. On the other hand, if $X = -1$ strategy *EST* (Earliest Starting Time) is used, in which the swap insertion consists in moves through a shortest path depending on the earliest insertion time of the swap gate. This means that, each time a swap is inserted, in order to decide between the two possible moves (i.e. to move $n(q_i)$ towards $n(q_j)$ or move $n(q_j)$ towards $n(q_i)$) it is chosen the swap gate that can start in an earlier time in the current partial schedule. Notice that, since the destination qubits $d(q_i)$ and $d(q_j)$ are in a shortest path between the original qubits $n(q_i)$ and $n(q_j)$, the swap gates may be inserted independently from $n(q_i)$ and $n(q_j)$.

Both swap-insertion strategies *MP* and *EST* have to select which shortest path to use, in case there are more than one. The following procedure is applied: every time a swap gate is inserted, if there are several possible shortest paths for reaching the destination, it is chosen the swap with the earliest ending time of all the possibilities at that point in time (i.e. only looking at the current swap insertion). Another possibility would be to take into account the complete path, but that would be much more computationally expensive, particularly if there are many possible shortest paths.

Looking at the quantum hardware in Fig. 3(b) (left) and the MaxCut problem instance in Fig. 3(a), one feasible chromosome is the one shown in Fig. 4. From this chromosome, the decoding algorithm will produce the solution represented in Figs. 3(b) (right), 5(a) and 5(b). In this example, the $X-$ values in the first

three positions in ch_2 are irrelevant as the qubits of the first three p-s gates in ch_1 are already in adjacent positions.

In the following it is described the procedure depending on the value of the last component of ch_2:

- If the X-value is in the range [0.0, 0.5) (as is in the depicted example, which is $X = 0.43$), then strategy *MP* is applied and the decoding algorithm inserts a swap gate in order to move n_2 towards n_3. There are two possible shortest paths: through n_1 or through n_4. In order to select one, the resulting end time after each insertion is checked; in this case there is a tie (both would end at time 12) so any of them can be chosen. Ties are broken by selecting the higher number (n_4 in the example). Therefore, the swap gate is inserted between n_2 and n_4, obtaining the depicted solution of Fig. 5(b) with makespan 16.
- If the X-value is in the range [0.5, 1.0) strategy *MP* is also applied, but in this case the decoding algorithm would move n_3 towards n_2. There are, again, two possible shortest paths. However in this case the swap gate between n_3 and n_1 would end at time 12, but the swap gate between n_3 and n_4 would end at time 8, so this last one is chosen and inserted obtaining a solution with makespan 15, i.e. better than the one of Fig. 5(b) (notice that in this case the $p - s(q_2, q_3)$ gate would have a duration of 4, unlike that of Fig. 5(b)). The resulting solution is represented in Fig. 6.
- If the X-value is -1, then strategy *EST* is applied. All the possible swap gates on a shortest path between n_3 and n_2 are checked, and that with the lowest starting time is selected. Hence, the swap gate between n_3 and n_4 is inserted, as it can start at time 6, whereas the other possibilities start at time 10, obtaining again a solution with makespan 15 (see Fig. 6).

5.3. General structure of DBGA-X

The proposed Decomposition-Based Genetic Algorithm, termed DBGA-X, uses a similar structure than that presented in [3]. Its flow chart is depicted in Fig. 9. It is assumed that qstate q_i starts in qubit n_i for $i = 1, \ldots, N$. Initial solution graphs SG_0 of the solutions of fictitious round 0 are trivial graphs $g_{init} \rightarrow g_{end}$. Then the method iterates on the number of compilation passes $r = 1, \ldots, p$, calling a genetic algorithm in each iteration r in order to extend to round r the solutions of round $r-1$. The output of DBGA-X is the best solution returned by the genetic algorithm run in round p.

The genetic algorithm called in each iteration r of DBGA-X starts from a set of *popSize* solution graphs SG_{r-1} for subproblem $1, \ldots, r-1$. Firstly *popSize* chromosomes (sg_{r-1}, ch_1, ch_2) are built, one for each sg_{r-1} in SG_{r-1}. In those chromosomes ch_1 is a random permutation of the $p - s$ gates of round r, whereas ch_2 is created in the following way: for each particular gene it is randomly decided between strategy *MP* (i.e. a random float value in [0.0, 1.0)) or strategy *EST* (-1), depending on a parameter *probStrategyMP* that controls the probability of selecting strategy *MP*. In Section 6.2 the influence of this parameter in the overall results is analyzed.

The initial population of chromosomes is evaluated, and then the genetic algorithm iterates until a stop condition is met. In each generation, all chromosomes are shuffled in random pairs, which are then applied the crossover and mutation operators with probabilities P_c and P_m respectively. The proposed method for recombination is the extension of the partial mapping crossover (PMX) introduced in [3] and illustrated in Fig. 8. The procedure used in the mutation operator is as follows: each position of the ch_2 part of each offspring chromosome is modified by choosing between strategy *MP* or *EST* depending on parameter *probStrategyMP*, and then if strategy *MP* is selected a random number in the range [0.0, 1.0) is generated.

Then the new solutions are evaluated, and *popSize* chromosomes are selected for the following generation by taking the best two chromosomes from each pair of parents with their corresponding two offspring.

The termination condition is a maximum number of consecutive generations without improving the best solution found so far. When it is met, the genetic algorithm returns a set of *popSize* solution graphs SG_r. These solution graphs represent a solution to the full problem if $r = p$, or will be the input for the next iteration of DBGA-X if $r < p$.

In [3] the interested reader can find further details on the algorithm.

6. Experimental study

The experimental study is organized as follows. Section 6.1 describes the benchmark set. Then Section 6.2 gives the running parameters of DBGA-X, analyzes the influence of parameter *probStrategyMP* and confirms the efficiency of the decomposition approach considering different instance sizes and number of compilation passes. A comparison with state-of-the-art methods is performed in Section 6.3.

6.1. Benchmark set

The reference benchmark [5] is publicly available in the web[2]. It has instances that consider quantum chips of different number of qubits ($N = \{4, 8, 21, 40\}$), utilization levels ($u = \{90\%, 100\%\}$) and number of compilation passes ($p = \{1, 2\}$). The utilization level refers to the maximum percentage of qstates of the quantum chip that are used in the instance. For each combination of $\{N, p, u\}$ the reference benchmark has 50 instances, each representing a graph to be partitioned by the *MaxCut* process. This work focuses on instances with $u = 100\%$ and $p = 2$, for being the most difficult from the reference benchmark.

Recently, larger hardware architectures have been developed, and so this work proposes new instances of size $N = 72$ and $N = 127$ (see Fig. 1) using the same methodology as that introduced in [5]. Also, the instances of the reference benchmark were extended to perform up to $p = 5$ compilation passes.

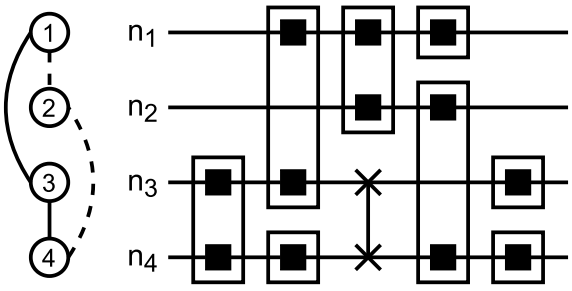
6.2. Analysis of DBGA

DBGA-X is implemented in C++ and run in 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.

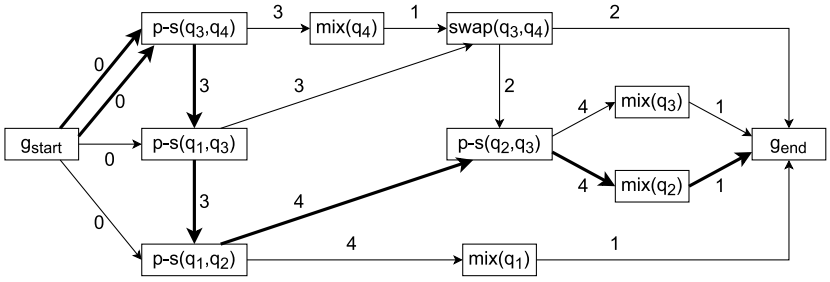
A preliminary parameter analysis was performed, finding that a reasonable configuration for DBGA-X is the following: population size of 1000 chromosomes, crossover rate $P_c = 100\%$ and mutation rate $P_m = 0.05\%$ for each position of the chromosome. Stop condition for each compilation pass is set to 200 consecutive generations with no improvement of the best solution. In this way similar or lower computation times than previous methods of the literature are obtained (see Section 6.3).

Some experiments are performed in order to test the proposed swap insertion strategies on instances of several sizes, considering both 2 and 5 compilation passes. In particular five values for the parameter *probStrategyMP* are tested: 0%, 25%, 50%, 75% and 100%, in order to assess the best combination of both strategies, or if one strategy is strictly better than the other.

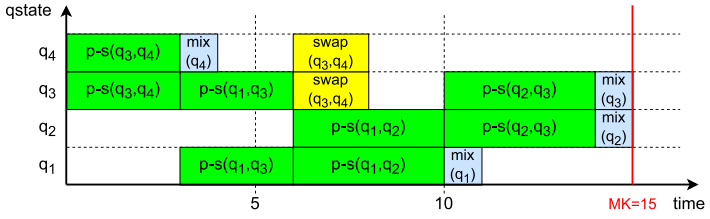
[2] https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_data.zip



(a) Quantum circuit and quantum hardware.



(b) Solution graph, whose makespan (15) corresponds to the length of the critical path.



(c) Gantt chart that depicts operations over time on each qstate.

Fig. 6. Another solution for the MaxCut instance shown in Fig. 3(a), represented by a quantum circuit (a), the related solution graph (b) and its Gantt chart (c). This solution would be obtained if the fourth position of ch_2 of the chromosome of Fig. 4 is either -1 or in the range [0.5, 1.0).

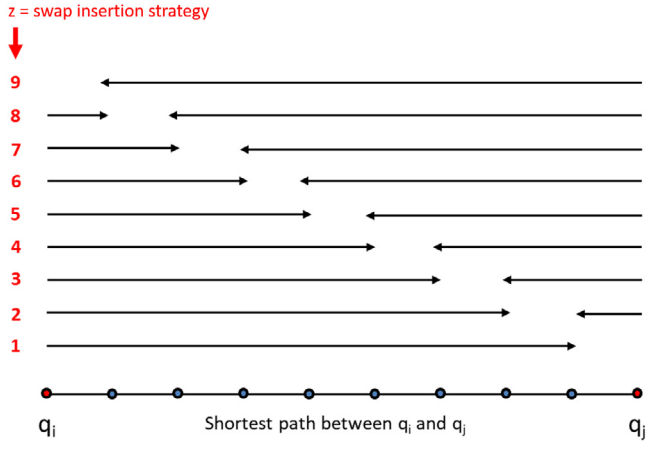


Fig. 7. Illustration of the strategy *MP* for inserting swaps. In this example the distance $dist_{ij}$ between the qstates is equal to 9. The picture enumerates all the possible $dist_{ij}$ swap insertion strategies through which the two qstates can be made adjacent along that shortest path.

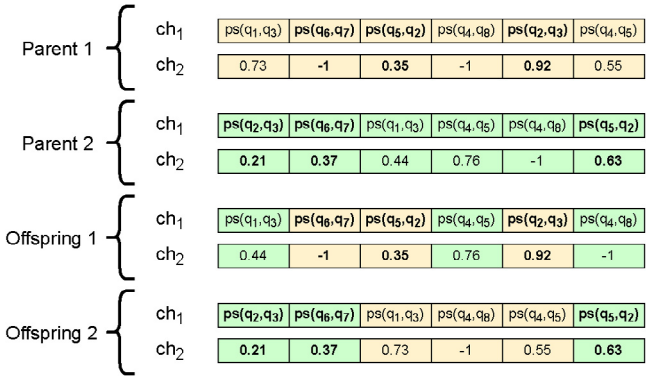


Fig. 8. Crossover operator: in the first offspring, a random selection of $p-s$ gates with their associated X-values is chosen from the first parent and placed in the same positions. Remaining positions are filled with the rest of $p-s$ gates, preserving the relative order they have in the second parent, and using its X-values. In order to create the second offspring the role of the parents is reversed.

The results show that *probStrategyMP* = 100% obtains the worst performance. Then, *probStrategyMP* = 0% and *probStrategyMP* = 75% are both significantly better than *probStrategyMP* = 100%, but the differences between 0% and 75% are not statistically significant. Finally, the best configurations are *probStrategyMP* = 25% and *probStrategyMP* = 50%, which are both statistically better than all others, but the differences between 25% and 50% themselves are not statistically significant. In summary, *probStrategyMP* = 50% is selected as the best option, as it is slightly better than 25% (although the differences are not statistically significant) and significantly better than 0%, 75% and 100%. For

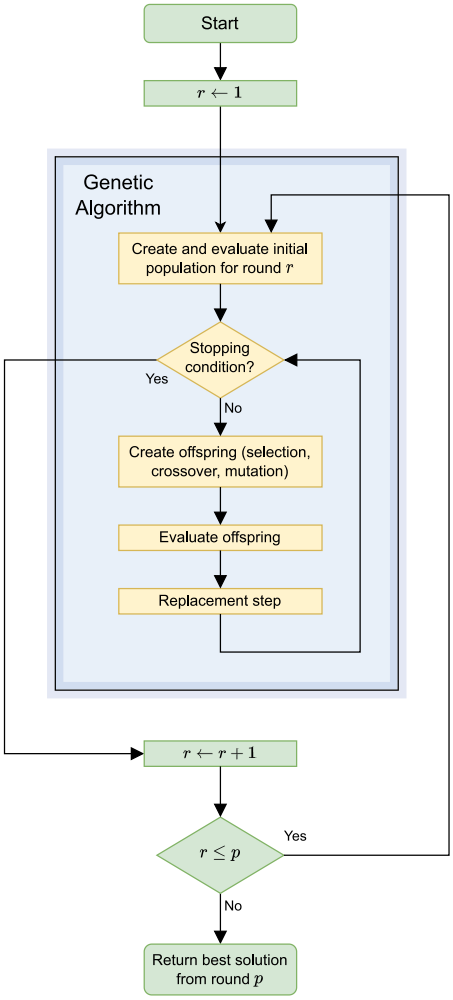


Fig. 9. Flow chart of DBGA-X.

reference, the p-values of a Wilcoxon signed rank test are the following: 0.0008646 vs. 0%, 0.4164 vs. 25%, 0.005459 vs. 75% and 0.00000001852 vs. 100%.

The efficiency of the decomposition approach was proven in [3], but this work extends that study to include larger instances and additional compilation rounds, in order to see its scalability. To this end, two variants of DBGA-X are run on a subset of the instances, in particular the first 10 instances of each subset with $u = 100\%$, considering $p = \{2, 5\}$ and $N = \{8, 21, 40, 72, 127\}$. The first variant does not decompose the problem in p rounds, but solves it all at once, while in the second variant the problem decomposition is used. The results of the comparison are reported in Table 1, showing the average makespan in 10 runs, the standard deviation, and the average computational time taken in a single run.

At a first glance it can be seen that the decomposition scheme obtains better makespan (values in bold indicate the lowest average in each instance), with usually lower standard deviation and lower computation times. In fact, it performs better and better as the problem difficulty increases, either by considering larger chip sizes or by increasing the number of compilation passes. For $N = 8$ the differences are very small, in fact in many cases the version without decomposition produces slightly better results. But when the problem difficulty increases there are makespan reductions: 3.3% average makespan reduction when considering $N = 21$ instances with $p = 2$ passes, and as much as 43.6% average reduction in the most difficult instances, i.e. those with $N = 127$ and $p = 5$ passes. Not only the decomposition scheme produces better results, but it also takes smaller computational time. This is because when solving all the problem at once it takes much longer to decode a chromosome, as all gates from all rounds have to be scheduled each time. Therefore it can be concluded that the large complexity of the problem makes it very difficult to solve and so its decomposition into smaller pieces, as already suggested in [3], is well motivated.

Interestingly, the makespan does not increase proportionally to the number of compilation passes, as it can be seen that makespans considering $p = 5$ are less than 250% higher than those with $p = 2$, as intuitively should be. It seems that the reduction is larger and larger as the instance size increases: in $N = 8$ instances the makespan increases by only 238% when performing $p = 5$ (with respect to $p = 2$), whereas in $N = 127$ instances the makespan increases by 183%. It can be concluded that DBGA-X is able to concurrently execute gates from different compilation passes in different parts of the quantum chip, and this leads to solutions with lower makespan, particularly in the largest quantum chips.

Fig. 10 shows two executions of DBGA-X with and without decomposing the problem in rounds, and two executions of the original DBGA proposed in [3] with and without decomposing the problem in rounds. The experiments were performed using the same computational time, for the instance no. 1 of the subset characterized by $N = 72$, $u = 100\%$ and $p = 5$. In Figs. 10(a) and 10(c) four large jumps in the average and best makespan can be noticed, which are caused by the algorithms switching from one round to the next. On the other hand, in Figs. 10(b) and 10(d) the full problem is solved at once and it can be observed that the algorithms are able to perform a lower number of generations in the same time, and also that the resulting makespan is considerably worse. Although the convergence patterns obtained by DBGA are comparable to those of DBGA-X, it can be seen that much worse makespan values are reached, even if the number of generations is higher. The more chaotic average values in the original DBGA are due to the diversification step performed in that method, which is not present in the new DBGA-X.

The new instances of size $N = 72$ and $N = 127$ and the full results and best schedules found by DBGA-X for all instances considered are publicly available on the web, in order to promote future research and comparisons with this proposal[3].

6.3. Comparison to the state of the art

The previous best state-of-the-art method is the ant colony optimization algorithm (QCC-ACO) presented in [12]. The authors reported generally better results (particularly in the larger $N = 40$ instances) than those of the decomposition-based genetic algorithm (DBGA) described in [3], which in turn obtained generally better results than those of the rollout heuristic (RH) presented in [11], which are mostly better than those obtained by the genetic algorithm (GA) described in [10].

QCC-ACO [12] was implemented in Java and run on an Intel Xeon E312 machine with 16 GB RAM. Its running time was limited to 60 s for $N = 8$ instances and to 300 s for the $N = 21$ and $N = 40$ instances. DBGA [3] was implemented in C++ and run in a Intel Core i5-7400 CPU at 3.00 GHz with 16 GB RAM and its average running time is about 5 s in $N = 8$ instances, 40 s in $N = 21$ instances and 150 s in $N = 40$ instances. RH [11] was implemented in Matlab and run on Intel i7 processor, 16 GB RAM and Windows 7 operating system. For $N = 8$ the execution time was limited to a maximum of 60 s, whereas for larger instances the maximum was set to 300 s. GA [10] was implemented in Java and given 60, 300 and 600 s for instances of size 8, 21 and 40 respectively.

[3] Repository section in http://www.di.uniovi.es/iscop/

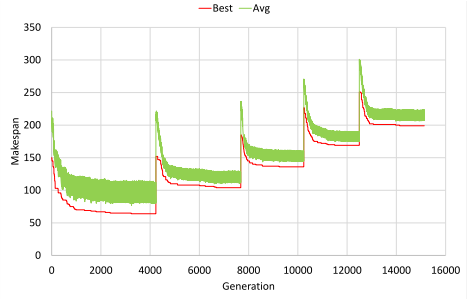
Table 1
Results of DBGA-X compared with a version without problem decomposition, considering different instance sizes $N = \{8, 21, 40, 72, 127\}$ and different number of compilation passes $p = \{2, 5\}$.

Instance	$p = 2$			$p = 2$			$p = 5$			$p = 5$		
	DBGA-X (no decomposition)			DBGA-X			DBGA-X (no decomposition)			DBGA-X		
	Average	Std.dev.	Time	Average	Std.dev.	Time	Average	Std.dev.	Time	Average	Std.dev.	Time
						$N = 8$ **Instances**						
1	**34.0**	0.0	2.5	35.0	0.0	1.1	**81.3**	1.25	7.7	82.7	0.48	3.3
2	**33.0**	0.0	2.7	35.0	0.0	1.1	**78.7**	0.48	7.9	86.0	0.0	3.4
3	**31.0**	0.0	2.7	**31.0**	0.0	1.1	**73.7**	0.95	8.7	73.9	0.32	3.4
4	**32.0**	0.0	2.6	**32.0**	0.0	1.1	**77.1**	0.32	8.3	78.7	2.06	3.5
5	**27.0**	0.0	2.6	27.3	0.48	1.1	**62.6**	0.52	7.8	62.9	0.32	3.4
6	**33.1**	0.32	2.6	33.9	0.32	1.1	**80.2**	1.23	8.0	80.4	1.71	3.3
7	**30.0**	0.0	2.6	**30.0**	0.0	1.1	69.3	0.67	7.9	**69.0**	0.0	3.3
8	**31.7**	0.48	2.8	32.0	0.0	1.1	**74.5**	0.71	6.9	77.0	0.0	3.3
9	**35.0**	0.0	2.7	**35.0**	0.0	1.1	84.4	1.17	9.3	**83.0**	0.0	3.5
10	35.3	0.82	2.9	**34.0**	0.0	1.2	85.0	1.05	10.0	**82.0**	0.0	3.7
#best	9			5			7			3		
						$N = 21$ **Instances**						
1	46.9	1.29	13.1	**45.4**	0.52	5.7	107.3	3.06	75.3	**103.4**	2.22	17.3
2	48.8	1.32	13.5	**48.1**	0.32	5.4	120.8	3.71	83.5	**112.7**	1.83	17.2
3	43.1	2.28	12.6	**41.5**	1.18	6.0	94.0	3.4	66.6	**88.0**	3.46	18.6
4	44.2	0.92	13.3	**43.7**	0.82	6.2	103.3	2.91	53.8	**95.9**	2.23	17.3
5	51.5	2.8	15.3	**47.5**	1.65	6.1	117.4	4.43	98.5	**110.8**	4.54	19.1
6	49.0	0.67	12.8	**48.4**	0.7	5.5	113.9	5.53	88.0	**108.0**	1.76	17.0
7	**52.2**	1.32	13.4	54.3	0.67	5.1	119.9	3.31	85.2	**118.7**	3.59	16.5
8	48.1	1.79	12.9	**45.7**	0.67	6.2	103.3	3.02	70.2	**100.5**	2.84	18.0
9	52.8	1.14	13.6	**48.5**	1.84	6.0	119.1	5.17	95.9	**111.1**	3.78	18.6
10	53.5	1.96	15.1	**50.6**	0.7	6.6	124.1	3.41	72.2	**118.2**	2.39	17.3
#best	1			9			0			10		
						$N = 40$ **Instances**						
1	62.6	4.43	43.8	**56.0**	1.41	23.6	131.2	5.03	463.3	**124.3**	3.71	74.2
2	71.7	5.48	46.8	**62.4**	1.43	26.4	146.5	6.87	532.9	**132.6**	4.14	80.9
3	67.5	4.67	54.4	**59.7**	1.64	26.7	144.4	7.95	528.9	**127.0**	3.77	75.5
4	76.0	5.23	46.1	**64.2**	2.7	30.9	146.8	10.05	368.2	**130.8**	4.16	82.0
5	70.2	3.01	49.8	**65.2**	3.33	29.2	145.9	8.16	558.5	**133.3**	5.14	76.2
6	78.4	4.14	48.4	**64.9**	2.13	30.5	157.9	8.77	492.2	**136.5**	3.75	86.0
7	78.7	6.58	48.0	**63.9**	2.77	26.7	158.5	5.87	567.9	**132.7**	3.74	82.7
8	67.6	3.1	48.7	**59.0**	1.05	28.2	143.8	9.74	442.6	**126.3**	3.95	72.4
9	68.7	4.37	47.9	**58.3**	2.06	26.8	143.8	8.13	391.2	**124.8**	2.15	75.7
10	79.4	3.81	47.3	**67.4**	3.72	27.7	170.8	30.81	510.7	**139.2**	5.29	81.2
#best	0			10			0			10		
						$N = 72$ **Instances**						
1	92.8	5.59	168.6	**72.8**	3.01	112.6	203.4	37.13	1487.3	**137.2**	3.61	336.1
2	96.4	5.62	195.6	**76.0**	7.72	119.1	257.5	26.37	1264.2	**143.3**	4.83	336.1
3	96.2	7.94	160.5	**73.4**	2.12	111.1	241.5	42.1	1255.3	**141.6**	4.45	341.5
4	88.9	7.82	171.1	**74.1**	6.12	115.6	176.8	25.73	1775.5	**134.4**	5.64	348.8
5	103.5	4.74	204.5	**81.3**	5.1	119.1	271.1	31.76	1355.5	**158.7**	11.35	349.0
6	97.3	7.94	188.2	**72.5**	5.84	114.2	265.6	36.72	1039.8	**150.7**	4.35	351.7
7	96.1	5.11	187.4	**75.4**	3.72	120.2	219.8	55.07	1787.0	**140.3**	3.95	361.8
8	97.4	7.32	177.1	**76.1**	5.63	114.4	208.7	34.25	1656.0	**142.8**	4.73	338.2
9	90.3	5.25	178.3	**70.8**	3.12	119.8	225.7	24.66	1276.0	**134.4**	3.2	333.9
10	87.3	6.6	161.9	**69.6**	3.75	97.9	200.4	31.86	1558.8	**137.6**	3.98	288.3
#best	0			10			0			10		
						$N = 127$ **Instances**						
1	175.4	12.26	868.8	**141.9**	5.72	445.2	463.6	22.71	4709.9	**259.1**	19.33	1478.5
2	181.9	14.83	897.6	**141.1**	5.47	505.9	422.5	34.78	5175.1	**262.6**	8.13	1688.3
3	191.0	12.71	844.5	**159.8**	6.97	439.3	527.5	74.28	4188.0	**280.2**	11.05	1541.1
4	183.5	12.03	849.7	**148.2**	7.44	471.7	451.6	33.2	4762.7	**269.7**	8.67	1549.7
5	189.5	14.17	861.0	**149.0**	6.72	471.2	502.6	69.84	4861.1	**280.5**	11.09	1657.0
6	175.1	7.02	860.8	**146.4**	11.03	422.8	490.7	64.58	4737.1	**270.8**	14.02	1580.6
7	186.2	10.75	865.1	**147.9**	4.93	474.2	465.3	40.69	4766.1	**265.4**	12.83	1539.1
8	171.1	9.84	893.9	**142.0**	5.68	451.9	481.9	89.27	5134.0	**256.0**	7.47	1603.5
9	186.7	10.03	875.2	**147.7**	7.26	451.9	492.8	43.12	4876.6	**279.8**	10.61	1548.1
10	193.3	13.53	861.5	**145.8**	3.22	480.3	479.2	31.3	4816.1	**265.0**	10.12	1597.0
#best	0			10			0			10		

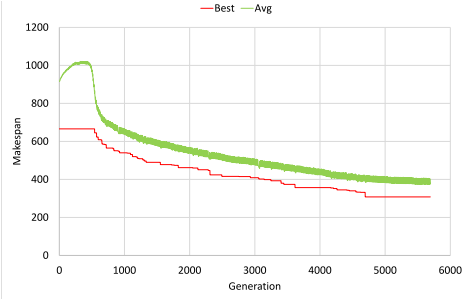
Values in **bold** indicate the lowest average result in each instance.

As it can be seen in Table 1 ($p = 2$ columns), computation times of DBGA-X are lower than those used by previous methods of the literature, although they are not directly comparable to those of QCC-ACO, RH and GA due to differences in target machine and programming language.

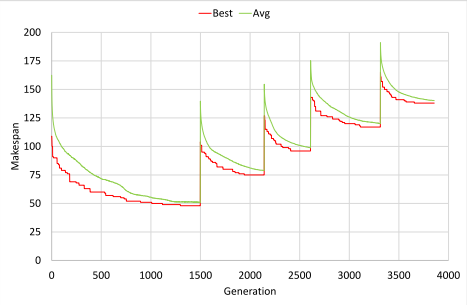
Fig. 11 graphically shows a comparison of the newly proposed DBGA-X and these previous methods. In particular it is shown the percentage reduction in makespan of the average results (in 10 runs) of each method with respect to the GA proposed in [10],



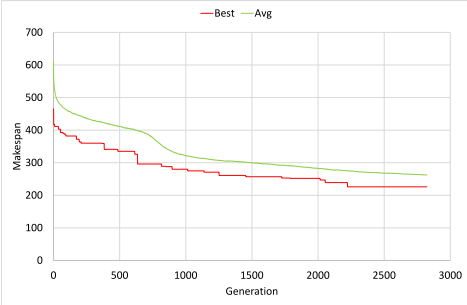
(a) DBGA. The peaks represent the pass from one round to the next one.



(b) DBGA without decomposing the problem in five rounds.



(c) DBGA-X. The peaks represent the pass from one round to the next one.



(d) DBGA-X without decomposing the problem in five rounds.

Fig. 10. Two example convergence graphs of the proposed DBGA-X (in the lower half) with and without decomposition, compared to two convergence graphs of the original DBGA proposed in [3] (in the upper half). Experiments were performed in instance no. 1 of the subset characterized by $N = 72$, $u = 100\%$ and $p = 5$. x-axis represent generations and y-axis represent fitness values. Avg represents the evolution of the average fitness of the population, and Best represents the best fitness of the population.

which is arguably the worst of the five methods (DBGA-X, QCC-ACO, DBGA, RH, GA). The comparison in $N = 8$, $N = 21$ and $N = 40$ instances is presented in Figs. 11(a), 11(b) and 11(c) respectively.

DBGA-X seems superior to other methods in $N = 21$, where it obtains a better average result in 41 of the 50 instances, with an average makespan reduction of 14.95% with respect to GA. The differences are even larger when considering $N = 40$ instances, where it obtains a better average result in 49 of the 50 instances, with an average makespan reduction of 30.95% with respect to GA. In $N = 8$ instances the differences between methods are smaller, but results of DGBA-X are still better, obtaining an average makespan reduction of 5.17% with respect to GA (versus 2.71% of QCC-ACO, 1.68% of DBGA and 2.21% of RH). Regarding best results, DBGA-X established new best solutions for 12 out of 50 of the $N = 8$ instances, for 34 out of 50 of the $N = 21$ instances, and for 47 out of 50 of the $N = 40$ instances.

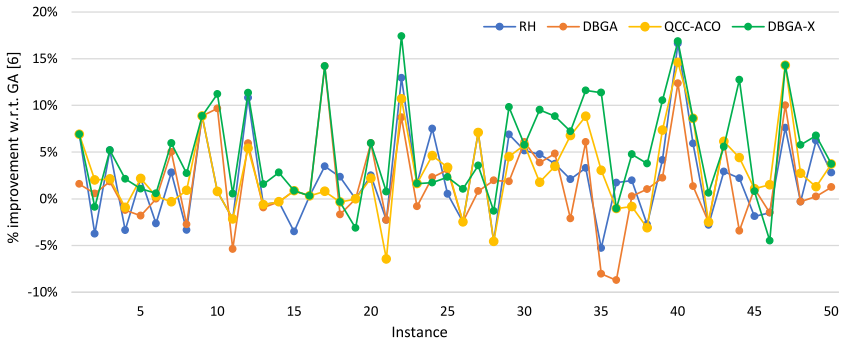
Non-parametric statistical tests are performed in order to validate the comparison. A Shapiro–Wilk test confirms the non-normality of the data, and then paired Wilcoxon signed rank tests are used to compare DBGA-X with the other methods, using 95% confidence level. The p- values obtained with respect to GA are 0.00000003662 in $N = 8$ instances, 3.864e−10 in $N = 21$ instances and 3.883e−10 in $N = 40$ instances. The p-values obtained with respect to RH are 0.000004752 in $N = 8$ instances, 4.13e−10 in $N = 21$ instances and 3.889e−10 in $N = 40$ instances. The p- values obtained with respect to DBGA are 0.000000594 in $N = 8$ instances, 3.887e−10 in $N = 21$ instances and 3.887e−10 in $N = 40$ instances. Finally, with respect to QCC-ACO the p- values obtained are 0.00005814 in $N = 8$ instances, 0.00000007129 in $N = 21$ instances and 5.936e−10 in $N = 40$ instances. Hence, for all instance sizes, the differences in the average result between DBGA-X and the previous methods of the state of the art are statistically significant, although the differences are smaller in the easier $N = 8$ instances.

As an example, Fig. 12 shows the Gantt chart of the best schedules obtained by DBGA-X to the instance no. 22 of the subset characterized by $N = 8$, $u = 100\%$ and $p = 2$, considering different number of compilation passes, $p = 1$, $p = 2$ and $p = 5$. In particular, the schedule with $p = 2$ has a makespan of 37, which is lower than the makespan reported by all previous methods in the literature for the same instance. It can also be observed how an increasing number of compilation passes creates some efficiencies that can improve the overall result, and so the makespan with $p = 5$ is less than 5 times that of $p = 1$, and it is also less than 2.5 times that of $p = 2$. In general, this is due to the fact that the previous compilation passes do perform a sort of re-adjustment of the state that favors the depth optimization for the subsequent passes. This does not occur on the first pass, as in the QCCP problem version the initial state is fixed.

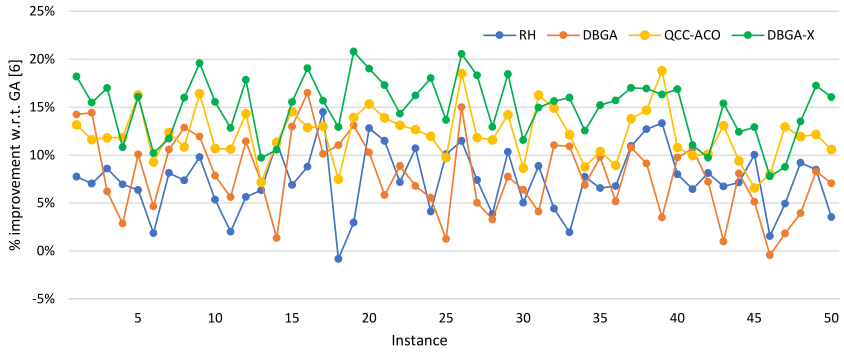
7. Conclusions and future work

An effective genetic algorithm for the Quantum Circuit Compilation Problem (QCCP) was designed based on the Decomposition-Based Genetic Algorithm (DBGA) proposed in [3]. In particular, a new coding/decoding mechanism was devised that reduces the search space by restricting the insertion of *swap* gates to a shortest path between the incumbent qubits. Two strategies for inserting *swap* gates are described, and the proper balance between both was figured out.

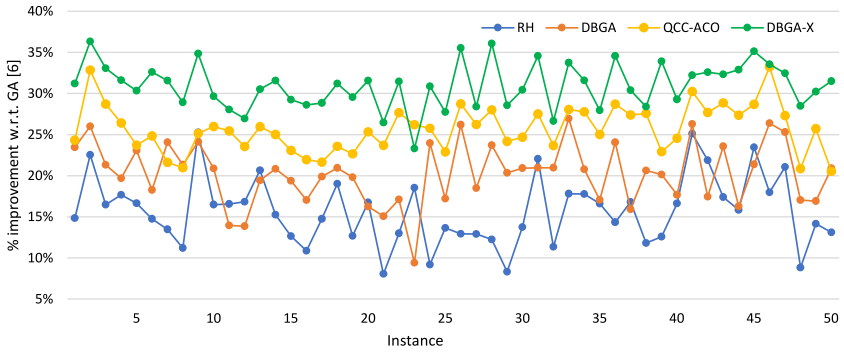
In the experimental study the efficiency of the proposal, termed DBGA-X, is proven by showing how the decomposition approach gives increasing benefits as the chip size and number



(a) N=8 instances



(b) N=21 instances



(c) N=40 instances

Fig. 11. Comparison of DBGA-X with previous state of the art methods RH [11], DBGA [3] and QCC-ACO [12]: percentage improvement in makespan with respect to GA [10] in all 50 instances of each of the (a) N = 8, (b) N = 21 and (c) N = 40 benchmarks.

of compilation passes increase. The proposal improves the performance of previous state-of-the-art methods in most instances of the reference benchmark introduced in [5], which is the most commonly used in the literature. Larger benchmark instances with $N = 72$ and $N = 127$ qubits and $p = 5$ compilation passes are also studied, which are not considered in previous studies but they are certainly more interesting in real environments.

The remarkable performance of DBGA-X compared with previous methods can be attributed to the extreme complexity of the QCCP problem, which motivates reducing the search space to a set of efficient solutions, even if the method may lose the ability to reach the optimal one.

There are many possible avenues for future research. The application of QAOA to other problems, as for example the Graph Coloring Problem [17] or the Minimum Vertex Cover Problem [27], is a subject of further study.

The MaxCut problem can also be tackled more thoroughly and consider the extensions described in [10,18,28,29]. One is the QCCP-I (variable initialization of qstates) which considers the initial location of qstates in the quantum hardware as additional decision variables. Another extension is the QCCP-X (crosstalk constraints), that prohibits executing at the same time two gates on neighboring qubits, in order to avoid possible interferences between qubits.

Multiobjective optimization is an interesting topic, because there are other measures that are interesting to minimize, besides the makespan. As an example, in [30] the authors minimize both the number of gates and the compilation time. In fact, in [31] it is suggested that the minimization of the number of *swap* gates and the circuit's depth might be mutually conflicting objectives, therefore motivating a multiobjective approach. Another example can be found in [32], where the authors consider the gate error rates in order to maximize the circuit's fidelity, while minimizing the number of gates.

Finally, new hardware architectures and other emerging technologies are being developed, as for example the recent IBM Osprey with 433 qubits, and so adapting the solving methods to exploit them is an important line of future work.

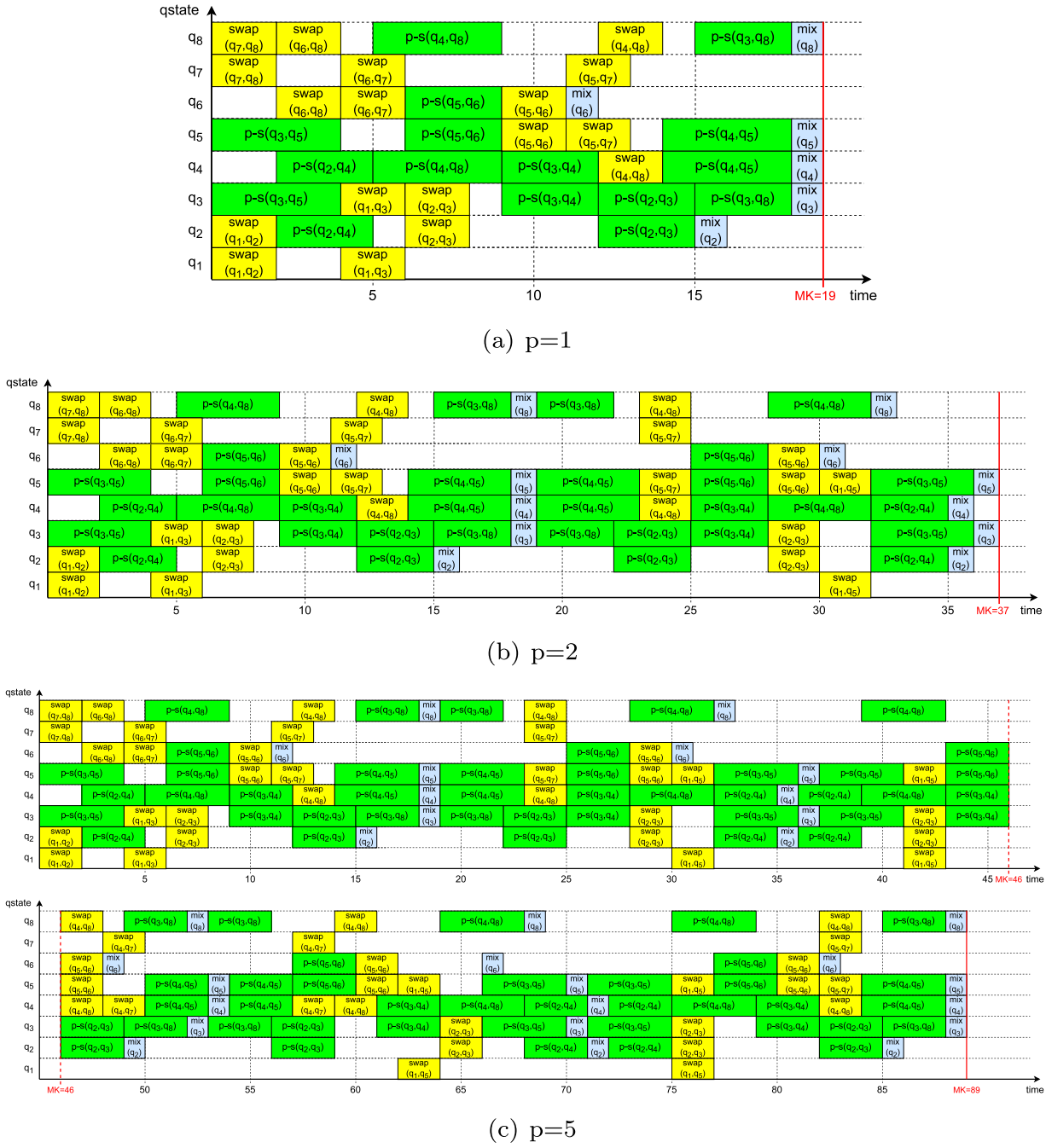


Fig. 12. Gantt representation of the best solutions to the instance no. 22 of the subset characterized by $N = 8$, $u = 100\%$ and $p = 2$, considering different number of compilation passes: (a) $p = 1$, (b) $p = 2$, (c) $p = 5$. In the case with $p = 5$ in order to improve visibility the Gantt is divided in two parts: from time 0 to 46, and from time 46 to 89.

CRediT authorship contribution statement

Lis Arufe: Sotfware, Data curation, Validation, Writing – original draft. **Riccardo Rasconi:** Conceptualization, Writing – review & editing. **Angelo Oddi:** Conceptualization, Writing – review & editing. **Ramiro Varela:** Funding adquisition, Formal analysis, Writing – review & editing. **Miguel A. González:** Conceptualization, Supervision, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ramiro Varela reports financial support was provided by Spanish State Agency of Research. Angelo Oddi, Riccardo Rasconi reports financial support was provided by European Space Agency. Miguel Angel Gonzalez, Lis Arufe reports financial support was provided by Spanish State Agency of Research. The corresponding author is member of the editorial board of Applied Soft Computing

Data availability

The data is available through links showed in the manuscript.

Acknowledgments

This research has been supported by the Spanish Government, AEI under research project PID2019-106263RB-I00. A. Oddi and R. Rasconi were supported by the PNRR MUR project PE0000013-FAIR and ESA, France Contract No. 4000112300/14/D/MRP "Mars Express Data Planning Tool MEXAR2 Maintenance".

References

[1] M. Zidan, S.F. Hegazy, M. Abdel-Aty, S.S. Obayya, Rapid solution of logical equivalence problems by quantum computation algorithm, Appl. Soft Comput. 132 (2023) 109844.
[2] D. Venturelli, M. Do, B. O'Gorman, J. Frank, E. Rieffel, K.E. Booth, T. Nguyen, P. Narayan, S. Nanda, Quantum circuit compilation: An emerging application for automated reasoning, in: S. Bernardini, K. Talamadupula, N. Yorke-Smith (Eds.), Proceedings of the 12th International Scheduling and Planning Applications Workshop, SPARK 2019, 2019, pp. 95–103.
[3] L. Arufe, M.A. González, A. Oddi, R. Rasconi, R. Varela, Quantum circuit compilation by genetic algorithm for quantum approximate optimization algorithm applied to maxcut problem, Swarm Evol. Comput. 69 (2022) 101030.
[4] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, M.D. Lukin, Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices, Phys. Rev. X 10 (2).
[5] D. Venturelli, M. Do, E.G. Rieffel, J. Frank, Temporal planning for compilation of quantum approximate optimization circuits, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, 2017, pp. 4440–4446.
[6] D. Venturelli, M. Do, E. Rieffel, J. Frank, Compiling quantum circuits to realistic hardware architectures using temporal planners, Quantum Sci. Technol. 3 (2) (2018) 025004.
[7] E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem, 2014, arXiv:1412.6062.
[8] E.A. Sete, W.J. Zeng, C.T. Rigetti, A functional architecture for scalable quantum computing, in: 2016 IEEE International Conference on Rebooting Computing, ICRC, IEEE, 2016, pp. 1–6.
[9] A. Oddi, R. Rasconi, Greedy randomized search for scalable compilation of quantum circuits, in: W.-J. van Hoeve (Ed.), CPAIOR 2018: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, Cham, 2018, pp. 446–461.
[10] R. Rasconi, A. Oddi, An innovative genetic algorithm for the quantum circuit compilation problem, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 7707–7714.
[11] S. Chand, H.K. Singh, T. Ray, M. Ryan, Rollout based heuristics for the quantum circuit compilation problem, in: 2019 IEEE Congress on Evolutionary Computation, CEC, 2019, pp. 974–981.
[12] M. Baioletti, R. Rasconi, A. Oddi, A novel ant colony optimization strategy for the quantum circuit compilation problem, in: C. Zarges, S. Verel (Eds.), Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021, in: Lecture Notes in Computer Science, vol. 12692, Springer International Publishing, Cham, 2021, pp. 1–16.
[13] R. Herrman, L. Treffert, J. Ostrowski, P.C. Lotshaw, T.S. Humble, G. Siopsis, Impact of graph structures for qaoa on maxcut, Quantum Inf. Process. 20 (2021) 289.
[14] K. Marwaha, Local classical max-cut algorithm outperforms $p = 2$ qaoa on high-girth regular graphs, Quantum 5 (2021) 437.
[15] J. Larkin, M. Jonsson, D. Justice, G.G. Guerreschi, Evaluation of qaoa based on the approximation ratio of individual samples, Quantum Sci. Technol. 7 (4) (2022) 045014.
[16] X. Zhou, S. Li, Y. Feng, Quantum circuit transformation based on simulated annealing and heuristic search, 2019, arXiv:1908.08853.
[17] M. Do, Z. Wang, B. O'Gorman, D. Venturelli, E. Rieffel, J. Frank, Planning for compilation of a quantum algorithm for graph coloring, 2020, arXiv:2002.10917.
[18] K.E. Booth, M. Do, J.C. Beck, E. Rieffel, D. Venturelli, J. Frank, Comparing and integrating constraint programming and temporal planning for quantum circuit compilation, in: Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, 2018, pp. 366–374.
[19] B. Samuel, J. Mathew, Resource allocation in a repetitive project scheduling using genetic algorithm, IOP Conf. Ser.: Mater. Sci. Eng. 330 (2018) 012098.
[20] S. Kaiafa, A.P. Chassiakos, A genetic algorithm for optimal resource-driven project scheduling, Procedia Eng. (ISSN: 1877-7058) 123 (2015) 260–267.
[21] G.S. Budhi, K. Gunadi, D.A. Wibowo. (, Genetic algorithm for scheduling courses, in: R. Intan, C.H. Chi, H. Palit, L. Santoso (Eds.), Intelligence in the Era of Big Data, ICSIIT 2015, in: Communications in Computer and Information Science, vol. 516, Springer, Berlin, Heidelberg, 2015.
[22] X. Luo, Q. Qian, Y.F. Fu, Improved genetic algorithm for solving flexible job shop scheduling problem, Procedia Comput. Sci. 166 (2020) 480–485.
[23] A.A.M. Zahir, S.S.A. Alhady, W.A.F.W. Othman, A.A.A. Wahab, M.F. Ahmad, Objective functions modification of GA optimized PID controller for brushed DC motor, Int. J. Electr. Comput. Eng. 10 (2020) 2426–2433.
[24] M.F. Ahmad, N.A.M. Isa, W.H. Lim, K.M. Ang, Differential evolution with modified initialization scheme using chaotic oppositional based learning strategy, Alex. Eng. J. 61 (2022) 11835–11858.
[25] M. Fernández-Pendás, E.F. Combarro, S. Vallecorsa, J. Ranilla, I.F. Rúa, A study of the performance of classical minimizers in the quantum approximate optimization algorithm, J. Comput. Appl. Math. 404 (2022) 113388.
[26] A. Botea, A. Kishimoto, R. Marinescu, On the complexity of quantum circuit compilation, in: Eleventh Annual Symposium on Combinatorial Search, SOCS, 2018, pp. 138–142.
[27] Y. Zhang, X. Mu, X. Liu, X. Wang, X. Zhang, K. Li, T. Wu, D. Zhao, C. Dong, Applying the quantum approximate optimization algorithm to the minimum vertex cover problem, Appl. Soft Comput. 118 (2022) 108554.
[28] M. Alam, A. Ash-Saki, S. Ghosh, Circuit compilation methodologies for quantum approximate optimization algorithm, in: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2020, pp. 215–228.
[29] L. Arufe, R. Rasconi, A. Oddi, R. Varela, M.Á. González, Compiling single round qccp-x quantum circuits by genetic algorithm, in: J.M. Ferrández Vicente, J.R. Álvarez-Sánchez, F. de la Paz López, H. Adeli (Eds.), Bio-Inspired Systems and Applications: From Robotics to Ambient Intelligence, Springer International Publishing, Cham, 2022, pp. 88–97.
[30] M.Y. Siraichi, V.F. d. Santos, S. Collange, F.M.Q. Pereira, Qubit. allocation, Qubit allocation, in: Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 113–125.
[31] G. Li, Y. Ding, Y. Xie, Tackling the qubit mapping problem for nisq-era quantum devices, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1001–1014.
[32] S. Niu, A. Suau, G. Staffelbach, A. Todri-Sanial, A hardware-aware heuristic for the qubit mapping problem in the nisq era, IEEE Trans. Quantum Eng. 1 (2020) 1–14.