Meta-Heuristic Algorithms for the Quantum Circuit Compilation Problem

Final Report

Authors: Angelo Oddi[1], Riccardo Rasconi[1], Vieri Giuliano Santucci[1], Marco Baioletti[2], Hamish Beck[3]
Affiliation: [1]ISTC-CNR, [2]University of Perugia, [3]ESA ACT

Date: 23 December 2021

Contacts:

Angelo Oddi
Tel: +39-06-44595-214
Fax: +39-06-44595-243
e-mail: angelo.oddi@istc.cnr.it

Leopold Summerer (Technical Officer)
Tel: +31(0)715654192
Fax: +31(0)715658018
e-mail: act@esa.int

Available on the ACT website
http://www.esa.int/act

Ariadna ID: 21/1201
Ariadna study type: Standard
Contract Number: 4000134995/21/NL/GLC/my

This page intentionally left blank

LIST OF FIGURES

1 INTRODUCTION

Quantum algorithms process information stored in qubits, the basic memory unit of quantum processors, and quantum operations (called gates) are the building blocks of quantum algorithms. In order to be run on quantum computing hardware, quantum algorithms must be compiled into a set of elementary machine instructions (i.e., quantum gates). Since currently available quantum circuits suffer a number of technological problems such as noise and decoherence, it is important that the process that carries out the quantum computation be somehow adapted to the physical limitations of the quantum hardware of interest, by means of a proper compilation.

For practical applications, it is important to make quantum computation able to tackle problem instances of more and more realistic size. To this aim, the ability to quickly compile quantum algorithms of good quality is paramount. Following our past research expertise on the subject [Oddi 2018, Rasconi 2019, Oddi 2020, Baioletti 2021], we aim at investigating the application of novel AI-based meta-heuristics to the problem of compiling quantum circuits to emerging quantum hardware, particularly focusing on Space-related applications.

We focus our initial modelling efforts by studying the so-called Quantum Alternate Operator Ansatz (QAOA) algorithms on the gate-model noisy intermediate-scale quantum (NISQ) processor units. Our approach intends to improve over the compilation algorithms applied to the current quantum computing software development kits (e.g., [Qiskit 2021]), devise solutions that are easily adaptable to different classes of problems.

1.1 The Noisy intermediate-scale quantum (NISQ) era

In the noisy intermediate-scale quantum (NISQ) era, the leading quantum processors contain about *50* to a few hundred qubits, but are not advanced enough to reach fault-tolerance nor large enough to profit sustainably from quantum supremacy. The term was coined by John Preskill in 2018, and it is used to describe the current state of the art in the fabrication of quantum processors.
The term 'noisy' refers to the fact that quantum processors are very sensitive to the environment and may lose their quantum state due to quantum decoherence. In the NISQ era, the quantum processors are not sophisticated enough to continuously implement quantum error correction.
The term 'intermediate-scale' refers to the quantum volume related to the not-so-large number of qubits and moderate gate fidelity.

The term NISQ algorithms refers to algorithms designed for quantum processors in the NISQ era. For example, the variational quantum eigensolver (VQE) or the quantum approximate optimization algorithm (QAOA), are hybrid algorithms that use NISQ devices but reduce the calculation load by implementing some parts of the algorithm in usual classical processors. These algorithms have been proven to recover known results in quantum chemistry and some applications have been suggested in physics, material science, data science, cryptography, biology and finance.

Usually, NISQ algorithms require error mitigation techniques to recover useful data, which however make use of precious qubits to be implemented. Thus, the creation of a computer with tens of thousands of qubits and enough error correction would eventually end the NISQ era. These “beyond-NISQ” devices would be able, for example, to implement Shor's

algorithm, for very large numbers and break RSA encryption. Until that point however, the need to produce circuits runnable in the current (or near-future) quantum architectures in a reasonably reliable manner (i.e., counting on noise minimization techniques rather than on error-correcting techniques) will stand. Hence, the need to provide quantum circuit compilation procedure that minimize the effects of decoherence by minimizing the circuit's depth (see Section 1.3).

1.2 Quantum Approximate Optimization Algorithm (QAOA)

QAOA (Quantum Approximate Optimization Algorithm) introduced in [Farhi 2014] is a quantum algorithm that attempts to solve combinatorial problems. Combinatorial optimization problems involve finding an optimal object out of a finite set of objects, often reduced to finding "optimal" bitstrings composed of 0's and 1's among a finite set of bitstrings. One such problem corresponding to a graph is for instance the Max-Cut problem.

More specifically, QAOA is a variational algorithm that uses a unitary $U(\beta, \gamma)$ characterized by the parameters (β,γ) to prepare a quantum state $|\psi(\beta, \gamma)\rangle$. The goal of the algorithm is to find optimal parameters $(\beta_{opt}, \gamma_{opt})$ such that the quantum state $|\psi(\beta_{opt}, \gamma_{opt})\rangle$ encodes the solution to the problem.

The unitary $U(\beta,\gamma)$ has a specific form and is composed of two unitaries:

1. $U(\beta) = e^{-i\beta H_B}$
2. $U(\gamma) = e^{-i\gamma H_P}$

where H_B is the mixing Hamiltonian and H_P is the *problem Hamiltonian*. Such a choice of unitary drives its inspiration from a related scheme called quantum annealing.

The state is prepared by applying these unitaries as alternating blocks of the two unitaries applied p times such that

$$|\psi(\beta, \gamma)\rangle = U_1(\beta)U_1(\gamma) \cdots U_p(\beta)U_p(\gamma) |\psi_0\rangle$$

where $|\psi_0\rangle$ is a suitable initial state.

A key issue in QAOA algorithms is to find the optimal parameters $(\beta_{opt},\gamma_{opt})$ such that the expectation value

$$\langle\psi(\beta_{opt}, \gamma_{opt})| H_P |\psi(\beta_{opt}, \gamma_{opt})\rangle$$

is minimized. Such an expectation can be obtained by doing measurement in the *Z-basis*. Generally, a classical optimization algorithm to find the optimal parameters is used. The involved steps are shown in Figure 1:

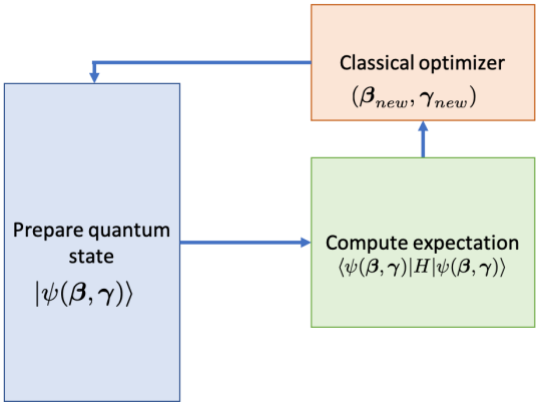

Figure 1 - QAOA optimization steps

And described in the following pseudocode:

1. Initialize *β* and *γ* to suitable real values.
2. Repeat until some suitable convergence criteria is met:
    a. Prepare the state $|\psi(\beta, \gamma)\rangle$ using QAOA circuit
    b. Measure the state in standard basis
    c. Compute $\langle\psi(\beta, \gamma)\,|H_P|\,\psi(\beta, \gamma)\rangle$
    d. Find new set of parameters $(\beta_{new}, \gamma_{new})$ using a classical optimization algorithm
    e. Set current parameters (β, γ) equal to the new parameters $(\beta_{new}, \gamma_{new})$

1.3 The QCC problem

Quantum Computing explores the implications of using quantum mechanics to model information and its processing. The impact of quantum computing technology on theoretical/applicative aspects of computation as well as on the society in the next decades is considered to be immensely beneficial [Nielsen 2011]. While classical computing revolves around the execution of logical gates based on two-valued bits, quantum computing uses quantum gates that manipulate multi-valued bits (*qubits*) that can represent as many logical states (*qstates*) as are the obtainable linear combinations of a set of basis states (state superpositions). A quantum circuit is composed of a number of qubits and by a series of quantum gates that operate on those qubits, and whose execution realizes a specific quantum algorithm.

Executing a quantum circuit entails the chronological evaluation of each gate and the modification of the involved qstates according to the gate logic. Current quantum computing technologies like ion-traps, quantum dots, super-conducting qubits, etc. limit the qubit interaction distance to the extent of allowing the execution of gates between adjacent (i.e., *nearest-neighbor*) qubits only. This has opened the way to the exploration of possible techniques and/or heuristics aimed at guaranteeing nearest-neighbor (NN) compliance in any quantum circuit through the addition of a number of so-called SWAP gates between adjacent qubits (Quantum Circuit Compilation Problem - QCCP), see ([Oddi 2018]).

The effect of a SWAP gate is to mutually exchange the qstates of the involved qubits, thus allowing the execution of the gates that require those qstates to rest on adjacent qubits. However, adding SWAP gates also introduces a time overhead in the circuit execution that generally depends on the quantum hardware's topology, as well as an increase of *noise*; on the other hand, it is highly desirable to minimize the circuit's execution time (i.e., *makespan*),

in order to mitigate the negative effects of *decoherence* and guarantee more stability to the quantum computation.

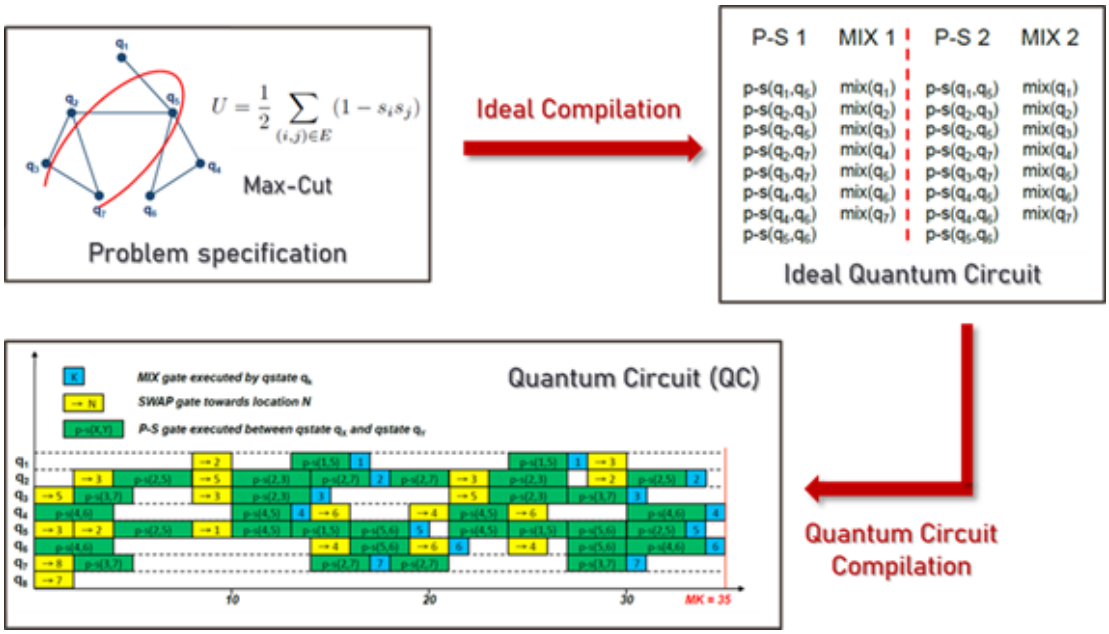

Figure 2 - The compilation process

The Quantum Circuit Compilation Problem (QCCP) can therefore be described as the problem of compiling (or "adapting") an *ideal* quantum circuit that realizes a specific function so that it can be run on a specific quantum hardware. Our approach leverages the fact that the QCCP can be reduced to a Planning & Scheduling (P&S) problem, and efficiently solved by exploiting the meta-heuristics generally used with P&S instances. To this aim, we based the development of the project on our recent research results (see [Oddi 2018, Rasconi 2019, Oddi 2020, Baioletti 2021]), in order to assess the effectiveness of our approach against a set of problems of particular interest, to be decided together with ESA's Advanced Concept Team (ACT).

As shown in **Error! Reference source not found.**, the quantum compilation process mainly deals with the careful synthesis (planning) and the temporal allocation (scheduling) of the necessary SWAPS that guarantee the satisfaction of the nearest-neighbourhood condition for all qstate pairs that are involved in the execution of every two-qubit gate, either PS (phase-shift or SWAP gates). In more details, the algorithm proceeds according to the following steps: (i) it starts from the specification of the problem of interest (e.g., Max-Cut), (ii) it synthesizes the quantum gates whose execution ideally implements the solution of the problem instance (Ideal Quantum Circuit), and finally (iii) it produces the compiled Quantum Circuit through the addition of properly planned SWAP gates.

1.4 Heuristic algorithms

The technology we produced is general, and the possibilities of its application potentially span over the whole spectrum of Quantum Alternating Operator Ansatz (QAOA) algorithms operated on NISQ technology ([Hadfield 2019]). In this respect, the applications that are interesting for the space community are several. Given our research background in the Planning & Scheduling (P&S) area (we have a record of applications of our technology to solve planning and scheduling problems targeted at realistic space applications), our

general strategy is to reduce the Quantum Circuit Compilation Problem to a P&S problem. The previous reduction comes very natural, as any quantum circuit can indeed be regarded as the temporal execution of a set of activities (i.e., the quantum gates) each requiring a number of resources (i.e., the qstates and/or the qubits on which the gates are executed), where each activity is characterized by a given duration, and where several temporal/causal links may exist among the activities for a variety of reasons, such as: (i) gates may not be commutative and hence they must be executed according to a certain order, or (ii) even in the commutative case, gates scheduling must be consistent with the ordering imposed by the execution pass the gates belong to. As a further observation, the QCCP is indeed a planning problem, as the compilation problem mainly entails the synthesis of a set of SWAP gates not originally present in the "ideal" circuit.

As previously stated, the problem of minimizing the makespan of quantum circuits on complex quantum hardware topologies is an NP-complete problem ([Botea 2018]), hence a heuristic approach is required, especially when large architectures are involved.
It can be easily demonstrated that the synthesis of a feasible quantum compilation circuit can be performed in polynomial time; intuitively, it is enough to begin from an empty circuit and iteratively adding all the gates belonging to the ideal quantum circuit when they are applicable (i.e., the NN condition is satisfied) or adding the SWAP gates that are necessary to their application. However, we have to face the problem that producing *any* quantum circuit is not enough: *we are interested in circuits whose makespan is as short as possible, because of the decoherence issue that still affects the quantum hardware currently available*. In other words, the problem we need to solve is no longer a feasibility problem but rather an optimization problem, which turns the problem's complexity from polynomial to NP-complete [Botea 2018]. This is the reason why solving techniques that rely on efficient meta-heuristics are necessary.

1.5 The Qiskit framework

Qiskit [Qiskit 2021] is an open-source software development kit (SDK) for working with quantum computers at the level of circuits, pulses, and algorithms. It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on IBM Quantum Experience or on simulators on a local computer. It follows the circuit model for universal quantum computation, and can be used for any quantum hardware (currently supports superconducting qubits and trapped ions) that follows this model.
Qiskit was founded by IBM Research to allow software development for their cloud quantum computing service, IBM Quantum Experience. Contributions are also made by external supporters, typically from academic institutions.
The primary version of Qiskit uses the Python programming language. Versions for Swift and JavaScript were initially explored, though the development for these versions have halted. Instead, a minimal re-implementation of basic features is available as MicroQiskit, which is made to be easy to port to alternative platforms.

As Qiskit is an open-source project committed to both: (i) bringing quantum computing to people of all backgrounds and to (ii) allowing for integrations on behalf of academic institutions, one of our objectives has been to implement our solutions in Python language to allow a seamless integration of our contribution as a possible extension to the Qiskit framework.

2 QUANTUM CIRCUITS AND THE COMPILATION PROBLEM

As it is evident from our works published so far, we focused our efforts on routing (i.e., compiling) circuits that implement the QAOA paradigm specifically applied to the Max-Cut problem. However, the QAOA approach lends itself to the representation of a set of other problems that may be of particular interest for the objectives of the present study.

More specifically, we are particularly interested in tackling problems whose structure is isomorphic to Planning & Scheduling (P&S) problems that may be particularly relevant in the space domain. To concretely identify the set of problems that could be realistically analysed within the limited timespan of this study, we focused our efforts in the *graph colouring problem* [Venturelli 2019], as it can be shown that a number of instances of P&S problems are reducible or strictly correlated to graph colouring. For example, we can consider instances of the satellite range scheduling problem [Zufferey 2008], where a set of communication jobs have to be assigned to a set of ground stations with the objective of minimizing the number of conflicting jobs.

Of course, a number of other problems may be considered in the future, for instance the real-world flight-gate assignment problem, i.e., the problem of assigning different gates to flights at an airport with the objective of minimizing the total transit times of all passengers [Stollenwerk 2020], or the resolution of the path planning problem for an exploratory rover that has to find the shortest way among a set of interesting locations for a long-range planetary exploration scenario, in the spirit of the Mars Sample Return (MSR) ESA mission concept. The previous problem may be of particular interest for Space: the problem can be modelled as a traveling salesman problem (see [Hadfield 2019]) and consists in planning a minimum-distance path among a set of scientifically interesting locations on the Martian surface to the aim of picking up the samples from past experiments (e.g., drills) and bring the samples to a space probe that will return them back to Earth.

2.1 Max-k-Cut Circuit

The combinatorial optimization problem initially tackled with QAOA [Fahri 2014] is the MAX-CUT (see Figure 3 (a)). Given an undirected graph *G = (V, E),* the objective of this problem is to find a subset *S* of the vertex set V such that the number of edges between S and the complementary subset *(V – S)* is maximized.

The circuit for solving the MAX-CUT with the QAOA approach is generally composed by *p phases (p >= 1*), and at least |*V*| qubits are necessary for its resolution. In each phase, the circuit comprises a level of PS gates, each of them working on the qubits associated to the end points of each edge, followed by a level of MIX gates, for each vertex (see Figure 4).

A generalization of the MAX-CUT is the GRAPH COLOURING problem (see Figure 3 (b)), where, given an undirected graph *G = (V, E)*, in which the set of vertices *V* can be coloured with one among *k* available colours, the objective is to maximize the number of edges in *E* that have end points with different colours. Clearly, the GRAPH COLOURING reduces to the MAX-CUT when k=2. A possible approach of creating a circuit for solving GRAPH COLOURING uses the *one-hot* encoding, by having *k* qubits for each vertex. The *i*-th qubit indicates whether the vertex is coloured with the colour *i (*see Figure 5*)*.

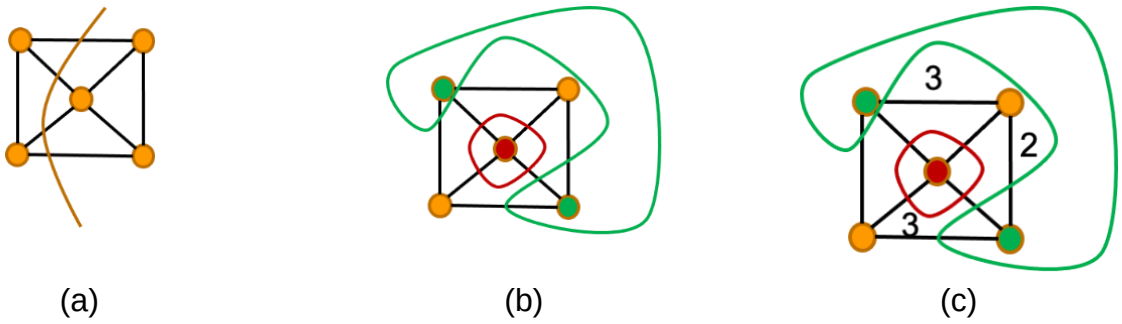

Figure 3 - (a) MAX-Cut, (b) Graph Colouring, (c) MAX-k-Cut

In its turn, the GRAPH COLOURING can be further generalized into the MAX-k-CUT problem (see Figure 3 (c)). In this case, given a weighted undirected graph $G = (V, E)$, the MAX-k-CUT problem consists of finding a *maximum-weight* k-cut, that is a partition of the vertices into k subsets, such that the sum of the weights w_{ij} of the edges (i,j) that have end points on different subsets is maximized.
Finally, LIST COLOURING generalizes MAX-k-CUT by restricting the set of allowed colours for each vertex.

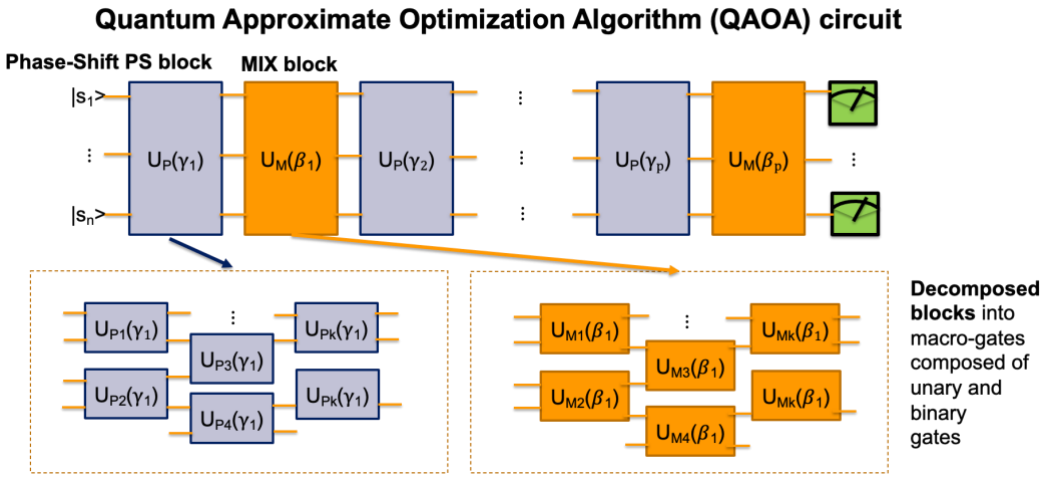

Figure 4 - Quantum Approximate Optimization Algorithm (QAOA) circuit

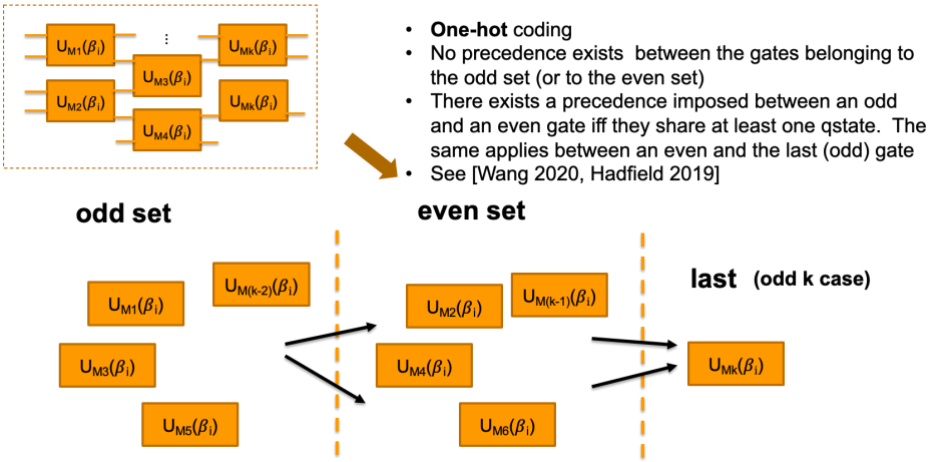

Figure 5 - Mixer blocks' implementation for Graph Colouring and Max-k-Cut

2.2 Satellite scheduling circuit

A combinatorial optimization problem based on LIST COLOURING and related to space applications is SATELLITE SCHEDULING. In this problem, a set of satellites are required to communicate with a set of *k ground stations* having unary capacity. Each satellite *i* can use only a subset S_i of the available ground stations. The aim of this problem is to maximize the number of non-conflicting communications. Note that this is generally an *oversubscribed* problem, as the satellite communication requests may exceed the available communication windows.

The SATELLITE SCHEDULING problem can be reduced to LIST COLOURING [Marx 2004] by creating a *conflict graph G(V, E)*, where *V* is the set of satellite communication windows, and there is edge *(i,j)* ∈ *E* if and only if the communication windows *i* and *j* in *V* have a non-empty intersection (see Figure 6). The weight w_{ij} on each edge *(i,j)* ∈ E represents the "importance of avoiding a conflict between the communication pair *(i,j)".* Finally, the number of colours *k* coincides with the number of ground stations.

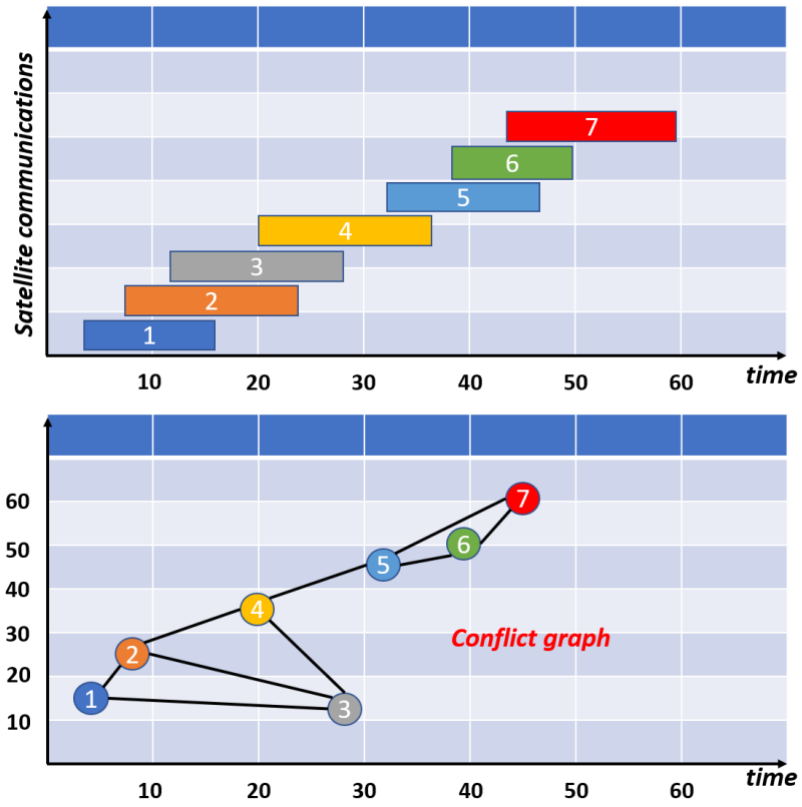

Figure 6 – An example of conflict graph: connected vertices correspond to overlapping communications

With respect to MAX-k-CUT, in order to correctly represent the SATELLITE SCHEDULING problem, the corresponding Quantum Approximate Optimization Algorithm (QAOA) circuit has to be modified in all its three components: 1) Initial state setting |*s*> [Cruz 2019]; 2) Phase-Shift PS blocks; 3) MIX blocks.

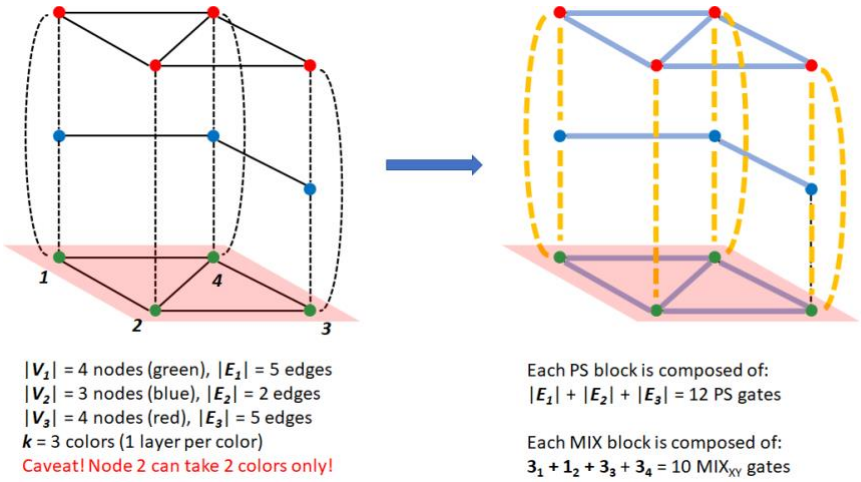

Figure 7 – One-hot coding for the List Colouring problem

An additional original contribution is the implementation one-hot coding schema for the Satellite Scheduling (reduced to List Colouring), such that the total number of used qubits is less than $k*|V|$ and coincides with the sum of the used colours calculated over the set of all nodes V (see Figure 7).

2.3 The Quantum Circuit Compilation Problem (QCCP)

The problems tackled in this report consists in compiling a given quantum circuit on a specific quantum hardware architecture. To this aim, we focus on the same framework used in [Do 2020], which is characterized by the following elements: (i) the class of Quantum Approximate Optimization Algorithm (QAOA) circuits [Farhi 2014; Guerreschi 2017] to represent an algorithm for solving the problems introduced in the previous Section 2.1 and Section 2.2 (ii) a specific hardware architecture, for example the one inspired by the architecture proposed by Rigetti Computing Inc. [Sete 2016]. The QAOA-based benchmark problems are characterized by a high number of commuting quantum gates (i.e., gates among which no particular order is superimposed) that allow for great flexibility and parallelism in the solution, which makes the corresponding optimization problem very interesting and guarantees greater circuit depth minimization potential for decoherence minimization [Venturelli 2017].

Formally, the Quantum Circuit Compilation Problem (QCCP) is a tuple $P = (C_0, L_0, QM)$, where C_0 is the input quantum circuit, representing the execution of the Max-k-Cut algorithm or the Satellite scheduling problem, L_0 is the initial assignment of the i-th qstate q_i to the i-th qubit n_i, and QM is a representation of the quantum backend (or quantum hardware).

- The input quantum circuit is a tuple $C_0 = (Q, VC_0, TC_0)$, where: (i) $Q = \{q_1, q_2, \ldots, q_N\}$ is the set of qstates; (ii) $VC_0 = P\text{-}S \cup MIX \cup \{g_{start}, g_{end}\}$ represents the set of *p-s* and *mix* gate operations (*P-S* and *MIX*) that have to be scheduled, such that: (i) every $p\text{-}s(q_i,q_j)$ and $mix(q_i, q_j)$ gate requires two qstates for execution; (ii) g_{start} and g_{end} are two fictitious reference gate operations requiring no qstates. Finally, TC_0 is a set of simple *precedence constraints* imposed on the *P-S*, *MIX* and $\{g_{start}, g_{end}\}$ sets.
- L_0 is the initial assignment at the time origin $t = 0$ of qstates q_i to qubits n_i.
- QM is a representation of the quantum backend as an undirected graph $QM = (V_N, E_{gate})$, where $V_N = \{n_1, n_2, \ldots, n_N\}$ is the set of qubits (nodes), E_{gate} is a set of undirected edges (n_i, n_j) representing the set of adjacent qubits.

A *feasible solution* is a tuple $S = (SWAP, TC)$, which extends the initial circuit C_0 to a compiled circuit $CC = (Q, V_{CC}, TC_C)$, such that $V_{CC} = SWAP \cup P\text{-}S \cup MIX \cup \{g_{start}, g_{end}\}$ and $T_{CC} = TC_0 \cup TC_C$ where: (i) *SWAP* is a set of additional $swap(q_i,q_j)$ gates added to guarantee the *adjacency constraints* for the set of *P-S* gates, and (ii) TC_C is a set of additional simple precedence constraints such that:

- For each qstate q_i, a *total order* ≤ is imposed among the set Q_i of operations requiring q_i, with $Q_i = \{op \in P\text{-}S \cup MIX \cup SWAP: op \text{ requires } q_i\}$;
- All the $p\text{-}s(q_i,q_j)$ and $swap(q_i,q_j)$ gate operations are allocated on adjacent qubits in *QB*;
- The graph (V_{CC}, TC_C) does not contain cycles.

Given a solution S, a path between the two fictitious gates g_{start} and g_{end} is a sequence of gates $g_{start}, op_1, op_2, \ldots, op_k, g_{end}$, with $op_j \in P\text{-}S \cup MIX \cup SWAP$, such that $g_{start} \leq op_1, op_1 \leq op_2, \ldots, op_k \leq g_{end} \in (TC_0 \cup TC_C)$. The length of the path is the number of all the path's gates and *depth(S)* is the length of the longest path from g_{start} to g_{end}. An *optimal solution S* is a feasible solution characterized by the *minimum depth*.

3 GRS ALGORITHM

In the following, we provide a detailed description of the Greedy Randomized Search (GRS) procedure used to compile the circuit introduced in previous Section 2. GRS has traditionally revealed a very effective method for the resolution of complex optimization problems (such as the QCCP), as it realizes a simple optimization process that quickly guides the search towards good solutions. The GRS is particularly useful in the cases where a high-quality solution is needed in a relatively short time. Among other applications, it is particularly suitable for constraint-based scheduling problems; since the QCCP can be reduced to a Planning and Scheduling (P&S) problem.

GRS Algorithm:

- **input**: Quantum backend *QM*, circuit *C,* cpu_time *T*
- **output**: best compiled circuit *CC**
    - *CC** ← *nil*
    - BestDepth ← *+inf*
    - **while not** (TimeLimitExceeded(T))
        - *CC* ← ***CompileCircuit***(*QM*, *C*)
        - **if** CC.depth < BestDepth **then**
            - CC* ← CC
            - BestDepth ← CC.depth
    - **return**(*CC**)

The above *GRS Algorithm* sketches the bulk of the implemented optimization process. It essentially realizes an optimization cycle in which a new solution *CC* is computed at each iteration through the *CompileCircuit*() algorithm, and its depth (*CC.depth*) is compared with the best depth found so far (*BestDepth*) in the iterative process. In case *CC.depth* is smaller than *BestDepth*, then the current solution *CC* becomes the new best solution *CC**. The optimization process continues until a stopping condition (generally a max time limit) is met, where the GRS procedure returns the best solution found.

As can be readily observed, the efficacy of the GRS mainly depends on the efficacy of the *CompileCircuit*() procedure (see below), which has the task of synthesizing increasingly better solutions. The *CompileCircuit*() is a random algorithm. It operates on **macro-gates**

containing primitive gates that use two qstates at most, and its pseudocode is shown in the following. Indeed, the *CompileCircuit*() procedure is in itself a heuristically-based iterative algorithm that implements a constructive methodology where a solution is built from scratch, and where the selection of which quantum gate to insert next in the solution is guided by a heuristic (the Quantum Gate Ranking Heuristic - QGRH) that returns a ranking that takes into account the "neighbouring cost" of all the gates that have yet to be inserted in the solution. At each iteration, the gate that guarantees the fastest realization of the neighbouring conditions of all the remaining gates is selected.

CompileCircuit Algorithm:

- **input**: Quantum backend *QM*, circuit *C*
- **output**: compiled circuit *CC*
    - *CC* ← InitSolution(*QM*, *C*)
    - t ← 0
    - **while** (not all the *INIT*, *PS* and *MIX* gates are inserted in *CC*)
        - *g* ← ***SelectExecutableGate***(*QM*, *CC, t*)
        - **if** g ≠ *null* **then**
            - CC ← ***InsertGate***(*g*, *CC*, *QM*)
        - **else**
            - *t* ← *t* + *1*
    - **return**(*CC*)

In the above *CompileCircuit()* algorithm the procedure *SelectExecutableGate()* selects a gate which can be a *PS*, a *MIX*, a ***SWAP*** gate. Indeed, it is a random algorithm targeted to minimize the solution depth, in particular its implementation is inspired to [Chand 2019], such that the selection of a gate *g* is based on two criteria: 1) the earliest start time gate selection (a value correlated to depth minimization); 2) a metric to minimize the number of swaps. At each iteration, the *SelectExecutableGate(QM, CC, t)* method selects the next gate to be inserted in the solution by means of the *InsertGate(g, CC, QM)* method. In all time instants *t* where no quantum gate can be selected for insertion, the current time t is increased (*t* ← *t* + *1*). The *CompileCircuit()* process continues until a complete solution is built.

4 EMPIRICAL EVALUATION

We have implemented and tested the proposed ideas leveraging the existing open-source quantum-related frameworks such as Qiskit [Qiskit 2021]. As described in Section 1.5, Qiskit is a known open-source Software Development Kit for working with quantum computers at the level of pulses, circuits and application modules. It allows for the creation, modification, simulation, and optimization of quantum circuits on a set both simulated and real quantum architectures, as well as allowing the possibility to test mapping algorithms on arbitrary quantum hardware topologies.

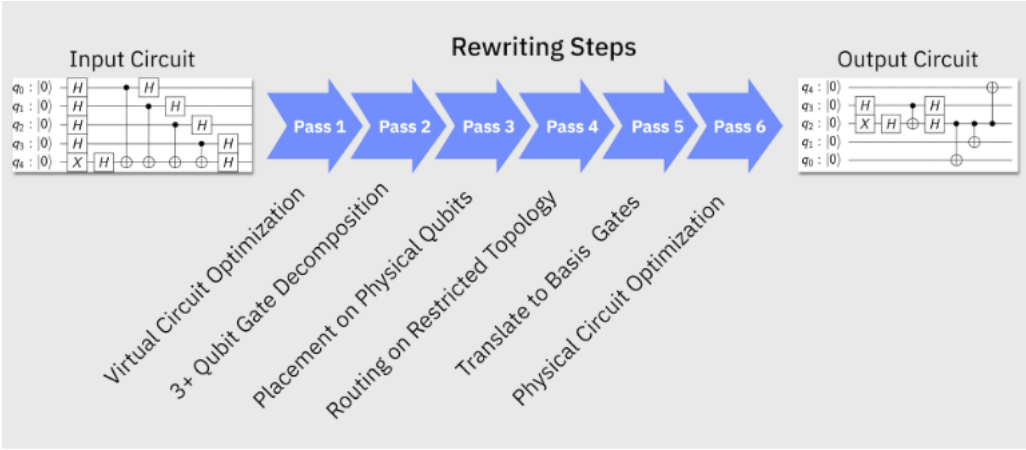

Figure 8 - Chain of possible modifications (rewritings) that can be applied to a quantum circuit, available through the Qiskit's Transpiler operator (image taken from: https://qiskit.org/documentation/apidoc/transpiler.html)

Figure 8 depicts the chain of possible modifications (rewritings) that can be applied to a quantum circuit, available through the Qiskit's *Transpiler* operator. The possible rewriting steps are the following: (i) the optimization transformations to the ideal (virtual) circuits that realize a specific function (Pass 1), (ii) the decomposition of 3+ quantum gates to 2-qubit gates in order to realistically allow their execution on the current quantum architectures (Pass 2), (iii) the mapping of the virtual qubits in a one-to-one manner to the "physical" qubits in the actual quantum device (Pass 3), (iv) the insertion of SWAP gates to the ideal circuit in order to overcome the qubit connectivity limitations of the current quantum hardware technology (Pass 4), (v) the translation of a quantum circuit in terms of the physical gates of the particular hardware of interest (Pass 5), and (vi) the optimization process applicable on the final physical circuit, generally performed in terms of gate simplifications.

Our contribution for this study has been focused on Pass 3 and Pass 4, through the implementation of efficient compilation procedures capable of tackling both the problem of quantum circuit compilation w.r.t. a given hardware topology (Pass 4) with the aim of minimizing the circuit's depth, and the problem of deciding the best initial virtual qubit -> real qubit mapping (Pass 3) with the aim of reducing the insertion of unnecessary SWAPS. The previous procedures were implemented in the Python language, in order to allow their integration within Qiskit. The performance of the proposed libraries was tested on a set of case study domains including the application of quantum computing to satellite scheduling problem.

We have defined a first benchmark set for the *graph colouring circuits*, obtained as an extension of the *N8* benchmark for the Max-Cut problem (e.g., see [Oddi 2018]), considering a number of colours $k = 3$. All the 22 instances have 7-nodes graphs, with $p=2$, hence quantum processors with at least 21 qubits (7 nodes x 3 colours = 21 qubits) are necessary; more specifically, we consider Rigetti-inspired 21-qubit processors. The Python version of the proposed greedy randomized search (GRS) algorithm compiles a QAOA circuit with the following choices: 1) *one-hot* coding for representing the MAX-k-Cut [Fuchs 2021]; 2) decomposition procedure for the QAOA blocks based on the identification of odd and even MIX_{XY} gates [Hadfield 2019, Wang 2020]. Figure 9 compares the proposed GRS algorithm with the following compiling algorithms available in Qiskit: *BasicSwap*, *StochasticSwap*, *SabreSwap*, *LookaheadSwap**. The algorithms are compared with respect to the depth of the compiled circuits (the circuit's depth represents the longest path in the compiled circuit graph) and on each algorithm a CPU time limit of 10 seconds is imposed for each run. Note that no solution is found by the *LookaheadSwap* algorithm within the previous CPU time limit.

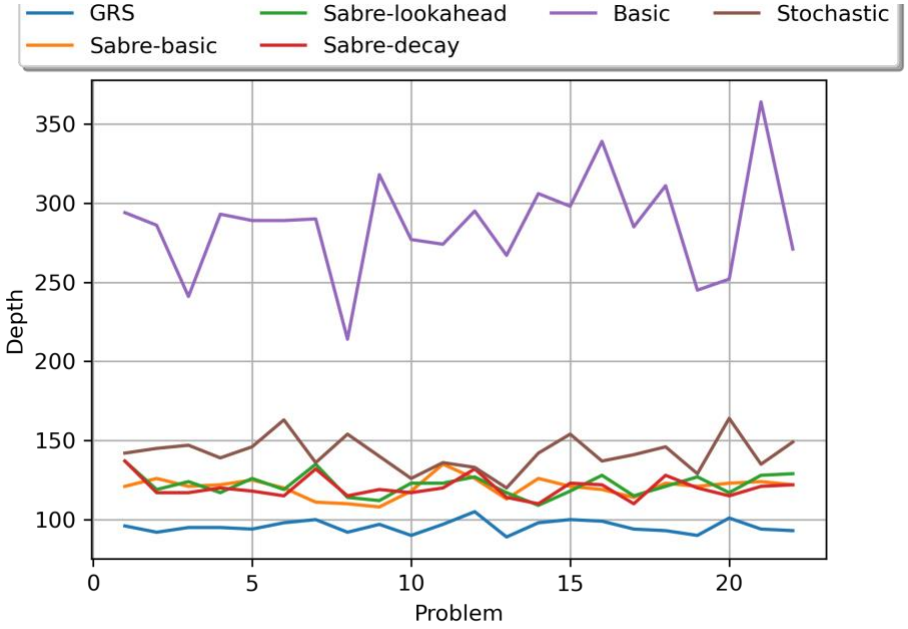

Figure 9 - GRS performance on graph colouring circuits

The following figure depicts the results obtained having a closer look to the comparison with the SabreSwap algorithm available in Qiskit, which can be run according to three different heuristics: basic, lookahead, decay

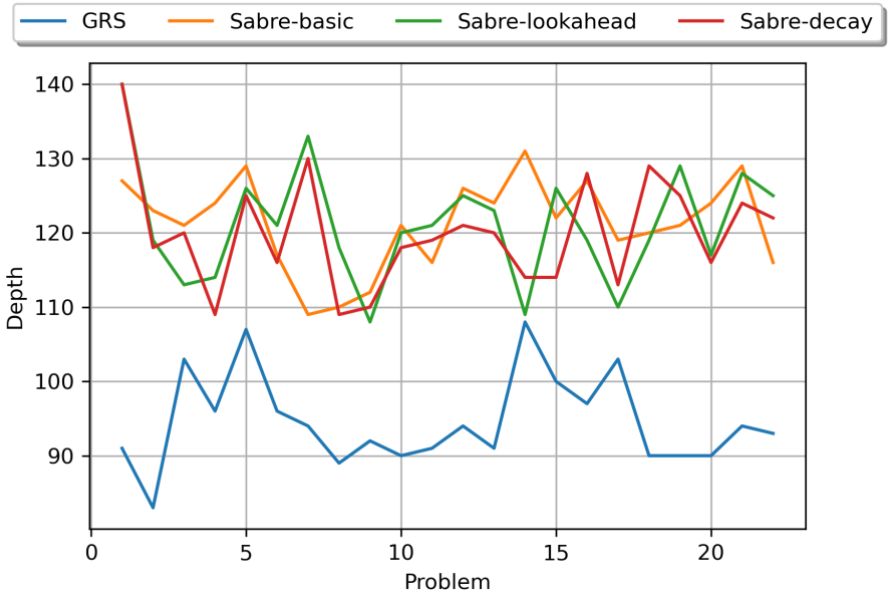

Figure 10 - GRS performance on Graph Colouring graphs

5 PYTHON MODULE

We have uploaded all the software proposed for the study *Meta-Heuristic Algorithms for The Quantum Circuit Compilation Problem* on a dedicated GitLab repository, which now contains the current version (not final) of the Python code. In order to install and test the current version of the software, we suggest to clone the full repository, and use the Jupyter notebook

Meta-HeuristicAlgorithmsForTheQuantumCircuitCompilationProblem.ipynb

contained in the folder *QuantumCompilers*.

Currently, the code works with the following main settings: Python 3.9.6, Qiskit 0.29.1. and requires part of the publicly available software downloadable from https://github.com/OpenQuantumComputing, and described in the paper [Fuchs 2021]. Note that the Python environment used to develop and run the software can be easily recreated by using the file *requirements.txt* contained in the folder *QuantumCompilers*.

The notebook mentioned above describes the generation, compilation and execution (on a qasm simulator) of a quantum circuit for solving the two problems proposed in the study: the *Max-k-Cut* and the *Space-Ground Communications* scheduling.

6 CONCLUSIONS

The aim of this study has been the investigation of the use of quantum computing as an *accelerator* for the resolution of optimization problems in the space domain. We have considered the compilation techniques for Noisy Intermediate-Scale Quantum (NISQ) devices. In particular, we have explored the QAOA (Quantum Approximate Optimization Algorithm) approach for solving optimization problems and studied the quantum circuits for two reference problems: the *Max-k-Cut* (an extension of the well-known *Graph Colouring* problem); the scheduling of *space-ground communications*. We have proposed a *greedy randomized search* (GRS) algorithm targeted at optimizing the compilation of quantum circuits and defined an original benchmark set for testing compilation algorithms. On the basis of our empirical validation the proposed GRS algorithm outperforms other compilation algorithms available in the Qiskit framework.

About the medium/long term goal of reaching the quantum *speed-up* over classical computers, we remark that the chosen QAOA approach exploits the intrinsic parallelism of quantum computing by concurrently evaluating all the possible assignments on problem variables and generating a probability distribution such that good assignments have a larger probability of being computed. According to [Fahri 2019] it is very unlikely that a conventional algorithm can create such a probability distribution efficiently. Moreover, as studied by [Guerreschi 2019], for QAOA applied to the solution of Max-Cut, *as the number of available qubits will increase to the thousands* (note that the proposed benchmark circuits used in this report have sizes of tens of qubits), QAOA will outperform the existing classical algorithms for combinatorial optimization.

7 REFERENCES

[Alam 2020] M. Alam, A. Ash-Saki and S. Ghosh. *Circuit Compilation Methodologies for Quantum Approximate Optimization Algorithm*. 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 215-228, doi: 10.1109/MICRO50266.2020.00029.

[Baioletti 2021] Marco Baioletti, Riccardo Rasconi, Angelo Oddi. *A Novel Ant Colony Optimization Strategy for the Quantum Circuit Compilation Problem*. EvoCOP 2021: 1-16.

[Botea 2018] A. Botea, A. Kishimoto, R. Marinescu (2018) “On the Complexity of Quantum Circuit Compilation”. In: Eleventh annual symposium on combinatorial search

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

[Clerc 2020] Maurice Clerc. *A general quantum method to solve the graph K-colouring problem*. 2020. ⟨hal-02891847v2⟩.

[Cruz 2019] Cruz D, Fournier R, Gremion F, Jeannerot A, Komagata K, Tosic T, Thiesbrummel J, Chan CL, Macris N, Dupertuis M-A, et al. *Efficient quantum algorithms for GHZ and W states, and implementation on the IBM quantum computer*. Adv Quantum Technol. 2019;2(5–6):1900015.

[Do 2020] Minh Do, Zhihui Wang, Bryan O'Gorman, Davide Venturelli, Eleanor Gilbert Rieffel, Jeremy Frank: *Planning for Compilation of a Quantum Algorithm for Graph Coloring*. CoRR abs/2002.10917 (2020)

[Farhi 2014] Farhi, E.; Goldstone, J.; and Gutmann, S. 2014. *A quantum approximate optimization algorithm*. arXiv preprint arXiv:1411.4028.

[Farhi 2019] Edward Farhi, Aram W. Harrowy. *Quantum Supremacy through the Quantum Approximate Optimization Algorithm*. arXiv:1602.07674v2

[Fuchs 2021] Fuchs, F.G., Kolden, H.Ø., Aase, N.H. et al. *Efficient Encoding of the Weighted MAX k-CUT on a Quantum Computer Using QAOA*. SN COMPUT. SCI. 2, 89 (2021). https://doi.org/10.1007/s42979-020-00437-z

[Guerreschi 2017] Guerreschi, G. G., and Park, J. 2017. *Gate scheduling for quantum algorithms*. arXiv preprint arXiv:1708.00023.

[Guerreschi 2019] Guerreschi, G.G., Matsuura, A.Y. *QAOA for Max-Cut requires hundreds of qubits for quantum speed-up*. Sci Rep 9, 6903 (2019). https://doi.org/10.1038/s41598-019-43176-9

[Hadfield 2019] Hadfield S, Wang Z, O'Gorman B, Rieffel EG, Venturelli D, Biswas R. *From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. Algorithms*. 2019;12(2):34.

[Li 2019] Gushu Li, Yufei Ding, and Yuan Xie. 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, 1001–1014. DOI:https://doi.org/10.1145/3297858.3304023

[Marx 2004] Marx, D. *Graph Colouring Problems and Their Applications in Scheduling*, Periodica Polytechnica Electrical Engineering, 48(1-2), pp. 11–16

[Murali 2019] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. *Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers*. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). Association for Computing Machinery, New York, NY, USA, 1015–1029. DOI:https://doi.org/10.1145/3297858.3304075

[Nielsen 2011] Nielsen, M.A., Chuang, I.L.: *Quantum Computation and Quantum Information*: 10th Anniversary Edition, 10th edn. Cambridge University Press, New York (2011)

[Oddi 2018] Angelo Oddi and Riccardo Rasconi. *Greedy Randomized Search for Scalable Compilation of Quantum Circuits*. 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2018), June 26-29, 2018 , Delft, The Netherlands.

[Qiskit 2021] Open-Source Quantum Development, https://qiskit.org/ (Accessed 21 December 2021)

[Rasconi 2019] Riccardo Rasconi, Angelo Oddi. *An Innovative Genetic Algorithm for the Quantum Circuit Compilation Problem*. AAAI 2019: 7707-7714.

[Sete 2016] Sete, E. A.; Zeng,W. J.; and Rigetti, C. T. 2016. *A functional architecture for scalable quantum computing.* In 2016 IEEE International Conference on Rebooting Computing (ICRC), 1–6.

[Stollenwerk 2020] T. Stollenwerk, S. Hadfield and Z. Wang. *Toward Quantum Gate-Model Heuristics for Real-World Planning Problems*. IEEE Transactions on Quantum Engineering, vol. 1, pp. 1-16, 2020, Art no. 3101816, doi: 10.1109/TQE.2020.3030609.

[Swamit 2019] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. *Not All Qubits Are Created Equal*: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). Association for Computing Machinery, New York, NY, USA, 987–999. DOI:https://doi.org/10.1145/3297858.3304007

[Venturelli 2019] 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

[Wang 2020] Zhihui Wang, Nicholas C. Rubin, Jason M. Dominy, and Eleanor G. Rieffel. 2020. *XY mixers: Analytical and numerical results for the quantum alternating operator ansatz*. Phys. Rev. A 101, 012320. DOI:https://doi.org/10.1103/PhysRevA.101.012320

[Zufferey. 2008] Zufferey, Nicolas, Amstutz, Patrick, Giaccari, Philippe. *Graph Colouring Approaches for a Satellite Range Scheduling Problem.* Journal of Scheduling, 2008, vol. 11, no. 4, p. 263-277.