ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200363

Planning for Compilation of a Quantum Algorithm for Graph Coloring

Minh Do[1] and **Zhihui Wang**[2] and **Bryan O'Gorman**[3]
and **Davide Venturelli**[4] and **Eleanor Rieffel**[5] and **Jeremy Frank**[6]

Abstract. The problem of compiling general quantum algorithms for implementation on near-term quantum processors has been introduced to the AI community. Previous work demonstrated that temporal planning is an attractive approach for part of this compilation task, specifically, the routing of circuits that implement the Quantum Alternating Operator Ansatz (QAOA) applied to the MaxCut problem on a quantum processor architecture. In this paper, we extend the earlier work to route circuits that implement QAOA for Graph Coloring problems. QAOA for coloring requires execution of more, and more complex, operations on the chip, which makes routing a more challenging problem. We evaluate the approach on state-of-the-art hardware architectures from leading quantum computing companies. Additionally, we apply a planning approach to qubit initialization. Our empirical evaluation shows that temporal planning compares well to reasonable analytic upper bounds, and that solving qubit initialization with a classical planner generally helps temporal planners in finding shorter-makespan compilations for QAOA for Graph Coloring. These advances suggest that temporal planning can be an effective approach for more complex quantum computing algorithms and architectures.

1 Introduction

Quantum computers apply quantum operations, called quantum gates, to qubits, the basic memory unit of quantum processors. Quantum algorithms are often specified as quantum circuits on idealized hardware, in which perfect gates can be applied to any set of qubits, whereas physical hardware has various constraints and imperfections. In practice, these idealized quantum circuits must be compiled to specific hardware. One common way of overcoming the restricted connectivity of such hardware is by adding additional gates that route qubit states to locations where the desired gate can act on them. Compilations that minimize the overall execution duration return results more quickly. More importantly, decoherence effects can destroy the computation in a short time and thus minimizing computation time is therefore vital to obtain results on near-term quantum hardware that does not support significant quantum error correction.

Recently, the use of temporal planners to compile quantum circuits was explored for QAOA applied to the MaxCut problem [25]; machine operations were modeled as PDDL2.1 durative actions, enabling domain-independent temporal planners to find a parallel sequence of conflict-free operations to implement the high-level quantum algorithm. Several state-of-the-art temporal planners were used to show empirically that temporal planning is a promising approach to compile circuits of various sizes to a model hardware chip featuring the essential characteristics of newly emerging quantum hardware. Building upon this work, several subsequent works have utilized different techniques to more effectively solve the same routing instance set. In [4], the authors extended the planning-based approach by integrating it with a constraint-programming solver to further improve the plan quality. Greedy randomized search and genetic algorithms are explored in [20, 22]; while those approaches provide improved results, they require building domain-dependent heuristics or encodings that may lack the flexibility of the model-based domain-independent temporal-planning approach introduced in [25], which can work on different classes of routing instances with a variety of hardware constraints and configurations.

In this paper, we expand the scope of the routing problem with a new target domain: QAOA for Graph Coloring [15, 27]; specifically, the optimization variant in which the number of properly colored edges is maximized. Compiling QAOA for Graph Coloring is different, harder, and more general than compilation of QAOA for MaxCut because of: (1) the existence of mix operations on two logical qubits (qstates); this leads to much more contention for resources on the gate-model hardware; and (2) the compilation task itself is more complex than it is for MaxCut. Thus, solving this problem class is key to future efforts to effectively utilize real-world gate-model quantum computers. Our main contributions over previous work are:

- *New instance class*: while our previous work concentrated on routing of QAOA for MaxCut, here we investigate routing of QAOA for Graph Coloring. Compiling Graph Coloring into a physical circuit requires a different set of gates, which include 'hybrid' gates, and more complex ordering between them, making the compilation task much more complicated. To our knowledge, we present a compilation study on the most complex Noisy-Intermediate Scale Quantum (NISQ) optimization algorithm application to date.
- *More diverse physical hardware architectures*: while previous work used an earlier hypothetical model from Rigetti, the computational results for this paper were conducted using hardware graphs and gate durations that are closer to the real hardware that Google, IBM, and Rigetti are building.
- *Qubit initialization*: we present initial research results singling out the problem of qubit initialization (QI), which is a sub-problem related to routing. Our approach of using classical planning to solve QI improves the performance of all tested temporal planners across a variety of problem setups.

[1] KBR & NASA ARC, USA, minh.do@nasa.gov
[2] USRA & NASA ARC, USA, zhihui.wang@nasa.gov
[3] UC Berkeley & NASA ARC, USA, bogorman@berkeley.edu
[4] USRA & NASA ARC, USA, davide.venturelli@nasa.gov
[5] NASA ARC, USA, eleanor.rieffel@nasa.gov
[6] NASA ARC, USA, Jeremy.D.Frank@nasa.gov

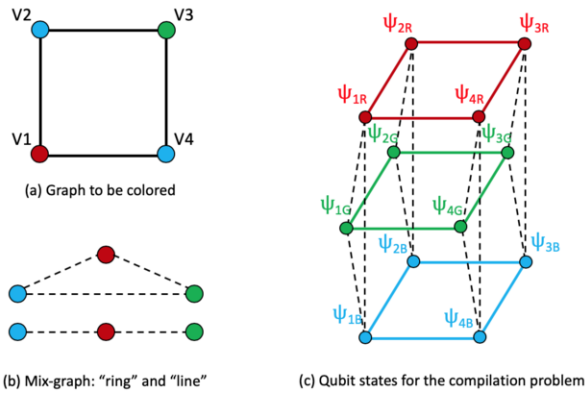

Figure 1: Routing for Graph Coloring: coloring a square with 3 colors.

The paper is structured as follows. The next section provides background on quantum circuit routing. Then, in Section 3, we outline the problem of circuit routing for QAOA on Graph Coloring. Section 4 describes how it can be modeled as a temporal planning problem using the PDDL2.1 standard modeling language. Section 5 describes our approach to using planning to initialize the qstates in order to minimize makespan. In Section 6, we describe different experiments and results showing the viability of our approach. Section 7 concludes the paper and outlines some of our future work directions.

2 Quantum Circuit Routing

General quantum algorithms historically have been described in an idealized architecture in which a gate can act on any subset of qubits. However, in an actual superconducting qubit device, such as the latest chips manufactured by IBM, Rigetti Computing, Google and Intel [7, 19, 2], physical constraints impose restrictions on the sets of qubits on which gates can be performed. Recently, a significant number of approaches have been explored for compiling idealized quantum circuits to realistic quantum hardware with a specific focus on "circuit routing"[7] (i.e., swap gate insertion strategies) in NISQ devices, targeting algorithms that could be run in the near-term [8, 28, 18, 22, 20, 14, 17].

For superconducting qubit architectures, qubits in these quantum processors can be thought of as nodes in a planar graph, with 2-qubit quantum gates associated with edges and 1-qubit quantum gates associated with nodes. Gates that operate on distinct sets of qubits may be able to operate concurrently, subject to additional restrictions, such as requiring the sets involved with concurrent gates to be non-adjacent. Furthermore, there are different types of quantum gate, each taking different duration that is dependent on the specific physical implementation. In order for the computation specified by the idealized circuit to be carried out, a particular type of 2-qubit gate, the *swap* gate, is often applied to exchange the state of two qubits. A sequence of swap gates moves the logical states of two distant qubits to a location where a desired gate can be applied. Swap gates may be available only on a subset of edges in the hardware graph, and swap duration may depend on where they are located. In this paper, we will consider the case in which swap gates are available between any two adjacent qubits on the chip, and all swap gates have the same duration; the more general cases are a straightforward generalization.

[7] Previous work referred to this as "quantum circuit compilation" (QCC).

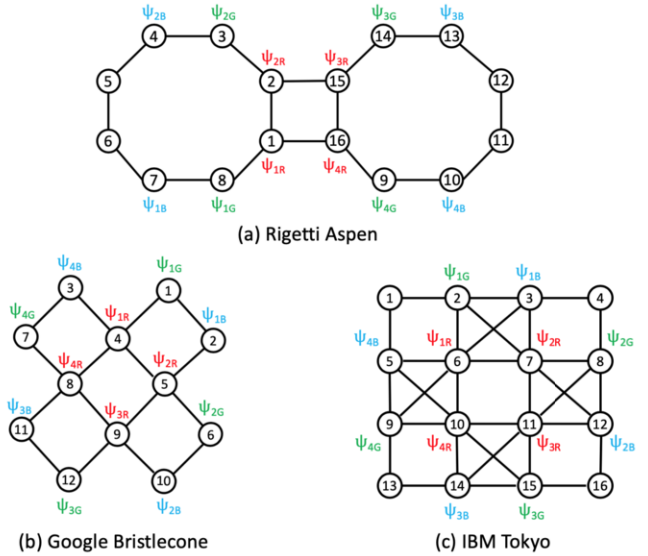

Figure 2: Example hardware chip layout from different companies.

3 Circuit Routing for Graph Coloring

In this paper, the Graph Coloring problem we will investigate is the *vertex coloring* problem in which all n vertices $v \in V$ of a given graph $\mathcal{G} = \{V, E\}$ are colored with k colors. The objective function is to maximize the number of edges $e = \langle v_1, v_2 \rangle \in E$ where vertices v_1 and v_2 are colored differently. This problem exemplifies the combinatorial structure of many scheduling and asset-allocation problems in industry and computer science research. Figure 1(a) shows a concrete example in which a square is properly colored with 3 colors.

The quantum algorithm that we follow is a variant of the "Quantum Alternating Operator Ansatz" [15], a generalization of the "Quantum Approximate Optimization Algorithm" (QAOA) [11], applied to the Graph Coloring problem.

The Ideal Circuit: the idealized QAOA circuit for Graph Coloring has been studied in [15] and [27]. We refer the quantum-computing skilled reader to those references for understanding the algorithm, and focus here only on its compiler-level implementation. It is specified by two types of 2-qubit gate, the *phase separation* (PS) gate and the MIX gate, which need to be applied to a set of instance-specific problem *goals*[8]. Each goal specifies a pair of *qubit states* (qstate), the information content of a qubit, that must have a PS or MIX "goal" gate applied to them. For the Graph Coloring problem, each qstate represents a *vertex-color* combination. Thus, for our leading example problem of coloring a graph of 4 vertices ($V = \{V_1, V_2, V_3, V_4\}$) with 3 colors (red [R], green [G], and blue [B]), there are a total of $4 \times 3 = 12$ qstates: $\Psi = \{\psi_{1R}, \psi_{2R}, \ldots, \psi_{4B}\}$, illustrated in Figure 1(c). The idealized Graph Coloring circuit is specified as follows:

- *PS gate requirements*: for QAOA for Graph Coloring, there should be one PS gate between any pair of qstates ψ_{iC} and ψ_{jC} that: (1) represent the same color C; (2) participate in an edge $e = \langle V_i, V_j \rangle \in E$. We denote the application of a PS gate as, e.g., PS(ψ_{1R}, ψ_{2R}), PS(ψ_{1R}, ψ_{4R}), PS(ψ_{3B}, ψ_{4B}).
- *MIX gate requirements*: one parameter of the QAOA for Graph Coloring is the "mix-graph" $\mathcal{G}_{mix}$ which consists of: (1) nodes

[8] Additional single qubit gates are also required in graph coloring but they are not relevant for routing since they can be executed at durations negligible compared to two-qubit gates, so we will disregard them in this paper.

representing different colors; and (2) edges representing which pairs of colors require MIX gates. In essence, mixing operators generate transitions between states representing different colors of the same vertex. Efficient mixing plays an important role in the quantum computation by achieving constructive interference that leads to a good solution. A complete mixing-graph would allow all colors to transit to each other faster than a minimally-connected mixing graph (a chain) where a color only transits to its neighboring colors in one step [27]. On the other hand, a denser mixing-graph will require more SWAP gates, leading to a longer circuit execution time. Figure 1(b) shows two examples of a mix-graph for the three colors used in our example: (1) "ring": each color is connected to the other two. If we remove one edge (e.g., $Green - Blue$), then we have a (2) "line" mix-graph. Given a mix-graph $\mathcal{G}_{mix}$, the requirement is to achieve one MIX gate for each pair of qstates ψ_{iC_j} and ψ_{iC_k} that: (1) represent the same vertex V_i; (2) are associated with two different colors C_j and C_k such that $\langle C_j, C_k \rangle$ is an edge in $\mathcal{G}_{mix}$. We denote the application of a MIX gate as, e.g, MIX(ψ_{1R}, ψ_{1G}), MIX(ψ_{1R}, ψ_{1B}), ..., MIX(ψ_{4G}, ψ_{4B}).

- *Goal orderings*: constraints on the circuit structure of QAOA enforce orderings between the PS and MIX gates: all PS gates involving a certain qstate ψ_{iC_j} should be completed before any MIX gate involving ψ_{iC_j} can be executed. Let's take the qstate ψ_{1R} as an example: the two PS gates PS(ψ_{1R}, ψ_{2R}) and PS(ψ_{1R}, ψ_{4R}) need to be completed before either of the two MIX gates: MIX(ψ_{1R}, ψ_{1G}) or MIX(ψ_{1R}, ψ_{1B}) can start. Note that goal ordering constraints do not enforce that *all* PS gates need to be completed before the MIX gates can start.

Architecture-specific operations: as described in Section 2, while the idealized quantum circuit for Graph Coloring contains the set of goal PS and MIX gates along with their orderings and the assumption that any goal gate can be applied anytime, they need to be "compiled" into architecture-specific operations that can actually be executed on a particular quantum chip that has numerous hardware constraints. Figure 2 shows several examples: (a) a 16-qubit Aspen chip layout by Rigetti Computing; (b) a 12-qubit section from a 72-qubit Bristlecone chip by Google; and (c) a 16-qubit section of IBM's 20-qubit Tokyo architecture.

The set of operations for Graph Coloring is significantly more complicated compared to the previous work on planning for routing of MaxCut [25, 4] where there are only three operations: 2-qubit PS operation, 1-qubit MIX operation, and 2-qubit SWAP operation. The increase in complexity arises also due to the availability of 'multi-purpose' or 'hybrid' operations that accomplish multiple objectives (e.g., SWAP + MIX). More specifically, the following types of operation need to be considered by the compiler to tackle QAOA for Graph Coloring on those architectures:

- *PS and MIX operations*: direct implementations of the ideal circuit PS and MIX gates that can be in principle applied to any pair of qstates residing on two qubits that are adjacent/interconnected on the hardware chip.
- *SWAP operation:* swaps the locations of two qstates residing on two interconnected qubits. For example, taking the layout of qstates on the Rigetti chip (Figure 2(a)): SWAP($\langle \psi_{2B}, q_4 \rangle, \langle \psi_{2G}, q_3 \rangle$) leads to: $\langle \psi_{2B}, q_3 \rangle$ and $\langle \psi_{2G}, q_4 \rangle$.
- *MOVE operation*: a variant of the SWAP operation that instead of swapping the locations of the two adjacent qstates, it moves a qstate to an adjacent empty qubit. For example (Figure 2(a)): MOVE($\langle \psi_{2B}, q_4 \rangle, q_5$) leads to $\langle \psi_{2B}, q_5 \rangle$, leaving q_4 empty[9].
- *SWAP-PS operation*: this operation combines the effects of the SWAP and PS operations. Thus, SWAP-PS($\langle \psi_{1R}, q_1 \rangle, \langle \psi_{2R}, q_2 \rangle$) in Figure 2(a) will switch the locations of ψ_{1R} and ψ_{2R} like the SWAP operation but also accomplish PS(ψ_{1R}, ψ_{2R}). For a pair of qstates that do not have a PS goal gate requirement, the SWAP-PS gate can be applied, but its effect is identical to using a SWAP gate. Thus: SWAP-PS($\langle \psi_{2B}, q_4 \rangle, \langle \psi_{2G}, q_3 \rangle$) has the same effect as SWAP($\langle \psi_{2B}, q_4 \rangle, \langle \psi_{2G}, q_3 \rangle$) since there is no goal gate requirement PS(ψ_{2B}, ψ_{2G}). Since the SWAP and SWAP-PS operations may have different durations depending on the particular physical connection, the planner needs to decide whether to use one over the other at a particular location on the chip.
- *SWAP-MIX operation*: similar to the SWAP-PS operation, this one combines the effects of the SWAP and the MIX operations. However, unlike SWAP-PS that can be applied to any pair of qstates on adjacent qubits, the SWAP-MIX operation can only be applied between qstates representing nodes in the same mix-graph (Figure 1(b)). Let's assume the "line" mix-graph in Figure 1(b), which has two connections $B \leftrightarrow R$ and $R \leftrightarrow G$, then: (1) SWAP-MIX($\langle \psi_{1R}, q_1 \rangle, \langle \psi_{1G}, q_8 \rangle$) has the combined effects of SWAP and MIX gates; (2) SWAP-MIX($\langle \psi_{1B}, q_7 \rangle, \langle \psi_{1G}, q_8 \rangle$) has the same effect as SWAP gate (but can have a shorter makespan to make it a better option than SWAP) since ψ_{1B} and ψ_{1G} are not connected on, but belong to the same, the mixgraph; (3) SWAP-MIX($\langle \psi_{1R}, q_1 \rangle, \langle \psi_{2R}, q_R \rangle$) are not allowed since they are not connected on a mix-graph.

Problem definition: Given an idealized circuit, comprised of PS and MIX gates, used to define a QAOA quantum algorithm for Graph Coloring, the circuit routing problem is to find a new architecture-specific circuit that implements the idealized quantum circuit, utilizing the architecture-specific PS, MIX, SWAP, MOVE, SWAP-PS, and SWAP-MIX operations as required. The objective is to minimize the overall duration to execute all operations in the new circuit.

4 Model Circuit Routing for Graph Coloring as a Temporal Planning Problem

Planning is the problem of finding a conflict-free set of actions and their respective execution times that connects the *initial-state* I and the desired *goal state* G. We now introduce some key background concepts for the routing-as-temporal planning problem.

Planner: a planner takes as input a specification of domain and problem instance, and returns a valid plan, if one exists. At the abstract level, the planner needs to solve the QAOA compilation problem described in the previous section: taking as input the required PS and MIX gates and utilizing the architecture and problem-specific operations to build a plan achieving all those gates.

Planning Domain Description Language (PDDL): The de-facto standard modeling languages used by many domain-independent planners. We use PDDL 2.1 [12], which allows the modeling of temporal planning formulations in which every action a has duration d_a, starting time s_a, and end time $e_a = s_a + d_a$. Action conditions are required to be satisfied either (1) instantaneously at s_a or e_a or (2) to be true starting at s_a and remain true until e_a. Action effects may instantaneously occur at either s_a or e_a. Actions can execute

[9] This operation does correspond to a SWAP operation in the real chip, where empty qubits don't exist. It is defined only for modeling convenience.

```
(:durative-action swap_mix_at_q1_q2
    :parameters (?s1 - qstate ?s2 - qstate)
    :duration (= ?duration 1.0)
    :condition (and (at start (located_at_q1 ?s1))
                    (at start (located_at_q2 ?s2))
                    (at start (ps_completed ?s1))
                    (at start (ps_completed ?s2))
                    (at start (mixgraph_edge ?s1 ?s2))
                    (at start (not (mixed ?s1 ?s2))))
    :effect (and (at start (not (located_at_q1 ?s1)))
                 (at start (not (located_at_q2 ?s2)))
                 (at end (located_at_q1 ?s2))
                 (at end (located_at_q2 ?s1))
                 (at end (mixed ?s1 ?s2))
                 (at end (mixed ?s2 ?s1))))
```

Figure 3: PDDL model of the SWAP-MIX operation.

when their temporally-constrained conditions are satisfied; and when executed will cause state-change effects. The most common objective function in temporal planning is to minimize the plan *makespan*, i.e., the shortest total plan execution time. This objective matches well with the objective of our targeted routing problem: minimizing the total circuit execution time (i.e., circuit depth).

Modeling Routing for Graph Coloring in PDDL 2.1: Following the software structure of the end-to-end compiler tool-chain presented in [24], at the highest level, we need to take as input:

- Qstates and their relations: this in turn encapsulates the graph to be colored $\mathcal{G}$ (Figure 1(a)) and the "mix-graph" $\mathcal{G}_{mix}$ (Figure 1(b)).
- The $\mathcal{G}_{machine}$ graph representing the hardware chip layout (Figure 2).

and turn them into *objects*, *predicates*, *actions*, *initial state*, and *goal state* of a temporal planning problem such that a valid plan represents a parallel sequence of architecture-specific operations (see Section 3) enabling all required PS and MIX gates.

Objects: as in [25], we model qstates as PDDL objects. Physical qubits and the connections in $\mathcal{G}_{machine}$, $\mathcal{G}_{mix}$, and $\mathcal{G}$ graphs are modeled implicitly with the list of predicates and action descriptions.

Predicates: the following facts are represented by PDDL predicates:

- located_at_$q(s)$: if a qstate s is located at a given qubit q.
- empty_q: if a given qubit q is empty. This predicate enables the MOVE action (see Section 3).
- psed(s_1, s_2) and mixed(s_1, s_2): if a PS or MIX gate has been accomplished between a pair of qstates s_1 and s_2.
- edge(s_1, s_2): if two qstates s_1 and s_2 are connected in the graph $\mathcal{G}$ to be colored. This predicate serves as a precondition of the PS action.
- mixgraph_edge(s_1, s_2): if qstates s_1 and s_2 are connected in the mix-graph $\mathcal{G}_{mix}$. This predicate serves as a precondition of the MIX and SWAP-MIX actions.
- same_mixgraph(s_1, s_2): if a given pair of qstates s_1 and s_2 belong to the same mix-graph, but are not connected by an edge. This enables the SWAP-MIX-AS-SWAP action between those qstates.
- ps_completed(s): if the PS phase for a given qstate s is finished.

Actions: there are 6 action templates representing the 6 architecture-specific operations described in Section 3: PS, MIX, SWAP, MOVE, SWAP-PS, and SWAP-MIX. These actions act on the edges of the graph $\mathcal{G}_{machine}$ with appropriate qstates as action parameters. As outlined in Section 3, the SWAP-MIX action's mixed(s_1, s_2) effect conditions on whether or not the two involved qstates s_1 and s_2 are connected in the $\mathcal{G}_{mix}$ graph. Given that many temporal planners

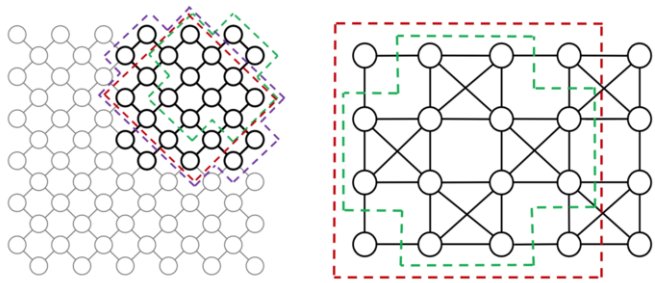

Figure 4: Google and IBM full chip configurations and the sections of 12 (green), 16 (red), 20 (purple), and 24 qubits used for our empirical evaluation.

can not handle conditional effects, to allow us to use a wider range of planners, we compile it into two deterministic actions: (1) SWAP-MIX: operates on two qstates s_1 and s_2 connected in $\mathcal{G}_{mix}$ (i.e., having mixgraph_edge(s_1, s_2) as a condition) and combines both MIX and SWAP action effects; and (2) SWAP-MIX-AS-SWAP: operates on two qstates s_1 and s_2 belonging to the same mix-graph but there is no edge connecting them (i.e., having same_mixgraph$(s_1, s_2) \wedge \neg$mixgraph_edge(s_1, s_2) as its conditions) and has the same action duration as SWAP-MIX while having the same effects as SWAP. We also introduce an auxiliary instantaneous action DonePS(s), which has a single effect ps_completed(s), to specify that a qstate s is done with the PS phase and is ready to move on to the MIX phase. In our example shown in Figure 1, DonePS(ψ_{1R}) can be executed when psed(ψ_{1R}, ψ_{2R}) and psed(ψ_{1R}, ψ_{4R}) are achieved and its ps_completed(ψ_{1R}) effect in turn enables any MIX or SWAP-MIX action involving ψ_{1R}, e.g. MIX(ψ_{1R}, ψ_{1B}) (see Figure 3).

Initial state: the initial state declares the initial values of all predicates: (1) located_at_$q(s)$ and empty_q specify the initial locations of all qstates s (and if some physical qubits q are empty); (2) mixgraph_edge(s_1, s_2) and same_mixgraph(s_1, s_2) values capture the $\mathcal{G}_{mix}$ graph; and (3) edge(s_1, s_2) values represent the connections in the input graph $\mathcal{G}$ to be colored.

Goal state: the goal state specifies the list of MIX gates, represented by the mixed(s_1, s_2) predicates, that need to be achieved. The ps_completed(s) conditions of the MIX and SWAP-MIX actions (see Figure 3) capture the ordering constraints between PS and MIX gates and ensure that all requisite PS gates will also be achieved.

5 Qubit Initialization

Qubit Initialization (QI) is the problem of assigning the initial locations of all qstates on the chip. Figure 2 shows examples of QI on the three machine configurations. Two approaches for QI were explored in the previous routing as Temporal Planning for MaxCut work [4]: (1) random initialization of qubits; and (2) for each qstate s, add an action INIT(s, q) to locate s on the physical qubit q; all INIT actions need to be carried out before any other action can start.

Basically, the second approach combines QI with routing, resulting in the combined Routing-I problem [4], with the hope that current state-of-the-art temporal planners would be able to find good initial locations for all qstates that support finding lower makespan plan. While [4] compared different temporal planners on solving Routing-I for MaxCut, this work didn't report the difference in makespan between random initialization and Routing-I. Our current investigation for Routing as Temporal Planning for Graph Coloring

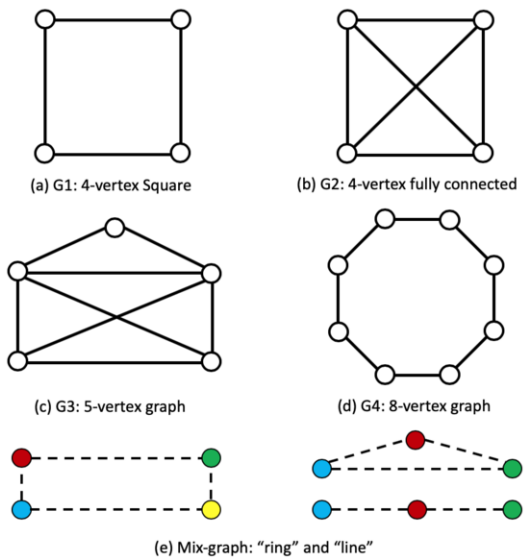

Figure 5: The benchmark instance set: Graphs to be colored (a-d) and mix-graph configurations (e).

shows that using a temporal planner to solve the Routing-I problems is not a clear-cut better option than random initialization (see results in Section 6.3), and it doesn't perform well compared to careful manual initialization utilizing domain knowledge about the problem structure and the hardware chip layout. We believe this is due to: (1) the large number of possible initial configurations (for a n-qubit chip, the number of possible initial qstate allocations is $n!$, modulo potential symmetries); and (2) since initialization needs to be fully done before gate composition begins, it is difficult for the existing planners to compute a good heuristic estimate on which initial state is a good one to start from since they are the furthest states from the goals. The farther a given state from the goals, the harder to get a good heuristic estimate for that state.

Qubit initialization as classical planning: instead of solving the combined Routing-I problem, we can heuristically solve the QI problem separately. Thus, we can try to find the initial locations I of qstates so that the subsequent routing problem yields shorter makespan plans. As described in Section 3, the routing for Graph Coloring requires each involved qstate to conduct all related PS gates, and then all related MIX gates. Since the PS gates will be done prior to the MIX gates, one heuristic is to initialize qstates so that if a pair of qstates s_1 and s_2 requires PS(s_1, s_2) as a goal, then they would be initially allocated on two qubits that are close to each other on the hardware chip. In our leading example, shown in Figure 1 and 2, we would like to put the two qstates ψ_{1R} and ψ_{2R} close initially to minimize the total amount of time to execute the needed SWAP gates to bring ψ_{1R} and ψ_{2R} to two connected qubits to enable the required PS(ψ_{1R}, ψ_{2R}) gate. To heuristically accomplish this objective, we setup and solve a cost-optimization classical planning problem P', which is significantly simpler than the temporal planning for circuit routing problem P described in Section 4, as follows:

Predicates: the following facts are represented by predicates for P': located_at_$q(s)$, empty_q, edge(s_1, s_2), and psed(s_1, s_2) (see Section 4 for their meaning). Predicates that are needed in P but are not used in P' are: mixed(s_1, s_2), mixgraph_edge(s_1, s_2), same_mixgraph(s_1, s_2), and ps_completed(s).

Actions: the *non-temporal* version of three actions SWAP, SWAP-PS, and PS in the PDDL2.1 model for P make up the action set for P'. The actions cost for SWAP and SWAP-PS in P' are set to be equal to the duration of the temporal SWAP and SWAP-PS in P while the action cost of PS is set to a constant value 1.

Initial State: declares the initial locations of all qstates s by setting up the values of located_at_$q(s)$ and empty_q; values of edge(s_1, s_2) represent the connections in the input graph $\mathcal{G}$ to be colored.

Goal State: specifies the list of PS gates that need to be achieved.

We then use a classical planner to find a solution π for P' with the objective function of minimizing the total plan cost. The final locations of all qstates *after* executing π are then used as the initial qstate allocation for the original temporal planning problem P. Since the goal of P' is to achieve all PS gates in P, we conjecture that qstate pairs (s_1, s_2) that are engaged in the PS(s_1, s_2) gates in P would likely be close to each other when π is done executing. If either s_1 or s_2 needs to be moved after PS(s_1, s_2) is accomplished to accommodate another PS gate (e.g., PS(s_1, s_3)), then the objective function of mimimizing the total action cost will enforce a sequence of SWAP and SWAP-PS gates that take the minimum total time to accomplish it. Thus, the final qstate locations after π is done executing setup an attractive initial locations for the original planning problem P where the same set of PS gates need to be accomplished before the subsequent MIX gates.

Given the simplified problem P' and the larger number of classical planners available, compared to temporal planners, the QI-as-classical-planning problem should be significantly easier to solve compared to the original Routing as Temporal Planning problem.

6 Empirical Evaluation

In this section, we will present preliminary empirical evaluation of different temporal planners applied to routing for Graph Coloring.

Hardware Architecture: We use the recently published hardware chip architecture layouts from Rigetti (Aspen), Google (Bristlecone), and IBM (Tokyo). The full-size chip from Rigetti is shown in Figure 2(a), and sections of the full chip from Google and IBM are shown in Figure 2(b) and (c) respectively. Figure 4 shows the full 72-qubit Bristlecone and the 20-qubit Tokyo architectures. Overall, the hardware architectures we use have 12–24 qubits: (1) Rigetti: 12-qubit section and the full 16-qubit chip (Figure 2(a)); (2) Google: 12-, 16-, 20- and 24-qubit sections of the Bristlecone 72-qubit architecture (Figure 4(a)); (3) IBM: 12- and 16-qubit sections and the full 20-qubit Tokyo chip (Figure 4(b)).

Software: Previous work on routing for MaxCut used TFD [10], LPG [13][10], CPT [26], POPF [6], and SGPlan [5][11]. We exclude CPT due to poor scalability (for both Graph Coloring and Max Cut) and replace POPF with the more modern code of another planner, OPTIC [3], in the same family. All planners were run in the anytime setting (except SGPlan which does not have this mode). For solving the QI-as-planning problem, we use the classical planner Fast Downward [16] with the LAMA 2011 [23] configuration.

Problem specifications: We test our approach on a range of randomly selected Graph Coloring instances with 4, 5, and 8 vertices and either the "line" or "ring" mix-graph. They are shown in Figure 5(a–d) as graphs G1–4. For example, graph G2R4 (Table 1, 2, and 4) means solving graph G2 in Figure 5 with the "Ring" mix-

[10] Since LPG is a stochastic planner, for each problem, we ran LPG 10 times and report its best result.

[11] We are considering TSGP as an alternative to SGPlan.

graph with 4 colors. The number of vertices n in the graph and the number of colors k in the mix-graph will require the number of physical qubits $n \cdot k$ within the range of 12-24 qubits, as described in the previous paragraph.

Gate durations: For all hardware architectures and for all edges in the hardware chips, the following gate durations are used: dur(SWAP) = 4, dur(MOVE) = 4, dur(PS) = 3, dur(SWAP-PS) = 4, dur(MIX) = 1, and dur(SWAP-MIX) = 1. These are effective durations based on logical gate synthesis using a common native gate sets, e.g., those available on Rigetti's chips [1]. Results were collected on a RedHat Linux VM running on a Macbook Pro with 4GB of RAM. The runtime limit was set to 600 seconds for problems involving 12–16 qubits and 1200 seconds for problems involving 20–24 qubits.

Table 1: Plan quality comparison in solving problems shown in Figure 5 with different hardware architectures shown in Figure 2 and 4. Qubit initialization is done manually using expert knowledge, similar to those in Figure 2. **bold** values indicate the best overall makespan. "-" indicates either the planner ran out of time before finding a solution (OPTIC) or crashed (SGPlan).

	Graph	TFD	OPTIC	LPG	SGPLAN
Rigetti-12	G1R3	**28**	**28**	31	61
	G1L3	**27**	42	33	44
Google-12	G1R3	**22**	45	46	83
	G1L3	**21**	38	41	68
IBM-12	G1R3	**23**	37	31	51
	G1L3	**17**	30	36	39
Rigetti-16	G1R4	**31**	-	80	94
	G2R4	**74**	-	119	156
Google-16	G1R4	**19**	46	20	34
	G2R4	76	49	**37**	48
IBM-16	G1R4	43	-	26	**20**
	G2R4	79	58	49	**29**
Google-20	G3R4	**64**	-	86	106
IBM-20	G3R4	113	-	**94**	-
Google-24	G4R3	125	**64**	83	-

Table 2: Comparing against analytical bounds for special hardware architectures: grid and line. Highlight in **bold** are makespan values that are better than the analytical bound. Underlined values are the best makespan produced by any planner when this best makespan is worse than the analytical bound.

	Analyt. Bound	TFD	OPTIC	LPG	SGPLAN
4 × 3 Grid	19	20	35	**16**	**13**
4 × 4 Grid	20	30	56	**20**	38
5 × 4 Grid	24	54	116	79	46
8 × 3 Grid	35	**25**	53	36	-
4 × 3 Line	64	**51**	77	**48**	90
4 × 4 Line	80	**71**	116	107	177
5 × 4 Line	100	118	-	140	194
8 × 3 Line	128	**81**	-	157	267

6.1 Solving Circuit Routing with Temporal Planner

Table 1 compares results between 4 temporal planners. Overall, TFD and LPG are the only two planners that can solve all problems within the time limit. In terms of solution quality, TFD is most often the best planner, especially for smaller problems. However, each planner has at least one case where it performs best. SGPlan generally returns the worst quality plan but it also performs the best in the two problems on IBM 16-qubit architecture. With regard to *ring* vs *line* mix-graph, as expected, the ring version of the 12-qubit problems normally has slightly longer makespan value than the line counterpart, given that it requires additional MIX goal gates. Between different hardware architectures, the Rigetti machine has many fewer connections for the same chip size (12 or 16-qubit), which leads to fewer parallel routes to move qstates. This led to longer makespan plans, in general, compared to solving the same problems on the Google and IBM architectures. Comparing the Google and IBM architectures of the same size, we expect planners should be able to find equal or shorter makespan plans given that they have the same overall shape except that the IBM ones have some additional connections. While that generally holds true for the 12-qubit version, looking at the results for the 16-qubit version we see TFD, OPTIC, and LPG perform worse on the IBM architecture than the Google architecture. It seems that the additional connections enlarge the planning search space and confuse the planners: they are not able to exploit the additional connectivity to find better quality plans. SGPlan is the only planner that show marked better performance on the IBM 16-qubit architecture.

Table 3: Improvement in makespan when solving a problem using qubit initialization provided by a classical planner (Section 5). Example: entry 8.1%(8/10) for cell IBM-16, G2R4, OPTIC means: among 10 randomly generated QIs for solving G2R4 on IBM-16, OPTIC can solve 8 with random QI while it can solve all 10 with QIs produced by the Fast Downward classical planner solving the same random problems. Accross the 8 problems that are solved with both QI setups, the average makespan improvement is 8.1%. When the number of solved problem is not listed, it means all 10 are solved with both QI options. RED indicates entries where the Fast Downward's QI performs worse than random QI.

	Graph	TFD	OPTIC	SGPlan
Rigetti-12	G1R3	17.9%	14.4%	19.1%
	G1L3	10.5%	2.7%	3.52%
Google-12	G1R3	9.7%	7.9%	-5.7%
	G1L3	16.3%	12.3%	13.1%
IBM-12	G1R3	9.2%	2.1%	10.9%
	G1L3	10.2%	18.4%	17.4%
Rigetti-16	G1R4	13.6%	23.8%(7/10)	15.7%
	G2R4	20.4%	24.6%(8/10)	18.1%
Google-16	G1R4	7.4%	5.3%	12.3%
	G2R4	17.5% (10/9)	7.9%(9/10)	11.3%
IBM-16	G1R4	11.2%	-3.9%	14.0%
	G2R4	6.4% (7/6)	8.1%(8/10)	7.2%
Google-20	G3R4	23.0%	2.6%(4/9)	23.7%
IBM-20	G3R4	12.6%	(2/3)	-
Google-24	G4R3	6.8% (9/9)	(0/2)	-

6.2 Planner vs Analytical Bounds

Without another systematic approach to solve Routing for Graph Coloring to compare with, one question remains: *how good is the quality of solutions returned by temporal planners in this domain?* For comparison, we can use simple manually constructed solutions, which are currently available for certain problem setups, as an upper bound [21]. Recall that for coloring a graph of n vertices with k colors on a hardware chip, we use $n \cdot k$ qubits. For hardware that supports the gates described in Section 3, we can get valid plans for the two hardware architectures: an $n \cdot k$ "grid" layout and a $n \cdot k$-length "line" layout with the following makespans:

$$\text{makespan}_{LINE} \leq n \cdot \tau_{\text{SWAP\_PS}} + nk \cdot \tau_{\text{SWAP}} + k \cdot \tau_{\text{SWAP\_MIX}}$$
$$\text{makespan}_{GRID} \leq n \cdot \tau_{\text{SWAP\_PS}} + k \cdot \tau_{\text{SWAP\_MIX}}$$

Table 4: Comparing different initialization strategies. **M**: **M**anual initialization (results from Table 1); **I**: Solving the combined QI + Routing (Routing-I) problem in one temporal-planning run (see Section 5); **Random** and Fast Downward(**FD**) are initialization setups explained in Table 3 with "AVG" and "Best" representing the average and best values across 10 random QIs. Values in RED show the best value across all setups; values in **bold** are best for a given planner. Random vs FD qubit initialization: Light Yellow background indicates better makespan by FD while Light Cyan indicates better makespan for random QI.

		TFD						OPTIC						SGPlan						LPG
Problem		M	I	Rand. (AVG)	Rand. (Best)	FD (AVG)	FD (Best)	M	I	Rand. (AVG)	Rand. (Best)	FD (AVG)	FD (Best)	M	I	Rand. (AVG)	Rand. (Best)	FD (AVG)	FD (Best)	M
Rigetti-12	G1R3	28	-	61.2	53	50.1	42	28	76	73.7	63	62.7	44	61	89	94	69	75.9	**53**	31
	G1L3	27	-	52.3	44	46.3	39	**42**	62	65	49	62.9	48	**44**	84	73.4	62	70.5	59	33
Google-12	G1R3	22	47	39.8	33	35.8	27	45	40	48.7	**31**	43.7	34	83	77	60.9	**43**	65.4	52	46
	G1L3	21	50	38.4	34	31.9	25	38	28	43	34	37.7	**27**	68	69	62.1	46	52	**38**	41
IBM-12	G1R3	23	35	33.8	27	30.5	26	**37**	61	47.5	40	45.5	**37**	51	63	56.5	47	49.8	**43**	31
	G1L3	17	25	29.9	19	26.4	21	30	39	44.2	31	36.1	**25**	39	50	52.9	42	43.2	**38**	36
Rigetti-16	G1R4	31	77	62.4	49	51.7	44	-	66	78.4	52	58.3	**43**	94	118	119.3	92	97.4	**68**	80
	G2R4	74	-	83.2	65	64.5	56	-	85	94.5	76	72.1	**63**	156	140	139	105	111	**89**	119
Google-16	G1R4	19	-	42.4	36	39.2	31	46	51	58.9	47	55.8	**39**	**34**	94	86.3	57	73.5	55	20
	G2R4	76	-	98.3	55	79.8	**43**	49	68	71.8	61	65.7	**48**	**48**	103	112.3	83	98.1	75	37
IBM-16	G1R4	43	-	60.9	47	53.7	**31**	-	57	50.8	34	54.1	**32**	20	-	77.3	63	65.9	53	26
	G2R4	79	-	85	74	79.5	**70**	58	56	73.5	57	64.6	**51**	29	-	89.2	67	82.7	52	49
Google-20	G3R4	64	-	128.9	64	90.7	58	-	-	86	76	82.9	**62**	106	-	152.3	115	115.1	**88**	86
IBM-20	G4R4	113	-	111.7	82	96.3	48	-	-	77.5	**70**	81	76	-	-	-	-	-	-	94
Google-24	G4R3	125	-	155.7	134	143.6	**113**	64	37	-	-	72.5	61	-	-	-	-	-	-	83

This can be accomplished through a predetermined qubit initialization and operation sequence such that while there will be SWAP gates required for the "line" layout, the "grid" layout only needs combined gates (i.e., SWAP-MIX and SWAP-PS) to accomplish all goal gates. Table 2 shows that while each planner's performance is quite inconsistent, the best plan returned by the planning approach compare well with reasonable upper bounds for those "grid" and "line" chip layout.

6.3 Qubit Initialization as Classical Planning

To measure the effect of qubit initialization for Graph Coloring by solving a classical planning problem, for each of the 15 problem configurations described in the previous Section 6.1 and shown in Table 1, we: (1) generate 10 random qubit initializations; (2) use the classical planner Fast Downward to solve the qubit initialization problem as described in Section 5 (time limit: 200 seconds); (3) solve two versions of the problem with: (i) random initialization and (ii) with initialization given by the solution returned by Fast Downward. Table 3 shows the makespan improvement of using the same planner[12], solving (ii) over solving (i); averaged over 10 random problem for each configuration. We exclude LPG from this experiment due to its randomness nature. LPG would employ different random seeds when solving (i) and (ii), making the comparison not justifiable.

In essence, Table 3 shows that given a (random) qubit initialization, different planners benefit from running a classical planner such as Fast Downward as a pre-processing phase to potentially find better initial locations for all qstates. The results show that all three planners benefit from this step for a majority of testing scenarios: solving higher overall number of instances while experiencing makespan improvement with the qubit initialization calculated by Fast Downward. There are only two cases in which average makespan increases: OPTIC/ IBM-16/G1R4 and SGPlan/Google-12/G1R3. The reason is likely that the majority of the random initializations happen to be very good starting points and Fast Downward's plans move qstates to overall worse initializations. Note that the classical planning setup for QI approximates and relaxes the problem by: (1) ignoring the MIX goals; (2) removing the temporal aspects (and thus finding a sequential non-parallel plan). Therefore, it's possible that the final qstate locations provided by the Fast Downward solution is worse than the starting random initialization.

Overall Comparison: Table 4 shows the makespan comparison between different settings for all planners. Overall, TFD consistently provides the best plan quality with manual initialization for smaller problems. For larger problems, where finding a good manual initialization is likely harder, different planners return the best quality solutions for different problem settings. Taking each individual planner, the data show, as expected, the best results are from either manual initialization or provided by using the Fast Downward (FD) planner. Between those two settings, FD provides better results for TFD and OPTIC on larger problems; for SGPlan, manual initialization provides better results for larger problems. While Routing-I, where a temporal planner solves both the QI and routing problem in one planning model, would be the most convenient setup, the results show that existing temporal planners are still not able to solve that problem effectively. Only one best makespan is found using this problem setup, provided by the OPTIC planner. It's also worth pointing out that while TFD is the most promising planner for routing, it performs worst on Routing-I with only 5/15 problems solved. Consistent with the results shown in Table 3, using the QI result provided by the Fast Downward planner, all three planners produce smaller makespan values (both average and best) compared to the random QI counterpart.

In summary, at large scale and for general instances, it is infeasible to manually solve the QI problem; for those cases, we show that the initialization process can be automated using a classical planner. The most promising planner is TFD and it should be the first one to try out. However, given that SGPlan is by far the fastest among all 4 tested planners, solving all tested problems within a few seconds, it's also worth running it besides other planners. While SGPlan is very inconsistent, it sometimes (see Table 4 and 2) produces plans with makespan values much lower than other planners.

[12] We use the same running time limit as in collecting results for Table 1

7 Conclusion and Future Work

In this paper, we describe our recent investigation into using model-based planning technology to solve the quantum circuit routing for QAOA applied to the Graph Coloring problem: temporal planner for solving the Routing and classical planner for solving the QI problem. Our empirical evaluation shows that temporal planners can solve problems with diverse setups effectively and compare reasonably to the best known analytical bounds on special cases. Qubit initialization as classical planning, utilising the Fast Downward planner, provides makespan improvement in the majority of problem configurations and provides an attractive alternative to manual qubit initialization using domain expert knowlege.

There are several directions we are pursuing in future work. First, we are working on running some plans produced by the temporal planners described in this paper on actual, physical hardware chips. Second, besides strengthening our current approach of using classical planning for qubit initialization with alternative PDDL encodings, some go beyond accomplishing just the PS phase by combining both the PS and MIX phases, we are also experimenting with several other approaches to QI such as posing it as a Quadratic Assignment(QA) [9] problem, which can then be solved using a quadratic programming solver such as CPLEX. While the initial result show poor scaling for the quadratic programming formulation, some heuristic approaches for solving the QA problem look promising. Third, we would like to analyze better the synergy between different temporal planning algorithms and the problem structures that are most suitable to them. Lastly, several portfolio approaches have been showing good performances at the recent International Planning Competition (IPC); we would like to investigate their performance on the Routing for Graph Coloring domain, and compare them to our current set of temporal planners used in this paper.

REFERENCES

[1] Deanna M. Abrams, Nicolas Didier, Blake R. Johnson, Marcus P. da Silva, and Colm A. Ryan, 'Implementation of the XY interaction family by calibration of a single pulse', *to be published*, (2019).

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al., 'Quantum supremacy using a programmable superconducting processor', *Nature*, **574**(7779), 505–510, (2019).

[3] J. Benton, Amanda Coles, and Andrew Coles, 'Temporal planning with preferences and time-dependent continous costs', in *Proceedings of the Twentieth-Second International Conference on Automated Planning and Scheduling (ICAPS-12)*, (2012).

[4] Kyle E. C. Booth, Minh Do, J. Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank, 'Comparing and integrating constraint programming and temporal planning for quantum circuit compilation', in *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS2018)*, pp. 366–374, (2018).

[5] Yixing Chen and Benjamin Wah, 'Temporal planning using subgoal partitioning and resolution in SGPlan', *Journal of Artificial Intelligence Research*, **26**, 323 – 369, (2006).

[6] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long, 'Forward-chaining partial-order planning', in *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*, (2010).

[7] AD Corcoles, A Kandala, A Javadi-Abhari, DT McClure, AW Cross, K Temme, PD Nation, M Steffen, and JM Gambetta, 'Challenges and opportunities of near-term quantum computing systems', *arXiv preprint arXiv:1910.02894*, (2019).

[8] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah, 'On the qubit routing problem', *arXiv preprint arXiv:1902.08091*, (2019).

[9] Burkard Rainer Ernst, Eranda Dragoti-Cela, P.M. Pardalos, and L.S. Pitsoulis, *The quadratic assignment problem*, volume 2, 241–337, Springer, 1998.

[10] Patrik Eyerich, Robert Mattmüller, and Gabriele Röger, 'Using the context-enhanced additive heuristic for temporal and numeric planning', in *Proceedings of the* 19^{th} *International Conference on Automated Planning and Scheduling*, pp. 318 – 325, (2009).

[11] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann, 'A quantum approximate optimization algorithm.', in *arXiv preprint arXiv:1411.4028*, (2014).

[12] Maria Fox and Derek Long, 'PDDL2.1: An extension to pddl for expressing temporal planning domains.', *Journal of Artificial Intelligence Research*, **20**, 61–124, (2003).

[13] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina, 'Planning through stochastic local search and temporal action graphs', *Journal of Artificial Intelligence Research*, **20**, 239 – 290, (2003).

[14] Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I Schuster, Henry Hoffmann, and Frederic T Chong, 'Partial compilation of variational algorithms for noisy intermediate-scale quantum machines', in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 266–278. ACM, (2019).

[15] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas, 'From the quantum approximate optimization algorithm to a quantum alternating operator ansatz', *Algorithms*, **12**(2), 34, (2019).

[16] Malte Helmert, 'The Fast Downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191—246, (2006).

[17] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo, 'Optimization of quantum circuit mapping using gate transformation and commutation', *Integration*, (2019).

[18] Gushu Li, Yufei Ding, and Yuan 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*, pp. 1001–1014. ACM, (2019).

[19] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete, 'Full-stack, real-system quantum computer studies: Architectural comparisons and design insights', *arXiv preprint arXiv:1905.11349*, (2019).

[20] Angelo Oddi and Riccardo Rasconi, 'Greedy randomized search for scalable compilation of quantum circuits', in *Proceedings of the Fifteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR2018)*, pp. 446–461, (2018).

[21] Bryan O'Gorman, William J. Huggins, Eleanor G. Rieffel, and K. Birgitta Whaley. Generalized swap networks for near-term quantum computing, 2019.

[22] Riccardo Rasconi and Angelo Oddi, 'An innovative genetic algorithm for the quantum circuit compilation problem', in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI2019)*, (2019).

[23] Silvia Richter and Matthias Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks.', *Journal of Artificial Intelligence Research*, **39**, 127–177, (2010).

[24] Davide Venturelli, Minh Do, Bryan O'Gorman, Jeremy Frank, Eleanor Rieffel, Kyle EC Booth, Thanh Nguyen, Parvathi Narayan, and Sasha Nanda, 'Quantum circuit compilation: An emerging application for automated reasoning', *ICAPS 2019 Workshop SPARK*, (2019).

[25] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank, 'Temporal planning for compilation of quantum approximate optimization circuits', in *Proceedings of The 26th International Joint Conference on Artificial Intelligence (IJCAI17)*, (2017).

[26] Vincent Vidal and Hector Geffner, 'Branching and pruning: An optimal temporal POCL planner based on constraint programming', *Artificial Intelligence Journal*, **170(3)**, 298—335, (2006).

[27] Zhihui Wang, Nicholas C Rubin, Jason M Dominy, and Eleanor G Rieffel, 'xy-mixers: analytical and numerical results for QAOA', *arXiv preprint arXiv:1904.09314*, (2019).

[28] Alwin Zulehner, Alexandru Paler, and Robert Wille, 'An efficient methodology for mapping quantum circuits to the IBM QX architectures', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (2018).