Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation

Kyle E. C. Booth[4,0], **Minh Do**[2,4], **J. Christopher Beck**[0], **Eleanor Rieffel**[1], **Davide Venturelli**[1,3], **Jeremy Frank**[2]
[0] Department of Mechanical & Industrial Engineering, University of Toronto
[1] Quantum Artificial Intelligence Laboratory, NASA Ames Research Center
[2] Planning and Scheduling Group, NASA Ames Research Center
[3] USRA Research Institute for Advanced Computer Science (RIACS)
[4] Stinger Ghaffarian Technologies (SGT Inc.)

Abstract

Recently, the makespan-minimization problem of compiling a general class of quantum algorithms into near-term quantum processors has been introduced to the AI community. The research demonstrated that temporal planning is a strong solution approach for the studied class of quantum circuit compilation (QCC) problems. In this paper, we explore the use of methods from operations research, specifically constraint programming (CP), as an alternative and complementary approach to temporal planning. We also extend previous work by introducing two new problem variations that incorporate important characteristics identified by the quantum computing community. We apply temporal planning and CP to the baseline and extended QCC problems as both stand-alone and hybrid approaches. The hybrid method uses solutions found by temporal planning to warm-start CP, leveraging the ability of temporal planning to find satisficing solutions to problems with a high degree of task optionality, an area that CP typically struggles with. These solutions are then used to seed the CP formulation which significantly benefits from inferred bounds on planning horizon and task counts provided by the warm-start. Our extensive empirical evaluation indicates that while stand-alone CP is not competitive with temporal planning, except for the smallest problems, CP in a hybrid setting is beneficial for all temporal planners in all problem classes.

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 since physical hardware has varying characteristics and architectures. These idealized quantum circuits must be compiled to specific hardware by adding additional gates that move qubit states to locations where the desired gate can act on them. Compilations that minimize the duration not only return results more quickly, but are vital to obtain results on near-term quantum hardware that does not support significant quantum error correction or fault tolerance: decoherence effects can destroy the computation in a short time. For this reason, it is critical to minimize computation duration.

Recently, use of temporal planners to compile quantum circuits was explored (Venturelli et al. 2017); machine instructions were modeled as PDDL2.1 durative actions, enabling domain-independent temporal planners to find a parallel sequence of conflict-free instructions 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.

In this paper, we extend the earlier work in three directions. First, we explore constraint programming (CP) as a complementary approach to temporal planning for quantum circuit compilation (QCC). Historically, operations research (OR) techniques have been the best approach for many combinatorial optimization problems and serve as the backbone of a number of planners. Our most significant investigation explores CP as an alternative to temporal planning for QCC. We use CP as either (1) a stand-alone approach or (2) working in tandem with temporal planners. Our main contributions are:

- A CP approach for QCC that is competitive with existing temporal planners on small problems, though not for larger problems.
- A hybrid planning/CP approach where temporal-planning solutions are used to warm-start the CP solver. Given the same amount of running time, our hybrid consistently outperforms both stand-alone temporal planning and CP approaches across all solvers and problem classes.

Second, we expand the previous problem definition (Venturelli et al. 2017) to include further optimization and additional constraints which reflect various realistic hardware architectures. Our expanded benchmarks include: (1) Initialization of qubit state locations, rather than starting from default locations and (2) crosstalk constraints, placing additional restrictions on gate operations.

Finally, we consider an expanded set of temporal planners for a more complete evaluation of temporal planning for QCC. Our extensive empirical evaluation shows that the additional constraints lead to a more diverse set of temporal planning benchmarks with different characteristics. The tested temporal planners, which utilize different planning frameworks, perform differently across problem variations belonging to the same problem class.

While there has been active development of software libraries to synthesize and compile quantum circuits from algorithm specifications (Wecker and Svore 2014; Smith,

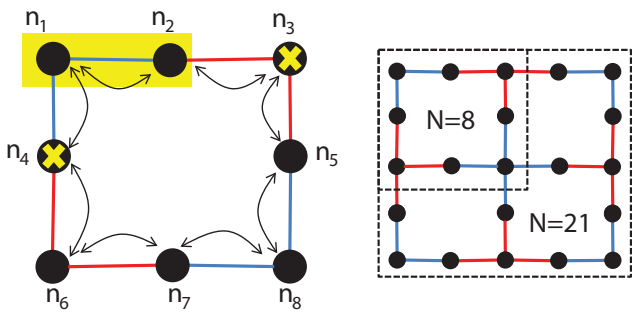

Figure 1: *Left*: A schematic for the 8-qubit chip design inspired by the prototype described in (Sete, Zeng, and Rigetti 2016) and used in (Venturelli et al. 2017) and in our numerical experiments. Available 2-qubit gates are represented by colored edges (swap capability symbolized by double arrows). Each color is associated to a distinct duration of the P-S gate (three for blue and four for red, in normalized clock cycles). Additionally, 1-qubit *mixing* gates of unit duration are present at each qubit (black dot). The yellow crosses on qubits 3 and 4 visualize the disabled qubits during the action of a gate between qubits 1 and 2 when *crosstalk* constraints are considered. *Right*: Dashed boxes indicate the 2 different chip sizes used in our empirical evaluation (see Sec. 6).

Curtis, and Zeng 2016; Steiger, Häner, and Troyer 2016; Devitt 2016; Barends et al. 2016), few approaches have been explored for compiling idealized quantum circuits to realistic quantum hardware with a specific focus on swap gate insertions (Beals et al. 2013; Brierly 2015; Bremner, Montanaro, and Shepherd. 2016), targeting algorithms that could be run in the near-term (Guerreschi and Park 2016).

The rest of our paper is structured as follows: in Section 2, we provide some background on QCC and the existing approach to solving QCC using domain-independent temporal planners. Next, Section 3 discusses our approach in using CP as a stand-alone solver for QCC. Section 4 discusses planning horizon and task quantities. Section 5 describes our novel planning-CP hybrid approach. Section 6 details our empirical evaluation results. We end the paper with Section 7 our conclusion, and potential future work.

2 Quantum Circuit Compilation

General quantum algorithms are often described in an idealized architecture in which a gate acts on any subset of available qubits. However, in an actual superconducting qubit architecture, such as the ones manufactured by IBM (IBM 2017), Rigetti (Reagor et al. 2017), Google (Neill et al. 2017) and UC Berkeley (Ramasesh et al. 2017), physical constraints impose restrictions on pairs of qubits that support gate interactions. Qubits in these quantum processors can be thought of as nodes in a planar graph, with 2-qubit quantum gates associated to edges and single qubit quantum gates associated with nodes. In Fig. 1 we present a *model chip* that is used in our benchmarks. Following the most common choice for benchmarks in the literature, the *model quantum algorithm* used is a variant of the "Quantum Alternating Operator Ansatz" (Hadfield et al. 2017) (also known as "Quantum Approximate Optimization Algorithm" (QAOA) (Farhi, Goldstone, and Gutmann. 2014)) applied to the $\mathcal{NP}$-Hard problem of Max-Cut. As described in (Venturelli et al. 2017) this algorithm is specified by a single type of 2-qubit gate, the *phase separation* (p-s) gate, which needs to be applied to a specific set of quantum variables depending on the problem instance. An example of problem instance can be found in Fig. 2-a where the edges of the graph indicate the required p-s gates. In the model chip, gate colors (red or blue) indicate different p-s gate durations in terms of clock cycles. A sequence of swap gates moves the information content (*qubit states*) of two distant qubits to a location where a desired p-s 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 the edge, however, in our benchmarks we assume swap gates are available on each edge with constant duration equal to 2 clock cycles.

Our benchmark algorithm works by repeating the same circuit $\mathcal{P}$ times, interleaving each run by a *mixing* phase. In the mixing phase, a set of single qubit mixing gates are applied. These gates are located at each qubit. All p-s gates that involve a specific qubit state must be carried out before the mixing on that state can be applied and the second p-s stage initiated. As in previous work, we consider $\mathcal{P} \in \{1, 2\}$.

2.1 Definitions

We let the set of qubits in the quantum circuit be represented as $N := \{n_1, n_2, \ldots, n_\alpha\}$ and the set of qubit states be represented as $Q := \{q_1, q_2, \ldots, q_\beta\}$. Each qubit, $n_i \in N$, starts in its corresponding (by index) state, $q_j \in Q$. The initial configuration of the circuit has qubit state j mapped to qubit i, for $i = j$ (Note: in the problems studied, $\alpha = \beta$). An integer value T is the scheduling horizon. Determining an appropriate T is discussed in Section 4.1.

We let S represent the set of swap gates in the circuit architecture, $S := \{s_1, s_2, \ldots, s_\gamma\}$, where each gate, $s_k \in S$, involves a qubit pair, $\langle n_i, n_j \rangle$. Similarly, we let P represent the set of p-s gates, $P := \{p_1, p_2, \ldots, p_\delta\}$, where each of these gates, $p_\ell \in P$ involves a pair of qubits. We define the set of swap and p-s gates that involve qubit $n_i \in N$ as $S(i)$ and $P(i)$, respectively. Swap and p-s gates have distinct durations for their activation (τ_{swap} and τ_ℓ, respectively), with p-s gate duration depending on the class of the gate, visualized as different colors in Figure 1 (thus, duration $\tau_\ell \in \{\tau_{red}, \tau_{blue}\}, \forall p_\ell \in P$). When the problem involves multiple p-s stages, mixing gates are available at each node in the architecture with a duration τ_{mix}.

The set of problem goals, defined as $G := \{g_1, g_2, \ldots, g_\epsilon\}$, encode the specific qubit state pairs that need p-s gates applied to them. Each goal, $g_o \in G$, identifies a pair of qubit states, $\langle q_i, q_j \rangle$. To achieve the goal, these quantum states must be adjacent in the architecture graph (in the case of the studied architecture, all adjacent qubits have a connecting a p-s gate). The p-s gate used for goal activation is a decision variable.

Example: Given the 8-qubit architecture in Figure 1 with each qubit $n_i \in N$ initially associated to the qubit state

$q_j \in Q$ (with $i = j$), let us assume that the idealized circuit requires the application of a p-s gate to the states q_2 and q_4. The sequence of gates to achieve the goal are:

$$\{\text{SWAP}_{n_4,n_1}, \text{SWAP}_{n_2,n_3}\} \rightarrow \text{SWAP}_{n_1,n_2} \rightarrow \text{PS}^{red}_{n_2,n_3}$$

The sequence takes $2\tau_{swap}$ + τ_{red} clock cycles where $\tau_\star$ represents the duration of the $\star$-gate.

2.2 Temporal Planning for QCC

The QCC problem can be modeled as a temporal planning problem, utilizing the standard planning domain definition language (PDDL) (Venturelli et al. 2017), as follows:

- Predicates are used to model the location of each qubit state and if the pre-defined goal requirement on a pair of qubit states has been achieved or not.
- The swap and p-s gates are modeled as temporal actions with: (1) conditions representing constraints on whether or not the involved qubit states residing on connecting qubit pairs and if the required gates have not been already executed; and (2) effects representing the new locations of the qubit states and that the desired p-s goals have been achieved.
- The standard objective function of minimizing the total plan makespan matches well with the desired goal of minimizing the circuit duration in the QCC problems studied.

While the basic mapping is outlined above, there are additional constraints and actions involved with different variations of the QCC problem (e.g., multiple p-s stages), we refer to (Venturelli et al. 2017) for the details.

2.3 Extensions

In this paper, we target QCC problems beyond the one addressed in (Venturelli et al. 2017). With the addition of *qubit state initialization* (QCC-I) and *crosstalks* (QCC-X) problem variations, we allow the implemented techniques to solve a unified problem that originally required two independent steps and handle a type of constraint that is often present in existing hardware.

Qubit State Initialization (QCC-I) In the previously studied QCC problem qubit states are assigned their initial locations on the chip before problem solving (e.g., qubit state $q_j \in Q$ is initially assigned to qubit $n_i \in N$). Here we both assign the initial locations of the qubit states and find the sequence of gates to achieve the goals. Modeling this initialization step in PDDL is rather straight forward.

- In the initial state: all qubits are "empty" and all qubit state locations are undetermined.
- Action $a_{i,j}$ initializes the location of qubit state $q_j \in Q$ on qubit $n_i \in N$ if: (1) q_j has not been initialized and (2) n_i is still empty.
- Action a_{init_finish} finalizes the initialization process with the condition that all qubit states have been initialized. After this action finishes executing, then all actions of the original QCC problem can start.

Crosstalk (QCC-X) In the existing problem definition, any given qubit can be involved with a single gate operation at any given time. For certain hardware architectures, *crosstalk* constraints further restrict qubit involvement. We represent this family of constraints inspired by the technology implemented by the devices manufactured by Google (Boxio 2016).[1] Specifically, when a given qubit $n_i \in N$ is involved in a gate operation, any qubit adjacent to it in the architecture is prevented from engaging in any gate operation. For example, if a 2-qubit gate operation is carried out between $\langle n_1, n_2 \rangle$ in Figure 1, then no gate operation involving n_3 (connected to n_2) or n_4 (connected to n_1) can be started until the $\langle n_1, n_2 \rangle$ operation is complete.

To model crosstalk constraints in PDDL, we introduce:

- A new predicate $crosstalk(n_i)$ to indicate if n_i is currently disabled by a gate operating on an adjacent qubit.
- An action representing a gate operation on a pair of adjacent qubits $\langle n_i, n_j \rangle$ will: (1) require $(not\ (crosstalk(n_i)) \wedge (not\ (crosstalk(n_j)))$ as action pre-conditions, and (2) for every qubit n_k that is connected to either n_i or n_j, $crosstalk(n_j)$ is part of the effect list of the start of the action.

3 Constraint Programming for Quantum Circuit Compilation

Operations research (OR) investigates solving many combinatorial optimization problems closely related to planning. Various techniques developed in OR are also utilized in existing planners as off-the-self solvers (Kautz and Selman 1999; Do and Kambhampati 2000; van den Briel and Kambhampati 2005), routines to solve key sub-problems (Benton, Coles, and Coles 2012) in decomposed approaches, models to calculate heuristic values (Pommerening et al. 2015; Piacentini et al. 2018), or as inference techniques customized for planning (Vidal and Geffner 2006).

Given their performance, we investigate the use of OR methods, namely *constraint programming* (CP), as an alternative and complementary approach to temporal planning for QCC problems. Preliminary efforts to develop a mixed-integer programming (MIP) formulation yielded less promising results and we elected not to pursue this MIP models for this work. In this section, we detail a CP model for the QCC problem.

3.1 Decision Variables

As is common in CP, our formulation utilizes continuous, integer, and optional/mandatory interval decision variables. An optional interval variable, var, is a rich variable type whose possible values are defined over a convex interval: $var := \{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$, where the variable takes on the value $\perp$ if it is not present in the solution[2] and

[1] Similar constraints are present in the devices by IBM and by UC Berkeley. These constraints could model the loss of circuit fidelity due to spurious uncompensated electromagnetic effects or to uncalibrated interactions. However, parallel execution of adjacent circuitry can also be prevented by design.

[2] Mandatory interval variables must be present in the solution.

s and e represent the start and end values of the interval. Such interval variables are a natural way to model swap, p-s, and mixing gate tasks, as they have duration and need to be assigned a start time. The variable Pres(var) takes on a value of 1 if the interval variable var is present in the solution. Constraints are only active over present interval variables. If present, Start(var), End(var), and Length(var) return the integer start and end times, as well as the length, of the interval variable var. Additionally, we can assign interval variables to *sequences* which model relationships such as Pre(var), which returns the interval variable preceding var in a candidate solution. A key limitation to CP technology, as opposed to temporal planning, is that it can only reason over variables in the model. For QCC problems, the number of times a particular swap or p-s gate will be used in a solution plan is unknown *a priori*, and thus we must define upper bounds for these values and allocate these quantities of tasks (instantiated as optional interval variables) to the model. We define bounds on swaps, $\mathcal{U}_{swap}$, and p-s activations, $\mathcal{U}_{ps}$, for each gate in the architecture in Section 4.2.

We model the problem in CP with an *event-based* formulation, tracking qubit state after each swap, p-s, or mixing gate task that involves that particular qubit. We define the set of all events potentially involving qubit $n_i \in N$ as E_i, including a dummy event for qubit state initialization.

The decision variables with associated domains (permissible values) used in our formulation are:

- $C_{max} := (continuous)$ Makespan of the generated plan and objective function value of the formulation, with possible values in $0 \leq C_{max} \leq T$.
- $x_{i,j} := (integer)$ State of qubit $n_i \in N$ after task $j \in E_i$. Each of these variables takes on a value in the set Q of available qubit states, namely $x_{i,j} \in \{1, 2, \ldots, |Q|\}, \forall j \in E_i, n_i \in N$.
- $y_{k,m} := (optional\ interval)$ Swap task m for swap gate $s_k \in S$. If present, has a start time Start($y_{k,m}$) $\in [0, T]$ and duration Length($y_{k,m}$) $= \tau_{swap}$. The set of optional swap tasks available for swap gate $s_k \in S$ is defined as: $\bar{y}_k := \{y_{k,1}, y_{k,2}, \ldots, y_{k,\mathcal{U}_{swap}}\}$.
- $z_{\ell,n} := (optional\ interval)$ P-s task n for p-s gate $p_\ell \in P$. If present, has start time Start($z_{\ell,n}$) $\in [0, T]$ and duration Length($z_{\ell,n}$) $= \tau_{ps}$, where $\tau_{ps} \in \{\tau_{red}, \tau_{blue}\}$, as per the architecture. The set of optional p-s tasks available for p-s gate $p_\ell \in P$ is defined as: $\bar{z}_\ell := \{z_{\ell,1}, z_{\ell,2}, \ldots, z_{\ell,\mathcal{U}_{ps}}\}$.
- $Z_o := (interval)$ Mandatory goal p-s task for goal $g_o \in G$. Start time Start(Z_o) $\in [0, T]$, duration Length(Z_o) $\in \{\tau_{red}, \tau_{blue}\}$ and end time End(Z_o). The makespan objective is the time of the latest completion time of these variables, namely: $C_{max} := \max_{g_o \in G} \big(\text{End}(Z_o)\big)$.

In problems with two p-s stages, we include the following additional decision variables for mix gates:

- $\omega_{i,j} := (optional\ interval)$ Task for mixing qubit state $q_j \in Q$ at qubit $n_i \in N$. If present, has start time Start($\omega_{i,j}$) $\in [0, T]$ with duration Length($\omega_{i,j}$) $=\tau_{mix}$. The set of optional mixing tasks available for qubit $n_i \in N$ is defined as: $\bar{\omega}_i := \{\omega_{i,1}, \omega_{i,2}, \ldots, \omega_{i,\beta}\}$.
- $\Omega_j := (interval)$ Mandatory mixing task for qubit state $q_j \in Q$. Has start time Start(Ω_j) $\in [0, T]$ duration Length(Ω_j) $= \tau_{mix}$, and end time, End(Ω_j), representing when the mixing of state $q_j \in Q$ is complete.

3.2 Formulation Objective and Constraints

With the problem parameters, decision variables, and associated domains defined, we detail our event-based CP formulation in Eqns. (1 - 12). Constraints (1 - 9) are required for one and two-stage p-s, while Constraints (10 - 12) are only required for two-stage p-s problems.

Objective (1) represents the problem objective which is to minimize the makespan, C_{max}, of the circuit compilation. The secondary objective, reduced in weight by a sufficiently small value ξ, minimizes the number of swap tasks.[3] The addition of this component was found to improve solver performance while remaining a reasonable objective for QCC problems. Constraint (2) initializes the qubit states to their required initial values and Constraint (3) requires that solution C_{max} be greater than the end time all goal variables.

We use a number of *global constraints* (van Hoeve and Katriel 2006). These constraints are defined over a set of variables and encapsulate frequently recurring combinatorial substructure in a way that improves the branch-and-infer search implemented by CP solvers. Constraint (4) uses the *NoOverlap* global constraint (Baptiste, Le Pape, and Nuijten 2012) to perform incomplete, efficient domain filtering on the start times of the interval variables. This modeling treats each qubit, $n_i \in N$, as a unary capacity resource, and ensures that swap, p-s, and mix gates are activated in such a way that two gates involving the same qubit are never active at the same time. The set E_i, consisting of all tasks potentially involving qubit $n_i \in N$, is defined as: $E_i := \{\dot{x}\} \cup \{\bar{y}_k : s_k \in S(i)\} \cup \{\bar{z}_\ell : p_\ell \in P(i)\} \cup \bar{\omega}_i$, where $\dot{x}$ is a dummy task.

Constraint (5) makes use of the *Alternative* global constraint (Laborie 2009), which links interval variables to a set of optional interval variables, enforcing that only one variable from the optional set can be present and the start time must coincide with the mandatory variable. We use this constraint to maintain the relationship between the goal variables, Z_o, and the optional p-s variables, $z_{\ell,n}$. For each p-s gate, the $z_{\ell,n}$ tasks are ordered such that they coincide with a single goal, and thus $|G| = |\bar{z}_\ell|, \forall p_\ell \in P$. Each goal activates a single p-s gate task across the set of p-s gates.

Constraint (6) implements qubit state updates when a swap interval variable is present, swapping the states of the qubits involved in the corresponding physical swap gate. The term $pre_i(y_{k,m})$ returns the task previous to swap task $y_{k,m}$ in the sequence for qubit $n_i \in N$, allowing the modeling of qubit state swap between the qubit pair, $\langle i, j \rangle$, involved in gate $s_k \in S$. Constraint (7) models a similar relationship for p-s gate tasks. We note that while swap tasks result in an exchange of states between qubits n_i and n_j, after a p-s task qubit states remain unchanged.

[3] This value is subtracted from the objective when comparing to temporal planning approaches for consistency.

Minimize:

$$C_{max} + \xi \cdot \sum_{s_k \in S} \sum_{y_{k,m} \in \bar{y}_k} \text{Pres}(y_{k,m}) \quad (1)$$

Subject to:

$$x_{i,0} = i, \quad \forall i \in \{1, 2, \ldots, \alpha\} \quad (2)$$

$$C_{max} \geq \text{End}(Z_o), \quad \forall g_o \in G \quad (3)$$

$$\text{NoOverlap}(E_i), \quad \forall n_i \in N \quad (4)$$

$$\text{Alternative}(Z_o, [z_{1,o}, \ldots, z_{\delta,o}]), \quad \forall g_o \in G \quad (5)$$

$$\text{Pres}(y_{k,m}) \rightarrow (x_{i,y_{k,m}} = x_{j,pre_j(y_{k,m})}) \wedge (x_{j,y_{k,m}} = x_{i,pre_i(y_{k,m})}), \quad \forall y_{k,m} \in \bar{y}_k, \langle i, j \rangle \in s_k, s_k \in S \quad (6)$$

$$\text{Pres}(z_{\ell,n}) \rightarrow (x_{i,z_{\ell,n}} = x_{i,pre_i(z_{\ell,n})}) \wedge (x_{j,z_{\ell,n}} = x_{j,pre_j(z_{\ell,n})}), \quad \forall z_{\ell,n} \in \bar{z}_\ell, \langle i, j \rangle \in p_\ell, p_\ell \in P \quad (7)$$

$$\text{Pres}(z_{\ell,n}) \rightarrow (x_{i,pre_i(z_{\ell,n})} = g_{\ell,1} \wedge x_{j,pre_j(z_{\ell,n})} = g_{\ell,2}) \vee (x_{i,pre_i(z_{\ell,n})} = g_{\ell,2} \wedge x_{j,pre_j(z_{\ell,n})} = g_{\ell,1}), \quad \forall z_{\ell,n} \in \bar{z}_\ell, \langle i, j \rangle \in p_\ell, p_\ell \in P \quad (8)$$

$$\text{Pres}(y_{k,m}) \geq \text{Pres}(y_{k,m+1}), \quad \forall y_{k,m} \in \bar{y}_k \setminus y_{k,\mathcal{U}_{swap}}, s_k \in S \quad (9)$$

$$\text{Alternative}(\Omega_j, [\omega_{1,j}, \ldots, \omega_{\alpha,j}]), \quad \forall q_j \in Q \quad (10)$$

$$\text{Start}(\Omega_j) \geq \text{End}(Z_o), \quad \forall g_o \in G(j), q_j \in Q \quad (11)$$

$$\text{End}(\Omega_j) \leq \text{Start}(Z_o), \quad \forall g_o \in G'(j), q_j \in Q \quad (12)$$

Constraint (8) ensures that if a particular p-s gate task is present, $z_{\ell,n}$, the states of the qubits involved, prior to the application of the p-s gate, match the specific goal that the p-s gate is matched with (goals being ordered to match the optional p-s gate tasks at any physical p-s gate location). In this constraint, the term $g_{\ell,1}$ represents the first qubit state required by goal $g_\ell \in G$, and $g_{\ell,2}$ the second.

To help remove some of the symmetries in the model, Constraint (9) specifies that homogeneous optional swap tasks must be used lexicographically.

For problems that have two stages of phase separation, a mixing gate must be applied to each qubit state after all the goals that utilize that state are achieved, and then the goals must be repeated. To achieve this, we introduce a second goal set, $G' := \{g_{\epsilon+1}, g_{\epsilon+2}, \ldots, g_{2\cdot\epsilon}\}$, which duplicates the first. We let the sets $G(j)$ and $G'(j)$ denote the goals from G and G', respectively, that involve qubit state $q_j \in Q$. Constraint (10) ensures that only one of the optional mixing tasks is used for each mixed qubit state and Constraints (11) and (12) ensure that the mixing tasks separate the two p-s goal sets. Additionally, goal requirements are amended to include the duplicated goal set, G'.

3.3 Qubit Initializations and Crosstalks

The QCC problem extensions introduced in Section 2 require minor alterations to the CP model.

QCC-I In the baseline and crosstalk problem variants, Constraint (2) is applied unchanged. However, in the qubit initializations problem variant this constraint is removed and replaced with the following:

$$\text{AllDifferent}(x_{1,0}, x_{2,0}, \ldots, x_{\alpha,0}) \quad (13)$$

The removal of Constraint (2) allows the solver to select initial values for qubit states, and the addition of Constraint (13) enforces that the initial states on all the qubits be different, ensuring that all qubit states are present on the chip.

QCC-X In the crosstalk variant of the problem, the qubits that can simultaneously participate in gate activations are further constrained. Constraint (4) is adjusted such that the sets $S(i)$ and $P(i)$ for a given qubit $n_i \in N$ include the gates that involve adjacent qubits to n_i as well.

4 Setting Bounds

4.1 Planning Horizon

Our CP formulation can be implemented using a horizon set to infinity, however, it was observed that smaller values improve performance.

Let ψ be the length of the side of the chip ($\psi = 3$ for 8-qubits, and $\psi = 5$ for 21-qubits), $\tau_{p-s}^{max} = max(\tau_{red}, \tau_{blue})$ be the maximum possible p-s gate duration, and $\phi = (2 \cdot \psi) - 3$ be the maximum number of swaps required to bring any two qubit states to a pair of adjacent qubits.

Lemma 1. *For $\mathcal{P} = 1$ problems, $T = |G| \cdot (\phi \cdot \tau_{swap} + \tau_{p-s}^{max})$ is an upper bound on the optimal plan makespan.*

Proof. Imagine that we are only able to perform a *single* task at a time. There are two components to achieving any goal: i) moving the required qubit states to adjacent qubits, and ii) applying a p-s gate task.

For a single goal, the worst case scenario would have the required states located on the opposite sides of the architecture (e.g., located on n_1 and n_8 in Figure 1), requiring a minimum of ϕ swaps to place the required states adjacent to each other. Then, we must apply a p-s gate, which in the worst case will take a duration of τ_{p-s}^{max}. We can perform all tasks for $|G|$ goals in sequence, leading to a plan with makespan no worse than: $T = |G| \cdot (\phi \cdot \tau_{swap} + \tau_{p-s}^{max})$. □

We use this T value as the planning horizon parameter in all of our CP experiments for single p-s stage problems.[4]

4.2 Swap and P-S Gate Tasks

For the scheduling formulation, we determine the number of activation tasks to be allocated per physical swap gate, $\mathcal{U}_{swap}$. If we consider achieving each goal sequentially, we could potentially have to move a qubit state through the entire architecture to become adjacent to the other qubit state. In this case, each swap gate is used once for each goal, yielding $\mathcal{U}_{swap} = |G|$. Note that, although this value is observed to perform well empirically, the circuit can work towards goals in parallel, leading (potentially) to optimal solutions

[4] A trivial extension of this proof is used to yield a scheduling horizon bound applicable to $\mathcal{P} = 2$.

that use more than $|G|$ swaps per physical gate; we will explore this in future work.

In contrast to swap tasks, our CP formulation allocates exactly one p-s task to each physical p-s gate for each goal, $g_o \in G$, resulting in $\mathcal{U}_{ps} = |G|$. This allows for the case that all of the goals are achieved using the same physical p-s gate. Note that the p-s gate tasks are actually ordered such that the first task corresponds to the first goal, and so forth (as noted in the CP model).

5 Hybrid Approach with Temporal Planning and Constraint Programming

Temporal planning is able to find a satisficing plans for most of the instances, often fairly quickly. The CP model, while performing well on small problems, struggles to find solutions for larger problems due to high levels of task optionality, reducing the amount of inference that can be performed. In 8-qubit problems with $\mathcal{P} = 1$, CP must generate $(|S| \cdot \mathcal{U}_{swap}) = (8 \cdot 8) = 64$ optional swap tasks to represent the problem, and for 21-qubit problems, $(24 \cdot 21) = 504$ tasks. In each of these cases, final solutions use roughly 10% of these tasks. Similar observations are made for p-s and mixing tasks.

Utilizing the ability of the temporal planners to find quality solutions early on in the search, we integrate the two techniques via a *warm-start* procedure, a fairly common boosting technique in OR (Kramer, Barbulescu, and Smith 2007; Beck, Feng, and Watson 2011), where solutions found by the temporal planner are encoded into a CP starting point. To implement the warm-start, a solution translator converts PDDL solutions to CP model variable values. An example of a partial translation is provided in Table 1. In this example, we assume that goal g_1 involves $\langle q_1, q_2 \rangle$, and that the gates between n_1 and n_2 are labeled the first gates in the architecture.[5] Note that swap and goal gates in the PDDL solution format must be translated into many variable values in the CP model, due to task optionality. CP can benefit from partial warm-starts (fixing a subset of variable values), however, empirical testing demonstrated that a full warm-start (all variable values fixed) worked best. Additional translation is required to achieve a full warm-start, but is omitted due to space constraints.

PDDL Solution	CP Warm-Start
`; Time 12.20`	CP_timeLimit = (timeLimit - 12.20)
`; Makespan 25.00`	sol.setValue($C_{max} = 25$)
`0:(GOAL_1_2 Q1 Q2)[3]`	sol.setStart($Z_1 = 0$), sol.setStart($z_{1,1} = 0$), sol.setAbsent($z_{\ell,1}$), $\forall p_\ell \in P \vert \ell > 1$
`7:(SWAP_1_2 Q1 Q2)[2]`	sol.setStart($y_{1,1} = 7$), sol.setAbsent($y_{1,m}$), $\forall y_{1,m} \in \bar{y}_1 \vert m > 1$

Table 1: Warm-start translation, PDDL solution to CP.

With the planning warm-start as an incumbent solution, the CP solver can immediately derive a tighter bound on planning horizon, T, equal to the makespan of the solution found by the planner. CP will then search for an improving solution, using the bounds inferred from the starting solution. Additionally, the solver can explore the neighbourhood of the seed solution in attempts to find similar solutions of higher quality, identifying high-impact variables set by the warm-start and exploring the search space beneath them while maintaining the ability to backtrack and investigate completely different areas as well.

In this work, we take the *highest quality* solution found by the temporal planner (from the standalone planner experiments) and use it to warm-start CP. We allot CP the remaining runtime (from the time that temporal planning finds its last solution until the time limit), allowing us to directly compare the solution quality of the hybrid approach versus the standalone planner.

6 Empirical Evaluation

6.1 Setup

Problem Instances We start with the previously studied QCC problem benchmark set (Venturelli et al. 2017) that compiles Quantum Approximate Optimization (QAOA) circuits (Farhi, Goldstone, and Gutmann. 2014) for MaxCut to the architecture inspired by the Rigetti Coputing's quantum computer (Sete, Zeng, and Rigetti 2016) (refer to Figure 1). Additionally:

- Most planners cannot solve $|N| = 40$ qubit problems. As such, we study two problem sizes: $|N| = 8$ and $|N| = 21$.
- Through preliminary investigation, no meaningful difference was observed for the empirical evaluation between 90% and 100% qubit utilizations, so we conduct our evaluation solely on problems with 100% qubit utilization.
- *Expansion:* We solve each benchmark instance using one of three problem variations: (1) the baseline problem, (2) QCC-I, and (3) QCC-X.

In total, we document results from five temporal planners, CP Optimizer, and our hybrid approach on nine sets of 50 problems each, for the total of 4,950 data points.

Software In addition to TFD (Eyerich, Mattmüller, and Röger 2009), LPG (Gerevini, Saetti, and Serina 2003), and SGPlan (Chen and Wah 2006), temporal planners used in the previous work (Venturelli et al. 2017), we also include results for two additional planners: CPT (Vidal and Geffner 2006) and POPF (Coles et al. 2010). CPT is an optimality-focused planner that uses CP inference techniques in a partial-order planning framework. POPF combines forward-state-space search with the partial-order planning framework to find flexible plans. We use the commercial software CP Optimizer to represent and solve our CP model.[6]

[5] For illustrative purposes, we assume swap gate 1-2 is only used once in the solution, incurring the absence of swap gates.

[6] Experiments are implemented in C++ on an Intel Core i7-2670QM with 8GB of RAM running Ubuntu 14.04 LTS Linux. We use CP Optimizer version 12.6.3 single-threaded with default search and extended NoOverlap inference (all other inference default). All planners, except CPT, used the same machine. CPT, due to software issues, was run on a RedHat Linux 2.4Ghz machine with 8GB RAM.

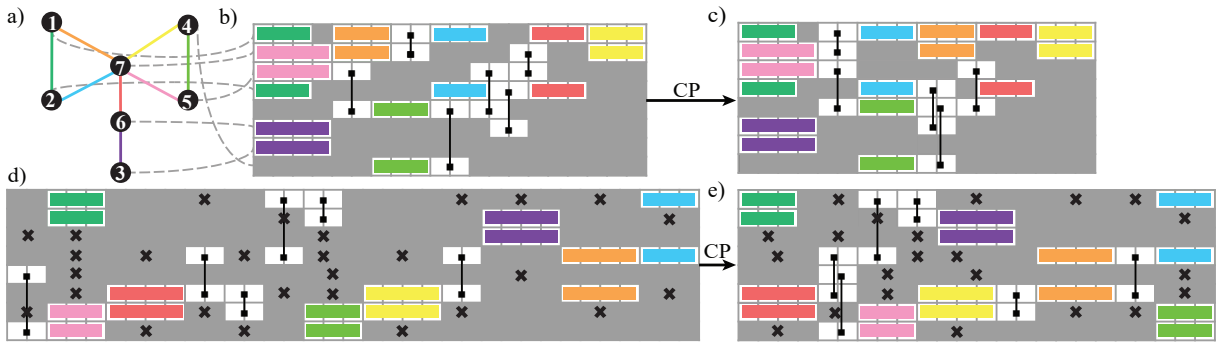

Figure 2: $a)$ Problem instance for $|N| = 8$ (using 7 qubits) for the QAOA-MaxCut model algorithm. Colored edges represent p-s gate executions between corresponding quantum states. $b) - e)$: Compilation plans obtained for the $\mathcal{P} = 1$ problem instance in (a). Clock cycles are on the horizontal axis and qubit locations are arranged on the vertical axis, sorted according to Figure 1. Colored boxes represent executed p-s gates, and swap gates by white boxes with vertical lines. (b) LPG planner compilation for QCC-I, where qubit state initialization is indicated by dashed lines. (d) LPG compilation for QCC-X: crosses identify disabled regions due to crosstalk constraints. (c) and (e), respectively, show the improvements when CP uses (b), (d) as a warm-start.

Three of the planners tested (LPG, TFD, and POPF) are anytime planners. These continue to return plans of gradually better quality until the alloted run-time is over. SGPlan returns a single satisficing solution, often very quickly. CPT, as designed, returns a single solution in the given run-time, which it attempts to prove optimal. CP Optimizer is an anytime solver and looks to improve solution quality over time.

6.2 Analysis

The results of our empirical analysis on the temporal planners, CP, and hybrid are summarized in Table 2.

Temporal Planning Overall, only LPG and POPF were able to consistently find solutions for all of the problem sets. CPT, a planner focusing on proofs of optimality, was able to only return solutions on the two problem sets with the smallest anticipated makespan values; this is not surprising given that CPT works by setting a bound B on plan horizon, increasing this value if no plan is found and tightening B if a plan is found. This multi-step procedure tends to be more time consuming as the optimal plan makespan increases.

SGPlan struggled with the newly introduced QCC-I, consistently returning an internal error, and also failed one QCC-X set. As it only returns a single satisficing solution the solution quality of this planner is typically lower than the other anytime planners. On problems for which SGPlan is able to find a satisficing solution, it typically does so very quickly (less than 2 seconds of runtime).

TFD was the best overall performer in the previous study (Venturelli et al. 2017), but the two new variations lead to totally opposite overall results. While it still performs very well in QCC-X,[7] it performs very poorly for QCC-I problem sets. The main reason for this is not clear, but it seems initialization actions confused the heuristics of TFD.

[7]TFD returned invalid solutions for nine $|N|$=21, $\mathcal{P}1$, QCC-X problems, violating crosstalk constraints between related swap gates. Hence 41/50 problems for this entry.

LPG and POPF perform the most consistently across most of the benchmark sets; LPG is the only planner that solved all 450 problems. POPF is the most consistent at finding good quality solution across all variations and problem sizes while LPG tends to do better in smaller problem sets but worse in more complex cases.

Constraint Programming Overall, the stand-alone CP approach is competitive with temporal planning in the three smaller $|N|$=8, $\mathcal{P}1$ problems sets, doing particularly well on QCC and QCC-X. The crosstalk constraints within QCC-X increase the scope of the NoOverlap constraints, enhancing inference that the solver can perform on variable start times, leading to strong performance on these smaller problems. The QCC-I proved difficult for CP as the lack of initially defined qubit states was more likely to lead the search away from candidate solutions. The large number of optional tasks incurred by larger problems overwhelm the approach and lead to the poor overall performance, as evidenced by the CP's inability to find any solutions for some problem sets.

Hybrid Approach Across all problem classes, except for the smallest problems where CPT was able to find proven optimal solutions, the planning-CP hybrid *always* results in better plans. A visualization of makespan improvement with the hybrid method over LPG stand-alone is provided in Figure 2. In the figure, both CP solutions seem to have explored the search space under the p-s task location variables as set by the planning warm-start, resulting in similar, yet condensed, compilations.

Table 3 shows the average improvement across different planners. Though the hybrid approach benefits all planners on all problem sets, the level of improvement varies. The approach consistently provides large performance boosts to SGPlan. As discussed, the low solution quality and ample time given to CP with a warm-start from SGPlan leads to this large improvement. There is little consistency in performance across the other planners (LPG, TFD, and POPF) except that: (1) for the smallest problem size $|N|$=8, $\mathcal{P}1$,

	LPG		TFD		SGPlan		POPF		CPT		CP
	Alone	Hybrid	Alone	Hybrid	Alone	Hybrid	Alone	Hybrid	Alone	Hybrid	Alone
$\|N\|$=8, $\mathcal{P}1$, QCC	0.93 (50)	0.95 (50)	0.87 (50)	0.93 (50)	0.63 (50)	0.92 (50)	0.83 (50)	0.85 (50)	**1.00 (50†)**	N/A	0.91 (50)
$\|N\|$=8, $\mathcal{P}1$, QCC-I	0.82 (50)	0.87 (50)	0.62 (2)	0.80 (2)	Error	N/A	0.80 (50)	0.82 (50)	**1.00 (50/43†)**	1.00 (50)	0.67 (42)
$\|N\|$=8, $\mathcal{P}1$, QCC-X	0.70 (50)	0.95 (50)	0.91 (50)	**0.96 (50)**	0.61 (50)	0.94 (50)	0.77 (50)	0.79 (50)	0.00 (0)	N/A	0.93 (50)
$\|N\|$=8, $\mathcal{P}2$, QCC	0.79 (50)	0.87 (50)	0.91 (38)*	0.95 (38)	0.72 (50)	**0.88 (50)**	0.85 (50)	0.87 (50)	0.00 (0)	N/A	0.78 (11)
$\|N\|$=8, $\mathcal{P}2$, QCC-I	0.70 (50)	0.80 (50)	0.00 (0)	N/A	Error	N/A	0.89 (50)	**0.91 (50)**	0.00 (0)	N/A	0.00 (0)
$\|N\|$=8, $\mathcal{P}2$, QCC-X	0.56 (50)	**0.88 (50)**	0.81 (50)	0.84 (50)	Error	N/A	0.67 (43)	0.78 (43)	0.00 (0)	N/A	0.63 (46)
$\|N\|$=21, $\mathcal{P}1$, QCC	0.82 (50)	0.86 (50)	0.61 (50)	0.72 (50)	0.64 (50)	0.72 (50)	0.92 (50)	**0.94 (50)**	0.00 (0)	N/A	0.39 (41)
$\|N\|$=21, $\mathcal{P}1$, QCC-I	0.61 (50)	**0.63 (50)**	0.00 (0)	N/A	Error	N/A	0.96 (42)	0.99 (42)	0.00 (0)	N/A	0.43 (7)
$\|N\|$=21, $\mathcal{P}1$, QCC-X	0.40 (50)	0.71 (50)	0.70 (41)*	0.89 (41)	0.61 (50)	**0.90 (50)**	0.59 (19)	0.64 (19)	0.00 (0)	N/A	0.00 (0)

Table 2: Plan score (max value of 1.0) using the formula employed by the international planning competition (IPC): for each instance i, if the best-known makespan for i is P_i, then for a given solver X that returns a plan p^i_X: Score$(i, X) = P_i/C_{max}(p^i_X)$. For a given 50-instance benchmark set, the score for each solver is the average over the instance scores for which a solution was found by the solver. † indicates the instances were solved to proven optimality. * indicates some solutions found were deemed infeasible (and thus not included). $|N| = 8$ problems are run for two minutes, and $|N| = 21$ problems for 10 minutes.

	LPG	TFD	SGPlan	POPF
$\|N\|$=8, $\mathcal{P}1$, QCC	1.8% (13)	7.1% (30)	31.0% (50)	10.1% (42)
$\|N\|$=8, $\mathcal{P}1$, QCC-I	5.9% (28)	22.8% (2)	N/A	6.9% (26)
$\|N\|$=8, $\mathcal{P}1$, QCC-X	26.7% (50)	6.2% (33)	35.3% (50)	19.6% (49)
$\|N\|$=8, $\mathcal{P}2$, QCC	9.1% (41)	5.5% (26)	18.0% (50)	9.6% (45)
$\|N\|$=8, $\mathcal{P}2$, QCC-I	12.2% (44)	N/A	N/A	10.3% (47)
$\|N\|$=8, $\mathcal{P}2$, QCC-X	35.7% (49)	8.4% (44)	N/A	26.2% (34)
$\|N\|$=21, $\mathcal{P}1$, QCC	4.4% (31)	18.3% (42)	11.4% (44)	6.2% (39)
$\|N\|$=21, $\mathcal{P}1$, QCC-I	2.7% (20)	N/A	N/A	3.8% (20)
$\|N\|$=21, $\mathcal{P}1$, QCC-X	40.9% (47)	22.6% (36)	32.2% (50)	33.5% (19)

Table 3: Average makespan improvement and number of instances that a positive improvement was observed (in brackets) with the hybrid approach for each planner.

QCC improvement is small, likely due to already high quality planning solutions, and (2) for the most complex set $|N|$=21, $\mathcal{P}1$, QCC-X the improvement is consistently large. In general, it seems the crosstalk constraints benefit the CP approach while inhibiting temporal planning. Among all solved problems, the number of problems that saw improvements are consistently high across different planners and problem sets.

7 Conclusions & Future Work

In this paper, we describe our recent investigation in using CP as an alternative to temporal planning to solve QCC problems. Our empirical evaluation shows that while CP as a stand-alone approach does not perform well when compared against the current state-of-the-art temporal planners, a hybrid of planning and CP, where CP is warm-started with a solution found by temporal planning, out-performs both planning and CP alone. Additionally, the newly introduced variations QCC-I and QCC-X lead to a more diverse QCC-based test benchmark for which different planning technologies performed vastly different across different problem sets.

Given these results, our work strengthens the message that AI planning is indeed a very suitable technology to address QCC challenges and clearly demonstrates the benefit of integrating it with OR methods for building the first effective compilers of real-world quantum hardware.

Our proposed hybridization is only an initial investigation of how to best combine temporal planning and CP for QCC problems. In a future work similar to (Beck, Feng, and Watson 2011) we will analyze in-depth the performance of the two components of our hybrid in relation to the problem classes in order to determine an adaptive schedule for switching between temporal planning and CP.

References

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2012. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media.

Barends, R.; Shabani, A.; Lamata, L.; Kelly, J.; Mezzacapo, A.; Heras, U. L.; Babbush, R.; Fowler, A. G.; Campbell, B.; Chen, Y.; et al. 2016. Digitized adiabatic quantum computing with a superconducting circuit. *Nature* 534(7606):222–226.

Beals, R.; Brierley, S.; Gray, O.; Harrow, A. W.; Kutin, S.; Linden, N.; Shepherd, D.; and Stather, M. 2013. Efficient distributed quantum computing. *Proceedings of the Royal Society A.* 469(2153):767 – 790.

Beck, J. C.; Feng, T.; and Watson, J.-P. 2011. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing* 23(1):1–14.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the* 22^{nd} *International Conference on Artificial Intelligence Planning and Scheduling (ICAPS)*.

Boxio, S. 2016. Characterizing quantum supremacy in near-term devices. In *arXiv preprint arXiv:1608.0026*.

Bremner, M. J.; Montanaro, A.; and Shepherd., D. J. 2016. Achieving quantum supremacy with sparse and noisy commuting quantum computations. In *arXiv preprint arXiv:1610.01808*.

Brierly, S. 2015. Efficient implementation of quantum circuits with limited qubit interactions.". In *arXiv preprint arXiv:1507.04263*.

Chen, Y., and Wah, B. 2006. Temporal planning using subgoal partitioning and resolution in sg-plan. *Journal of Artificial Intelligence Research* 26:323 – 369.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Devitt, S. J. 2016. Performing quantum computing experiments in the cloud. *Physical Review A* 94(3):222–226.

Do, M. B., and Kambhampati, S. 2000. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. In *Proceedings of The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS)*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the* 19^{th} *International Conference on Automated Planning and Scheduling*, 318 – 325.

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

Gerevini, A.; Saetti, L.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239 – 290.

Guerreschi, G. G., and Park, J. 2016. Gate scheduling for quantum algorithms. *arXiv preprint arXiv:1612.08208*.

Hadfield, S.; Wang, Z.; O'Gorman, B.; Rieffel, E. G.; Venturelli, D.; and Biswas, R. 2017. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. *ArXiv e-prints*.

IBM. 2017. The ibm quantum experience. `http://www.research.ibm.com/quantum/`. 2017-02-19.

Kautz, H., and Selman, B. 1999. Blackbox: Unified sat-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*.

Kramer, L. A.; Barbulescu, L.; and Smith, S. F. 2007. Understanding performance tradeoffs in algorithms for solving oversubscribed scheduling. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 1019. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Laborie, P. 2009. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 148–162. Springer.

Neill, C.; Roushan, P.; Kechedzhi, K.; Boixo, S.; Isakov, S.; Smelyanskiy, V.; Barends, R.; Burkett, B.; Chen, Y.; Chen, Z.; et al. 2017. A blueprint for demonstrating quantum supremacy with superconducting qubits. *arXiv preprint arXiv:1709.06678*.

Piacentini, C.; Castro, M.; Cire, A.; and Beck, J. C. 2018. Linear and integer programming-based heuristics for cost-optimal numeric planning. In *AAAI 2018*.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2015. Heuristics for cost-optimal classical planning based on linear programming. In *IJCAI*, 4303–4309.

Ramasesh, V.; O'Brien, K.; Dove, A.; Kreikebaum, J. M.; Colless, J.; and Siddiqi, I. 2017. Design and characterization of a multi-qubit circuit for quantum simulations. In *March Meeting 2017*. American Physical Society.

Reagor, M.; Osborn, C. B.; Tezak, N.; Staley, A.; Prawiroatmodjo, G.; Scheer, M.; Alidoust, N.; Sete, E. A.; Didier, N.; da Silva, M. P.; Acala, E.; Angeles, J.; Bestwick, A.; Block, M.; Bloom, B.; Bradley, A.; Bui, C.; Caldwell, S.; Capelluto, L.; Chilcott, R.; Cordova, J.; Crossman, G.; Curtis, M.; Deshpande, S.; El Bouayadi, T.; Girshovich, D.; Hong, S.; Hudson, A.; Karalekas, P.; Kuang, K.; Lenihan, M.; Manenti, R.; Manning, T.; Marshall, J.; Mohan, Y.; O'Brien, W.; Otterbach, J.; Papageorge, A.; Paquette, J.-P.; Pelstring, M.; Polloreno, A.; Rawat, V.; Ryan, C. A.; Renzas, R.; Rubin, N.; Russell, D.; Rust, M.; Scarabelli, D.; Selvanayagam, M.; Sinclair, R.; Smith, R.; Suska, M.; To, T.-W.; Vahidpour, M.; Vodrahalli, N.; Whyland, T.; Yadav, K.; Zeng, W.; and Rigetti, C. T. 2017. Demonstration of Universal Parametric Entangling Gates on a Multi-Qubit Lattice. *ArXiv e-prints*.

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

Smith, R. S.; Curtis, M. J.; and Zeng, W. J. 2016. A practical quantum instruction set architecture. In *arXiv preprint arXiv:1608.03355*.

Steiger, D. S.; Häner, T.; and Troyer, M. 2016. Projectq: An open source software framework for quantum computing. In *arXiv preprint arXiv:1612.08091*.

van den Briel, M., and Kambhampati, S. 2005. Optiplan: A planner based on integer programming. *Journal of Artificial Intelligence Research* 24:919–931.

van Hoeve, W.-J., and Katriel, I. 2006. Global constraints. *Foundations of Artificial Intelligence* 2:169–208.

Venturelli, D.; Do, M.; Rieffel, E.; and Frank, J. 2017. Temporal planning for compilation of quantum approximate optimization circuits. In *Proceedings of The 26th International Joint Conference on Artificial Intelligence (IJCAI17)*.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence Journal* 170(3):298—335.

Wecker, D., and Svore, K. M. 2014. Liqui $|>$: A software design architecture and domain-specific language for quantum computing. In *arXiv preprint arXiv:1402.4467*.