Temporal Planning for Compilation of Quantum Approximate Optimization Circuits

Davide Venturelli[1,3], Minh Do[2,4], Eleanor Rieffel[1], and Jeremy Frank[2]
[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

We investigate the application of temporal planners to the problem of compiling quantum circuits to emerging quantum hardware. While our approach is general, we focus our initial experiments on Quantum Approximate Optimization Algorithm (QAOA) circuits that have few ordering constraints and thus allow highly parallel plans. We report on experiments using several temporal planners to compile circuits of various sizes to a realistic hardware architecture. This early empirical evaluation suggests that temporal planning is a viable approach to quantum circuit compilation.

1 Introduction

We explore the use of temporal planners to optimize compilation of quantum circuits to newly emerging quantum hardware. Previously, only special purpose quantum hardware was available, namely, quantum annealers that could run one type of quantum optimization algorithm. The emerging gate-model processors are universal in that, once scaled up, they can run any quantum algorithm. IBM recently provided public access to a 5-qubit gate-model processor through the cloud [IBM, 2017], recently updated to 17 qubits, and scalable gate-model quantum computing architectures are being manufactured by other groups, such as TU Delft [Versluis *et al.*, 2016], UC Berkeley [Ramasesh *et al.*, 2017], Rigetti Computing [Sete *et al.*, 2016], and Google [Boxio, 2016]. All cited groups have announced plans to build gate-model quantum processors with 40 or more qubits in the near term.

Quantum algorithms process information stored in qubits, the basic memory unit of quantum processors, and quantum operations (called *gates*) are the building blocks of quantum algorithms, just as instructions on registers are the building blocks of classical algorithms. Quantum algorithms must be compiled into a set of elementary machine instructions (the gates), which are applied at specific times, in order to run them on quantum computing hardware. For a review of quantum computing, see [Rieffel and Polak, 2011].

Quantum algorithms are often specified as quantum circuits on idealized hardware since physical hardware has varying constraints. For example, the emerging gate-model quantum hardware mentioned above use superconducting qubits in a planar architectures with nearest-neighbor restrictions on the memory locations (qubits) to which the gates can be applied. Idealized circuits generally do not consider those nearest neighbor constraints. For this reason, compiling idealized quantum circuits to specific hardware requires adding supplementary gates that move qubit states to locations where the desired gate can act on them.

Quantum computational hardware suffers from *decoherence*, which degrades the performance of quantum algorithms over time. Thus, it is important to *minimize* the *duration of the circuit* that carries out the quantum computation, so as to minimize the decoherence experienced by the computation. Optimizing the duration of compiled circuits is a challenging problem due to the parallel execution of gates with different durations. Further, for quantum circuits with flexibility in when the gates can be applied, or when some gates can be applied in a different order while still achieving the same computation, the search space for feasible compilations is often very large. That freedom makes it more challenging to find optimal compilations, but also means there is a greater potential win from improved compilation optimization than for less flexible circuits.

While there has been active development of software libraries to synthesize and compile quantum circuits from algorithm specifications [Wecker and Svore, 2014; Smith *et al.*, 2016a; Steiger *et al.*, 2016a; Devitt, 2016; Barends *et al.*, 2016], few approaches have been explored for compiling idealized quantum circuits to realistic quantum hardware [Beals *et al.*, 2013; Brierly, 2015; Bremner *et al.*, 2016], leaving the problem open for innovation. An analogous issue arising when compiling classical programs is the *register allocation* problem, in which program variables are assigned to machine registers to improve execution time; this problem reduces to graph coloring [Fu *et al.*, 2005]. Recent studies explore exact schemes [Wille *et al.*, 2014], approximate tailored methods [Kole *et al.*, 2017] or formulations suited for off-the-shelf Mixed Integer Linear Programming (MILP) solvers such as Gurobi [Bhattacharjee and Chattopadhyay, 2017].

In this paper, we apply temporal planning techniques to the problem of compiling quantum circuits to realistic gate-model quantum hardware. Specifically, we model machine instructions as PDDL2.1 durative actions, enabling domain-independent temporal planners to find a parallel sequence of conflict-free instructions that when executed can achieve

what the high-level quantum algorithm intends to achieve. While our approach is general, we focus our initial experiments on QAOA circuits that have few ordering constraints and thus allow highly parallel plans. We report on experiments using several temporal planners to compile circuits of various sizes to an architecture inspired by those currently being built. This early empirical evaluation suggests that temporal planning is a viable approach to quantum circuit compilation. A more elaborate discussions of the techniques and results obtained can be found in the extended version of this paper [Venturelli *et al.*, 2017].

2 Architecture-Specific Compilation Problem

Quantum circuits for general quantum algorithms are often described in an idealized architecture in which any 2-qubit gate can act on any pair of qubits. In an actual architecture, physical constraints impose restrictions on which pairs of qubits support gate interactions. For superconducting qubit architectures, qubits in a quantum processor can be thought of as nodes in a planar graph, and 2-qubit quantum gates are associated to edges (Fig. 1). Gates that operate on distinct sets of qubits may be able to operate concurrently though there may be additional restrictions, such as requiring the sets to be non-adjacent, as in Google's proposed architecture [Boxio, 2016]). Furthermore, there are different types of quantum gates, each taking different durations, with the duration depending on the specific physical implementation.

In order for the computation specified by the idealized circuit to be carried out, we use a particular type of 2-qubit gate, the *swap* gate, which exchanges the state of two qubits. A sequence of swap gates moves the contents 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. For the purposes of this study, 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, but our approach can handle the more general cases.

Problem definition: Given an idealized circuit used to define a general quantum algorithm, the circuit compilation problem is to find a new architecture-specific circuit that implements the idealized quantum circuit by adding swap gates as required. The objective is to minimize the overall duration to execute all gates in a new circuit.

Compilation example: Fig. 1 shows the hypothetical chip design we will use for our experiments on circuit compilation. It is inspired by the architecture proposed by Rigetti Computing Inc. [Sete *et al.*, 2016]. Qubits are labeled with n_i and the colored edges indicate the types of 2-qubit gates available, in this case swap gates and two other types of 2-qubit gate (further described in Section 4).

To illustrate the challenges of finding effective compilation, we present some concrete examples, with reference to the 8-qubit section in the top left of Fig. 1. Suppose that at the beginning of the compilation, each qubit location n_i is associated to the qubit state q_i. Let us also assume that the

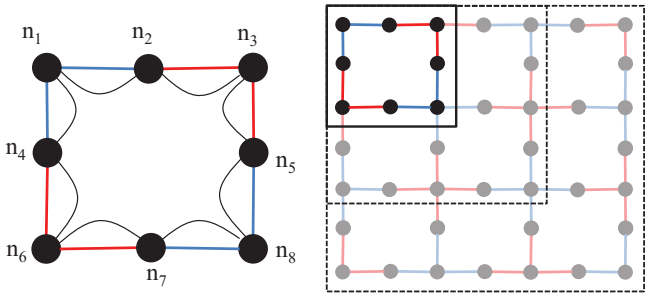


Figure 1: *Left*: A schematic for the hypothetical chip design used in our numerical experiments, with available 2-qubit gates represented by colored arcs in a weighted multigraph. Each color is associated to a specified, distinct gate-type and duration: SWAP gates (black) and two other types of 2-qubits gates (red and blue). The 1-qubit gates are present at each qubit (black dot). *Right*: Dashed boxes indicate the 3 different chip sizes used in our empirical evaluation (see Sec. 5). For visual clarity, only the label locations and the SWAP-gates for the smaller chip size, corresponding to the top-left sector of the largest chip, are shown.

idealized circuit requires the application of a red gate to the states q_2 and q_4, initially located on qubits n_2 and n_4. One way to achieve this task would be to swap the state in n_4 with n_1, while at the same time swapping n_2 with n_3. Another swap, between n_1 and n_2, positions q_4 in n_2 where a red-gate connects it to q_2 (which is now in n_3).

The sequence of gates to achieve the stated goal are:

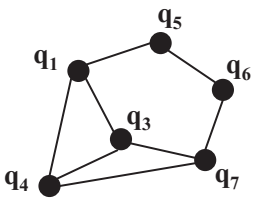
$$\{\text{SWAP}_{n_4,n_1}, \text{SWAP}_{n_2,n_3}\} \rightarrow \text{SWAP}_{n_1,n_2} \rightarrow \text{RED}_{n_2,n_3} \equiv \text{RED}(q_2, q_4) \quad (1)$$

The first line refers to the sequence of gate applications, while the second corresponds to the algorithm objective specification (a task defined over the qubit states). The sequence in Eq. (1) takes $2\tau_{swap} + \tau_{red}$ clock cycles where $\tau_\star$ represents the duration of the $\star$-gate.

As the second example, the idealized circuit requires $\text{BLUE}(q_1, q_2) \wedge \text{RED}(q_4, q_2)$, in no particular order. If $\tau_{blue} > 3 \times \tau_{swap}$, the compiler might want to execute BLUE_{n_1,n_2} while the qubit state q_4 is swapped all the way clockwise in five SWAPs from n_4 to n_3 where RED_{n_2,n_3} can be executed. However, if $\tau_{swap} < 3 \times \tau_{blue}$, it is preferable to wait until the end of BLUE_{n_1,n_2} and then start to execute the instruction sequence in Eq. (1).

3 Compiling QAOA for the MaxCut Problem

While our approach can be used to compile arbitrary quantum circuits to a wide range of architectures, in this paper we concentrate on one particular case: compiling QAOA circuits [Farhi *et al.*, 2014] for MaxCut to the architecture shown in Fig. 1. We choose to work with QAOA circuits because they have many gates that "commute" with each other (i.e., no ordering enforced). Such flexibility means that the compilation search space is larger than for other less flexible circuits. Thus, compared to other less flexible classes of circuits, finding the optimal compilation is more challenging, but there is potential for greater compilation optimization. We selected MaxCut as the target problem



PS1	*MX*	*PS2*
P-S(q_1, q_4)	MIX(q_1)	P-S(q_1, q_4)
P-S(q_1, q_3)	MIX(q_3)	P-S(q_1, q_3)
P-S(q_3, q_4)	MIX(q_4)	P-S(q_3, q_4)
P-S(q_3, q_7)	MIX(q_5)	P-S(q_3, q_7)
P-S(q_4, q_7)	MIX(q_6)	P-S(q_4, q_7)
P-S(q_6, q_7)	MIX(q_7)	P-S(q_6, q_7)
P-S(q_5, q_6)		P-S(q_5, q_6)
P-S(q_1, q_5)		P-S(q_1, q_5)

Figure 2: An example 6-vertex MaxCut problem on a randomly generated graph (qstates q_2 and q_8 are not appearing in this instance) and the p-s and mix gates for $p = 2$.

since it has become one of the de facto benchmark standards for quantum optimization of all types and is considered a primary target for experimentation in the architecture of [Sete *et al.*, 2016].

MaxCut Problem: Given a graph $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. The objective is to partition the graph vertices into two sets such that the number of edges connecting vertices in different sets is maximized.

A quadratic boolean objective function for MaxCut is:

$$U_{MaxCut} = \frac{1}{2} \sum_{(i,j)\in E} (1 - s_i s_j), \qquad (2)$$

where s_i are binary variables, one for each vertex v_i, with values +1 or -1 indicating to which partition the vertex v_i is assigned. From this formulation, an idealized QAOA circuit requires a 2-qubit gate for each quadratic term in Eq. (2), as well as a 1-qubit gate for each vertex [Farhi *et al.*, 2014].

Idealized QAOA circuits for MaxCut alternate between a *phase separation* step (PS) and a *mixing* step. The phase-separation step for QAOA for MaxCut is simpler than for other optimization problems and consists of a set of identical 2-qubit gates that must be applied between certain pairs of qubits, depending on the graph of the MaxCut instance under consideration. We will refer to these as *p-s* gates, and the main goal of the compilation is to plan out those gates. All p-s gates can be carried out in any order (subject to constraints on the chip). In the mixing phase, a set of 1-qubit operations are applied, one to each qubit. All p-s gates that involve a specific qubit q must be carried out before the mixing operator on q can be applied. These two steps are repeated p times. We consider p = 1 and p = 2 in our experiments (detailed in Section 5). Fig. 2 shows a concrete 6-vertex MaxCut example with the set of available p-s and mix gates for $p = 2$.

With reference to Fig. 1, the constraints on the compilation problem are:

- SWAP gates are located at every edge with τ_{swap} = 2.
- there are two kind of non-swap gates: P-S gates are 2-qubit gates and MIX gates are 1-qubit gates.
- P-S gates are located at every edge of the grid, but their duration τ_{p-s} can be 3 or 4 depending on their location (respectively blue or red edges in Fig.1).
- MIX gates are located at every vertex with τ_{mix}=1.
- In the initialization stage, which is not considered as part of the compilation problem, a quantum state is assigned to each qubit.

4 Compilation of a Quantum Circuit as 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 problem of compiling quantum circuits as a temporal planning problem.

Planners: A planner is software that takes as input a specification of domain and problem descriptions and returns a valid plan if one exists. At the abstract level, the planner needs to solve the QAOA compilation problem exemplified in Fig. 2: it identifies the required P-S or MIX gates and builds a conflict-free schedule for all those gates.

Planning Domain Description Language (PDDL) is the de facto standard modeling languages used by many domain-independent planners. We use PDDL 2.1, which allows the modeling of temporal planning formulation in which every action a has duration d_a, starting time s_a, and end time $e_a = s_a + d_a$. Action conditions *cond*(a) are required to be satisfied either (i) instantaneously at s_a or e_a or (ii) required to be true starting at s_a and remain true until e_a. Action effects *eff*(a) may instantaneously occur at either s_a or e_a. Actions can execute 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 quantum circuit compilation problem.

Modeling Quantum Gate Compilation in PDDL 2.1: PDDL is a flexible language that offers multiple alternative ways to model a planning problems. These modeling choices can greatly affect the performance of existing PDDL planners. For instance, many planners pre-process the original domain description before building plans; this is time-consuming, and may produce large 'ground' models depending on how action templates were written. Also, not all planners can handle all PDDL language features effectively (or even at all). We have iterated through different modeling choices with the objective of constructing a PDDL model that: (i) contains a small number of objects and predicates for compact model size; (ii) uses action templates with few parameters to reduce preprocessing effort; while (iii) ensuring that the model can be handled by a wide range of existing PDDL temporal planners.

At the high-level, we need to model: (i) conceptually how actions representing P-S, SWAP, and MIX gates affect qubits and qubit states (qstate); (ii) the actual qubits and qstates involved with a particular compilation problem, their initial locations and final goal requirements, (iii) the underlying qubit-connecting graph structure. We follow

```
(:constants q1 q2 q3 q4 q5 q6 q7 q8 - qstate)

(:durative-action mix_q1_at_1
    :parameters ( )
    :duration (= ?duration 1)
    :condition (and (at start (located_at_1 q1))
                    (at start (GOAL_PS1 q1 q5))
                    (at start (GOAL_PS1 q1 q7))
                    (over all (not (mixed q1))))
    :effect (and (at start (not (located_at_1 q1)))
                 (at end (located_at_1 q1))
                 (at end (mixed q1))))
```

Figure 3: PDDL model of an example MIX gate.

the conventional practice of modeling (i) in the *domain description* while (ii) is captured in the *problem description*. One common practice is to model (iii) within the problem file. However, given that we target a rather sparse underlying qubit-connecting graph structure (see Fig. 1), we decide to capture it within the domain file to ease the burden of the potentially time-consuming step of "grounding" and pre-processing step for existing planners. Specifically:

Objects: We need to model three types of object: qubits, qstates, and the location of the P-S and SWAP gates (i.e., edges connecting different qubits). Since qstates are associated to specific qubits (by means of the predicate $located\_at$, see Fig. 3 for concrete example), they have been modeled explicitly as planning objects, while the qubits and the gate locations (i.e., edges) are modeled implicitly. It is clear from the action definitions in Fig. 3 that qubit locations are embedded explicitly within the action declaration. This approach avoids declaring qubits as part of the action parameters, significantly reducing the number of ground actions to be generated. Our empirical evaluation shows that capturing the graph structure explicitly in the domain file speeds up the preprocessing time of all tested planners, sometime as significantly as 40x.

Actions: Temporal planning actions are created to model: (i) 2-qubit SWAP gates, (ii) 2-qubit P-S gates, and (iii) 1-qubit MIX gates. The most complex constraint to model is the conditions to mix a qstate q given the requirement that *all* P-S gates involving q in the previous phase separation step have been executed. We explored several other choices to model this requirement such as: (i) use a metric variable $PScount(q)$ to model how many P-S gates involving q have been achieved at a given moment; or (ii) use ADL quantification and conditional effect constructs supported in PDDL. Ultimately, we decided to explicitly model all P-S gates that need to be achieved as conditions of the $\text{MIX}(q)$ action. This is due to the fact that alternative options require using more expressive features of PDDL2.1 which are not supported by many effective temporal planners[1]. Fig. 3 shows an example of the mix gate modeled in PDDL[2].

Alternative model: Given that non-temporal planners can perform much better than temporal planners on problems of the same size, we have also created the non-temporal version of the domain by discretizing action durations into consecutive "time-steps" t_i, introducing additional predicates $next(t_i, t_{i+1})$ enforcing a link between consecutive time-steps. However, initial evaluation of this approach with the M/Mp SAT-based planner [Rintanen, 2012] (which optimize parallel planning steps) indicated that the performance of non-temporal planners on this discretized (larger) model is much worse than the performance of existing temporal planners on the original model.

5 Empirical Evaluation

We have modeled the QAOA circuit compilation problem as described in the previous sections and tested them using various off-the-shelf PDDL 2.1 Level 4 temporal planners. The results were collected on a RedHat Linux 2.4Ghz machine with 8GB RAM.

Problem generation: three grid sizes with N = 8, 21 and 40 qubits (dashed boxes in Fig. 1) were used. This design in Fig. 1 is representative of devices to come in the next 2 years; a gate-model 8-qubit chip with the grid we used will be available shortly from Rigetti. For each grid size, we generated two problem classes: (i) p = 1 (only one PS-mixing step) and (ii) p = 2 (two PS-mixing steps). To generate the graphs G for which a MaxCut needs to be found, for each grid size, we randomly generate 100 Erdös-Rényi graphs G [Erdös and Rényi, 1960]. Half (50 problems) are generated by choosing N of $N(N-1)/2$ edges over respectively 7, 18, 36 qstates randomly located on the circuit of size 8, 21, and 40 qubits (referred to herafter as 'Utilization' u=90%). The other half are generated by choosing N edges over 8, 21, and 40 qstates, respectively (referred to herafter as 'Utilization' u=100%). In total, we report tests on 600 random planning problems with size ranging from 1024 - 232,000 ground actions and 192 - 8,080 predicates.

Planner setup: Since larger N and/or p lead to more complex settings with more predicates, ground actions, and thus require planners to find longer plans, the allocated cutoff time for different setting are as follow: (i) 10 minutes for $N = 8$, (ii) 30 minutes for $P = 1, N = 21$; (iii) 60 minutes for other cases. We select planners that performed well in the temporal planning track of previous IPCs, while at the same time representing a diverse set of planning technologies: (i) *LPG*: which is based on local search with restarts over action graphs [Gerevini *et al.*, 2003]; (ii) *Temporal FastDownward (TFD):* a heuristic forward state-space (FSS)

[1]Only one of six planners in the Temporal track of the latest IPC (2014) supports numeric variables and also only one of six supports quantified conditions. Preliminary tests with our PDDL model using metric variables using several metric-temporal planners shows that they perform much worse than on non-metric version.

[2]The full set of PDDL model for all our tested problems is available at: `https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_data.zip`.

	P1						P2			
	N8		N21		N40		N8		N21	
Util	0.9	1.0	0.9	1.0	0.9	1.0	0.9	1.0	0.9	1.0
SGPlan	50	50	50	50	50	50	50	50	-	-
TFD	50	50	50	50	-	-	50	50	50	50
LPG	50	50	50	50	10	14	50	50	-	6

Table 1: Summary of the solving capability of selected planners. Numbers indicate how many random problems out of 50 have been solved.

	p=1, N8		p=1, N21		p=2, N8	
Utilization	0.9	1.0	0.9	1.0	0.9	1.0
SGPlan	0.74	0.76	0.68	0.68	0.76	0.80
TFD	**0.96**	**0.98**	**0.96**	**0.95**	**1.0**	**0.99**
LPG	0.82	0.83	0.83	0.81	0.53	0.51

Table 2: Plan quality comparison between different planners using IPC formula (higher value indicates better plan quality).

search planner with post-processing to reduce makespan [Eyerich *et al.*, 2009]; and (iii) *SPGlan:* partition the planning problem into subproblems that can be solved separately, while resolving the inconsistencies between partial plans using extended saddle-point condition [Wah and Chen, 2004; Chen and Wah, 2006].

We ran SGPlan (Ver 5.22) and TFD (Ver IPC2014) with their default parameters while for LPG (Ver TD 1.0) we ran all three available options: *-speeed*, *-quality*, and *find n plans* (with $n = 10$). Since LPG ($n = 10$) option always dominates both LPG-quality and LPG-speed by solving more problems with better overall quality for all setting, we will exclude results for LPG-quality and LPG-speed from our evaluation discussion. For the rest of this section, LPG result is represented by LPG ($n = 10$).

Evaluation Result Summary: Table 1 shows the overall performance on the ability to find a plan of different planners. SGPlan stops after finding one valid plan while TFD and LPG exhaust the allocated time limit and try to find gradually improving quality plans. Since no planner was able to find a single solution for N = 40 and p = 2, we omit the result for this case from Table 1. Overall, SGPlan and TFD were able to solve the highest number of problems, followed by LPG. SGPlan can find a solution very quickly, compared to the time it takes other two planners to find the first solution. It is the only planner that can scale up to $N = 40$ for p = 1 (finding plans with 150-220 actions). Unfortunately, SGPlan stopped with an internal error for N = 21 and p = 2. TFD generally spent a lot of time on preprocessing for $p = 1, N = 21$ (around 15 minutes) and $p = 2, N = 21$ (around 30 minutes) but when it's done with the pre-processing phase it can find a solution very quickly and also can improve the solution quality very quickly. TFD spent all of the 60 minutes time limit on preprocessing for $N = 40$ problems. LPG can generally find the first solution quicker than TFD (still much slower than SGPlan) but does not improve the solution quality as quickly as TFD over the allocated timelimit.

Plan quality comparison: we use the formula employed by the IPCs to grade planners in the temporal planning track

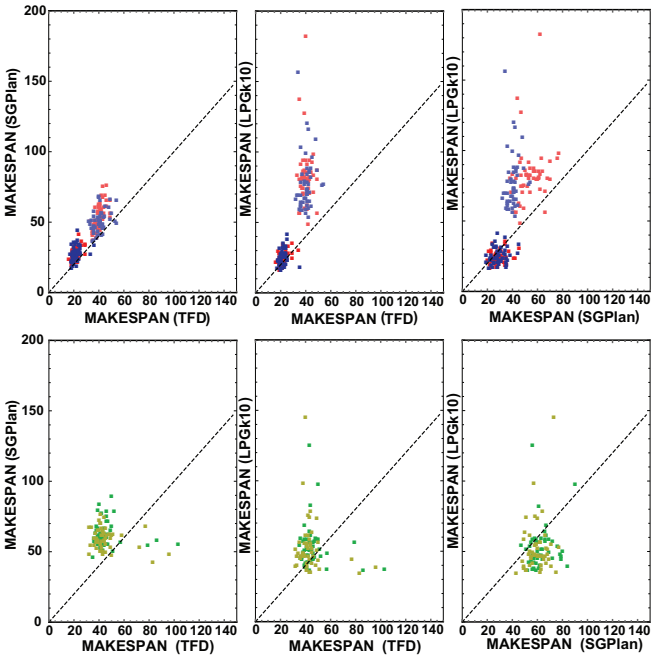


Figure 4: Instance-by-instance comparison of *SGPlan*, *TFD* and *LPG*. Top panel shows results for N=8: red dots indicate instances with u=90% while blue dots are for u=100%. Lower makespan data points refer to p=1 while higher makespans refer to p=2 (see Table 1). Bottom panel shows results for $N = 21$: Green indicates u=90% and yellow u=100%.

since IPC6 [Helmert *et al.*, 2008]: for each planning instance i, if the best-known makespan is produced by a plan P_i, then for a given planner X that returns a plan P_X^i for i, the score of P_X^i is calculated as: $makespan(P_i)$ divided by $makespan(P_X^i)$. A comparative value closer to 1.0 indicates that planner X produces better quality plan for instance i. We use this formula and average the score for our three tested planners over the instance ensembles that are completely solved by the time cutoff. Table 2 compares different planners with regard to plan quality. For N = 8 and p = 1, TFD found the best or close to the best quality plans. LPG is about 15% worse while SGPlan, which unlike TFD and LPG only find a single solution, produces lower quality plans. The comparison results for $N = 21$ and $p = 1$ is similar. For N = 8 and p = 2, TFD again nearly always produces the best quality plan. However, for this more complex case, SGPlan produces overall better quality plans compared to LPG, even though LPG returns multiple plans for each instance.

Fig. 4 shows in further detail the head-to-head makespan comparison between different pairs of planners, specifically pairwise comparisons between TFD, SGPLan, and LPG: TFD always dominates SGPlan, TFD dominates LPG majority of the times, and SGPlan dominates LPG on bigger problems, but is slightly worse on smaller problems.

Planning time comparison: Both TFD and LPG use "anytime" search algorithms and use all of their allocated time to try finding gradually better quality plans. In contrast, SGPlan returns a single solution and thus generally take a very short

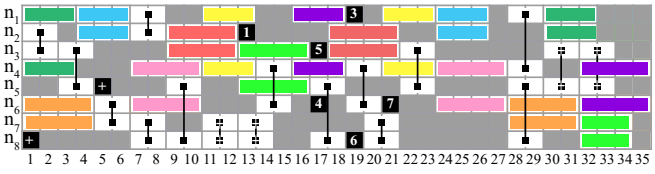


Figure 5: Compilation of $p = 2$ QAOA performed by TFD for the MaxCut problem depicted in Fig. 2 on the $N = 8$ grid in Fig. 1; with time-step on the x-axis and qubit locations on the y-axis. Each row indicates what gate operates on each qubit at a given time-step during the plan. Colored blocks represents p-s gates (of duration 3 or 4 depending on whether they are RED or BLUE). White blocks are swap gates. Gates synchronized in pair, since they involve 2-qubit. Black blocks with numbers are mix gates acting on the corresponding state. Gates marked with a $+$ indicate superfluous gates that could be detected and eliminated in post-processing.

amount of time with the median solving time for SGPlan in p=1$|N_8$, p=1$|N_{21}$, P=1$|N_{40}$ and P=2$|N_8$ are 0.02, 1, 25, and 0.05 seconds.

Other planners: we also conducted tests on: *VHPOP*, *HSP**, *CPT*, and POPF. While LPG, SGPlan, and TFD were selected for their ability to solve large planning problems, we hoped that HSP*, CPT, and VHPOP would return optimal plans to provide a baseline for plan quality estimation. Unfortunately, HSP*, CPT, VHPOP (and also POPF) failed to find a single plan even for our smallest problems.

Discussion: Our preliminary empirical evaluation shows that the test planners provide a range of tradeoffs between scalability and plan quality. At one end, SGPlan can scale up to large problems and solve them in a short amount of time, providing reasonably good quality plans (compared to the best known solutions). At the other end, TFD utilizes all of the allocated time to find the best quality solutions but, in general, is the slowest by far to obtain valid solution. LPG balances between the two: it can either find one solution quickly like SGPlan or can utilize the whole cutoff time to find better quality solutions. Since planning is exponentially hard with regard to problem size, being able to partition it into subproblems of smaller sizes definitely helps SGPlan find a valid solution quickly. However, there are several reasons that TFD and LPG can find overall better quality solutions: (i) their anytime algorithms allow them to gradually find better quality plans, using the previously found plans as a baseline for pruning unpromising search directions; (ii) SGP's partitioning algorithm is based on logical relationship between state variables and actions and ignores all temporal aspects. Thus, combining plans for sub-problems using logical global constraints can lead to plans of lower quality for time-sensitive objective function such as minimizing the plan makespan.

What's missing from our analysis is the assessment on how good the quality of the best plans found compared to optimal solutions. At the moment, there is no published work on finding optimal solution for this problem and, as outlined above, our current effort to get existing optimal-makespan planners to find solutions has not been successful. This is one important future research direction. Based on an "eye-test" and manual analysis, the best plans returned are usually of good quality but not without defects. Fig. 5 shows a visualization, in a 'Gantt chart' format, of a plan found by TFD for the problem instance depicted in Fig. 2. In this plan, qstate q_1 initially located at n_1, is undergoing the following sequence of actions:

$$\begin{aligned}\text{P-S}_3(q_1,q_4) \rightarrow \text{P-S}_3(q_1,q_3) \rightarrow \text{P-S}_4(q_1,q_5) \rightarrow \text{MIX}(q_1)\\ \rightarrow \text{WAIT}(4) \rightarrow \text{P-S}_4(q_1,q_5) \rightarrow \text{WAIT}(2)\\ \rightarrow \text{P-S}_3(q_1,q3) \rightarrow \text{WAIT}(3) \rightarrow \text{P-S}_3(q_1,q_4)\end{aligned}$$

This plan has a short makespan, but contains some unnecessary gates. Examples are the repeated swaps at time 11 and 30, and the mixing of the un-utilized logical states q_2 and q_8 at times 1,5. These spurious gates/actions do not affect the makespan, and they can be identified and eliminated by known plan post-processing techniques [Do and Kambhampati, 2003]. We also believe a tighter PDDL model will help eliminate extra gates.

6 Future Work

This work paves the way for potentially impactful future work on the use of artificial intelligence methods for quantum computing. In future work, we plan to further tune the performance of the planners, including choosing an initial assignment of qstates to qubits favorable for compilation. In order to scale reliably to larger QAOA circuit sizes, we will develop decomposition approaches in which $p > 1$ could be divided into multiple $p = 1$ problems to be solved independently and matched in a post-processing phase. We will also compare with other approaches to this compilation problem such as sorting networks [Beals *et al.*, 2013; Brierly, 2015; Bremner *et al.*, 2016], and we will look at parameters values for the durations that match existing hardware, in collaboration with experimental groups. A virtue of the planning approach is that the framework is very flexible with respect to features of the hardware graph, including irregular structures. In the PDDL modeling, we can include additional features that are characteristic of quantum computer architectures, such as the crosstalk effects of 2-qubit gates and the ability to *quantum teleport* quantum states across the chip [Copsey *et al.*, 2003]. We will also consider other families of quantum circuits and more sophisticated measures against which to optimize the compilation beyond simply the duration of the cirucit. This temporal planning approach should be of great interest to the community developing low-level quantum compilers for generic architectures [Steiger *et al.*, 2016b; Häner *et al.*, 2016] and to designers of machine-instructions languages for quantum computing [Smith *et al.*, 2016b; Bishop, 2017].

Acknowledgements

Authors acknowledge useful discussions with Will Zeng, Robert Smith, and Bryan O'Gorman. The authors appreciate support from the NASA Advanced Exploration Systems program and NASA Ames Research Center (Sponsor Award No. NNX12AK33A).

References

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

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

[Bhattacharjee and Chattopadhyay, 2017] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies. *arXiv preprint arXiv:1703.08540*, 2017.

[Bishop, 2017] Lev S Bishop. Qasm 2.0: A quantum circuit intermediate representation. *Bulletin of the American Physical Society*, 62, 2017.

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

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

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

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

[Copsey *et al.*, 2003] Dean Copsey, Mark Oskin, Francois Impens, Tzvetan Metodiev, Andrew Cross, Frederic T Chong, Isaac L Chuang, and John Kubiatowicz. Toward a scalable, silicon-based quantum computing architecture. *IEEE Journal of selected topics in quantum electronics*, 9(6):1552–1569, 2003.

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

[Do and Kambhampati, 2003] M. B. Do and S. Kambhampati. Improving the temporal flexibility of position constrained metric temporal plans. In *Proceedings of the* 13^{th} *International Conference on Artificial Intelligence Planning and Scheduling (ICAPS)*, 2003.

[Erdös and Rényi, 1960] P. Erdös and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:569–573, 1960.

[Eyerich *et al.*, 2009] P. Eyerich, R. Mattmüller, and G. 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*, pages 318 – 325, 2009.

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

[Fu *et al.*, 2005] C. Fu, K. Wilken, and D. Goodwin. A faster optimal register allocator. *The Journal of Instruction-Level Parallelism*, 7:1 – 31, 2005.

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

[Häner *et al.*, 2016] Thomas Häner, Damian S Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programs. *arXiv preprint arXiv:1604.01401*, 2016.

[Helmert *et al.*, 2008] M. Helmert, M. Do, and I. Refanidis. The 2008 international planning competition: Deterministic track. `http://icaps-conference.org/ipc2008/deterministic/`, 2008. 2008-02-19.

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

[Kole *et al.*, 2017] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. A new heuristic for n-dimensional nearest neighbor realization of a quantum circuit. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[Ramasesh *et al.*, 2017] Vinay Ramasesh, Kevin O'Brien, Allison Dove, John Mark Kreikebaum, James Colless, and Irfan Siddiqi. Design and characterization of a multi-qubit circuit for quantum simulations. In *March Meeting 2017*. American Physical Society, 2017.

[Rieffel and Polak, 2011] E. G. Rieffel and W. H. Polak. *Quantum computing: A gentle introduction*. MIT Press, 2011.

[Rintanen, 2012] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45 – 86, 2012.

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

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

[Smith *et al.*, 2016b] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*, 2016.

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

[Steiger *et al.*, 2016b] Damian S Steiger, Thomas Häner, and Matthias Troyer. Projectq: An open source software framework for quantum computing. *arXiv preprint arXiv:1612.08091*, 2016.

[Venturelli *et al.*, 2017] D. Venturelli, M. Do, E. Rieffel, and J. Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. In *arXiv preprint (`https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_ArXiv.pdf`)*, 2017.

[Versluis *et al.*, 2016] R Versluis, S Poletto, N Khammassi, N Haider, DJ Michalak, A Bruno, K Bertels, and L DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *arXiv preprint arXiv:1612.08208*, 2016.

[Wah and Chen, 2004] B. Wah and Y. Chen. Subgoal partitioning and global search for solving temporal planning problems in mixed space. *International Journal on Artificial Intelligence Tools*, 13(4):767 – 790, 2004.

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

[Wille *et al.*, 2014] Robert Wille, Aaron Lye, and Rolf Drechsler. Exact reordering of circuit lines for nearest neighbor quantum architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1818–1831, 2014.