Efficient Integration of Complementary Solvers for Quantum Circuit Compilation

Sandhya Saisubramanian[1] and **Minh Do**[2,3] and **Jeremy Frank**[2] and **Shlomo Zilberstein**[1]
[1] College of Information and Computer Sciences, University of Massachusetts Amherst
[2] Planning and Scheduling Group, NASA Ames Research Center
[3] KBR Wyle

Abstract

There has been recent success in solving quantum circuit compilation (QCC) using automated reasoning techniques, particularly using a combination of temporal planning and constraint programming (CP). The existing approach uses a fixed and equal time allocation for each solver in the hybrid setup. As new quantum hardware and problems become available, predetermining an optimal time allocation that works well across problems is challenging. Adapting the hybrid approach to newer settings requires *adaptive, intelligent* algorithm switching techniques. We present a metareasoning strategy that monitors and predicts the performance of the current solver to determine when to switch to the next solver. We evaluate our switching strategy for QCC using a combination of temporal planner and CP solver. Empirical results demonstrate the clear benefits of the switching strategy based on metareasoning, compared to the existing fixed strategy.

1 Introduction

Quantum computing is an emerging paradigm with the potential to solve certain problems faster than classical computing. Quantum computers apply quantum operations called quantum gates to qubits, which are the basic memory units of quantum processors. General quantum algorithms are typically specified as quantum circuits on an idealized architecture in which a gate can act on any subset of qubits. In an actual superconducting qubit device, physical constraints impose restrictions on the sets of qubits on which gates can operate. Many current chip designs feature qubits arranged in a grid; a qubit shares gates with only the 4 or 8 'nearest neighbors'. Thus, the idealized circuit must be *compiled* to specific hardware by, among other operations, adding additional gates that move qubit states (qstates) to locations where the desired gate from the original circuit can act on them. We refer to this as the "quantum circuit compilation" (QCC) problem [1]. The compilation must minimize the total circuit execution time due to the short decoherence time of near-term quantum hardware.

Venturelli et al. (2017) proposed solving QCC of Quantum Alternating Operator Ansatz algorithm (QAOA) for MaxCut as a temporal planning problem, which was then solved using off-the-shelf domain-independent planners. Recently, Booth et al. (2018) proposed solving this problem using a combination of a temporal planner and a constraint programming (CP) solver. The authors show that warm-starting a CP solver with the planner's solution produces considerable improvement over using either of them in isolation. The results suggest that the hybrid approach is a promising technique to solve different QCC problems. A key limitation in the current hybrid approach of Booth et al. (2018) is that each solver in the hybrid setting is allocated a fixed and equal solving time that is independent of the solvers, the problem size or complexity, and the total solving time. As the chip design and gate architectures are fast evolving, it is non-trivial to predetermine a fixed strategy that works well across all problem configurations. Adapting the hybrid approach to newer settings requires a more adaptive and intelligent algorithm-switching technique. How to switch intelligently between solvers in the hybrid setting where the subsequent solver is warm-started with solution from the previous solver, such that the makespan value of the final solution is minimized?

We present a meta-level control approach for switching between a temporal planner (say P_1) and CP solver (say P_2) for solving QCC, by monitoring and predicting the performance of the solvers in real-time. Given a total time T for solving the problem, an optimal switching strategy involves identifying the time t_s to switch from P_1 to P_2, such that the final makespan produced by P_2, when *warm-started* with a solution from P_1, is minimized. In practice, it is impossible to accurately predict the performance of P_2 a priori, when warm-started with a solution of a particular quality and given a fixed time for solving the problem. It is also impossible to accurately predetermine how long P_1 will take to produce a solution of a given quality. Therefore, we propose an approximation, *local monitoring*, to switch between the solvers in our hybrid setting by monitoring the performance of P_1 and switching when a threshold in quality is reached (Figure 1). Empirical evaluation of our approach on the QCC of QAOA for MaxCut problems demonstrates the advantages of switching with local monitoring over the fixed equal time allocation strategy, with a plan score improvement of up to 169% on N21 C problems, which are among the most difficult to solve.

[1]More recent work refers to this step of compilation as 'routing' to distinguish it from other compilation steps.

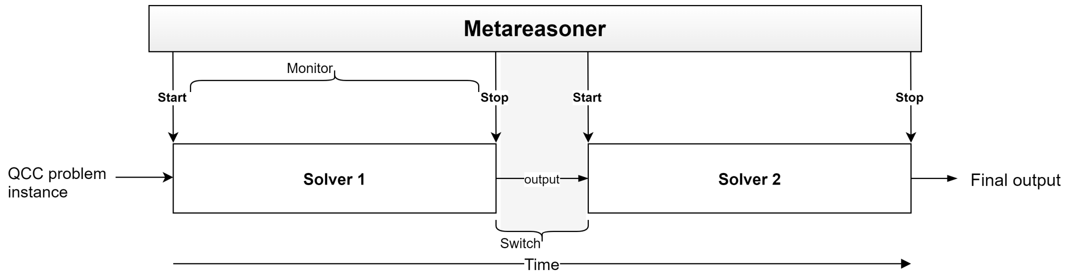


Figure 1: Illustration of local monitoring-based meta-level control for QCC.

The following are our primary contributions:

- Formalizing the problem of real-time intelligent switching between solvers in the hybrid setting for solving QCC of QAOA as a metareasoning problem (Section 3);
- Presenting a local monitoring approach that switches based on the performance of planner P_1 by monitoring and predicting its performance online (Section 4); and
- Evaluating the performance of our approach on QCC of QAOA to solve Maxcut problems (Section 5).

2 Background and Related Work

Quantum Circuit Compilation General quantum algorithms historically have been described in an idealized architecture in which a gate can act on any subset of qubits. However, in an actual superconducting qubit device, such as the latest chips manufactured by IBM, Rigetti Computing, Google and Intel (Corcoles et al. 2019; Murali et al. 2019; Arute et al. 2019), physical constraints impose restrictions on the sets of qubits on which gates can be performed. Recently, a number of approaches have been explored for compiling idealized quantum circuits to realistic quantum hardware utilizing various swap gate insertion strategies in noisy intermediate-scale quantum (NISQ) devices, targeting algorithms that can be executed in the near-term (Cowtan et al. 2019; Zulehner, Paler, and Wille 2018b; Li, Ding, and Xie 2019; Rasconi and Oddi 2019; Oddi and Rasconi 2018; Gokhale et al. 2019; Itoko et al. 2019). We refer to this problem as "quantum circuit compilation" (QCC).

Early efforts in QCC focused on developing heuristic compilation algorithms for nuclear magnetic resonance architectures with the objective of minimizing circuit duration (Maslov, Falconer, and Mosca 2008). Advances in quantum technology coupled with improvements in fault-tolerance have led to the design of larger devices and the development of more sophisticated compilation approaches (Amy et al. 2013; Shafaei, Saeedi, and Pedram 2013; Wille, Lye, and Drechsler 2014; Lye, Wille, and Drechsler 2015).

Venturelli et al. (2017) proposed the first PDDL formulation for solving QCC of Quantum Alternating operator Ansatz algorithm (QAOA) for the MaxCut problem. The authors modeled QCC as a temporal planning problem, which was solved using off-the-shelf domain-independent planners. As a follow-up to this work, Booth et al. (2018) proposed solving QCC using a combination of a temporal planner and a constraint programming (CP) solver. Warm-starting a CP solver with a temporal planner's solution yielded considerable improvement over using either of the solvers in isolation. While we focus on the compilation of QAOA for Max-Cut in superconducting architectures, other architecture-agnostic methodologies are also being investigated and these typically require defining the desired circuit a priori (Zulehner, Paler, and Wille 2018a).

Recent works (Oddi and Rasconi 2018; Rasconi and Oddi 2019) showed further performance improvement on the same benchmark set by utilizing domain-specific heuristics within the greedy randomized, and later, genetic algorithm. There are also efforts to solve QCC outside the traditional AI community (Maslov, Falconer, and Mosca 2008; Amy et al. 2013; Shafaei, Saeedi, and Pedram 2013; Wille, Lye, and Drechsler 2014; Lye, Wille, and Drechsler 2015; Zulehner, Paler, and Wille 2018a).

Meta-level control Meta-level control of algorithms or metareasoning is the process of reasoning about the performance of the underlying algorithm. Metareasoning for time-critical decisions offers a principled approach to balance the trade-off between computation time and solution quality. Existing metareasoning approaches typically target settings with one anytime planner and the goal is to decide whether to act on the current solution of the anytime algorithm or to wait until the solution improves (Cserna, Ruml, and Frank 2017; Hansen and Zilberstein 2001; Hay and Russell 2011; Horvitz and Rutledge 1991; Russell and Wefald 1991; Svegliato, Wray, and Zilberstein 2018; Zilberstein 2008). In this work, the metareasoner decides whether to continue with the current solver or warm-start the next solver with the current solution.

Another related line of work utilizes metareasoning to identify the most-suitable algorithm for a problem from a portfolio of algorithms (Carchrae and Beck 2005; Lieder et al. 2014). In contrast, we consider a portfolio in which successive solvers acting on the problem are warm-started with the solution produced by the former. The key challenge is to identify when to switch from one solver to the next, by reasoning about the quality of solution of the current solver used to warm-start the subsequent solver.

3 Meta-Level Control of Solvers for QCC

We address the problem of real-time intelligent switching between a pipeline of algorithms in a portfolio, specifically for solving the QCC problem, using meta-level control. The system consists of two components: a *solver* module that solves the problem and a *metareasoner* module that reasons about the solver module. The focus is on building a metareasoner to decide when to switch between a temporal planner and a CP solver, for QCC.

Solver Module: Building on the existing work of Booth et al. (2018), we consider the solver module as a hybrid with two solvers, $\{P_1, P_2\}$, where P_1 is a temporal planner and P_2 is a CP solver, to act on the QCC problem. P_1 and P_2 share a model of the QCC problem. We use the QCC model described in Booth et al. (2018). The solvers have a fixed ordering and act sequentially on the problem until the total allotted solving time (T) elapses. That is, P_1 first operates on the problem and when P_1 is terminated, P_2 is warm-started with a solution found by P_1, and runs until T is reached.

Metareasoning Module: The metareasoner M monitors the solver to identify the time instant t_s, $0 \leq t_s \leq T$, to switch from P_1 to P_2 such that the makespan m of the temporal plan returned by P_2 at T is minimized (Figure 1). Let $m_\pi^{P,t}$ denote the plan makespan produced by a solver P when warm-started with a plan π and time t allotted for solving the problem. We drop the subscript π when the solver is not warm-started with a solution. The best makespan under this setting is:

$$m^* = \min_t m_\pi^{P_2, T-t}$$

where π is the 'best' solution produced by P_1 when executed for time t. Then the optimal switching time is

$$t_s^* = \arg\min_{0 \leq t \leq T} m_\pi^{P_2, T-t}$$

To accurately estimate the optimal switching time for a given novel planning problem, the metareasoner is required to have full knowledge of:

1. all the time points $t_i \leq T$ at which P_1 produces a new solution π_i; and
2. final solution quality (i.e., makespan value) $m_{\pi_i}^{P_2, T-t_i}$ returned by P_2 when warm-started with π_i and with a time limit $T - t_i$.

However, it is impossible to accurately estimate these values a priori due to lack of information about the differences in the way in which the search space is explored by each solver, especially when it is an anytime algorithm and in the setting where P_2 is warm-started with a solution. Therefore, in the rest of the paper, we will describe an algorithm to approximate t_s with the following assumptions.

Assumption 1: Since we are minimizing plan makespan, the makespan value of the solution produced by P_1 is a reasonable metric for determining the 'best' solution for warm-starting P_2.

At switching time t_s, the plan π_1 with the least makespan among all solutions found by P_1 until t_s is the best plan for warm-starting P_2.

Assumption 2: The rate of solution quality improvement in P_1 is an accurate indicator for when P_1 produces the best solution to warm-start P_2.

Thus, t_s is determined by monitoring only the performance of P_1 to identify if it has produced the 'best' solution for warm-starting P_2.

4 Local Monitoring with Online Prediction

Since the metareasoner's switching decisions are solely based on its monitoring of the performance of P_1, without considering P_2, we call this a *local monitoring* approach. For practical purposes, the metareasoner monitors the performance of P_1 at discrete time intervals, every θ time units. The decision to switch or continue is made every $\Delta t = \eta \cdot \theta$ time units, where η is a constant, until the switch to P_2 occurs or until timeout. Since we consider a setting in which P_2 is warm-started with P_1's solution, the metareasoner does not switch until P_1 generates at least one solution.

Local monitoring with perfect information If the metareasoner has knowledge of the best (last) solution $\pi_{P_1}^*$ produced by P_1, within T, then it can switch as soon as P_1 has produced a plan π with this makespan value $m^{P_1,T}$. This is a case of perfect information as $m^{P_1,T}$ is known to the metareasoner a priori. Since the agent switches after finding the **b**est **s**olution from P_1 with **p**erfect **i**nformation, we refer to this approach as BSPI.

Switching when the final makespan $m^{P_1,T}$ is found may not be the optimal switching time since it may result in very little or no time for P_2 to act on the problem. However it is a reasonable proxy to guide the switching decision, based on Assumption 1. A similar idealized strategy is described in (Booth et al. 2018). In practice, $m^{P_1,T}$ is unknown a priori and it is also challenging to accurately estimate this value. Therefore, we describe below an online approach to estimate the makespan improvement on-the-fly.

Online Performance Prediction with Myopic Lookahead: A popular technique in the existing metareasoning literature to estimate the improvement in solution quality is to utilize a *performance profile* (Hansen and Zilberstein 2001) of the planner. A performance profile is used to predict the expected improvement in solution quality, typically as a function of the current solution and computation time. Generating an accurate performance profile involves significant preprocessing, which makes it unsuitable for novel settings in which large amounts of preprocessing data are unavailable. We avoid the need for preprocessing by predicting the makespan improvement of P_1 online, based on its current performance and using a myopic lookahead strategy.

Specifically, the metareasoner monitors the performance of P_1 every θ time units (say 1 second), and collects the makespan value produced, if a solution is found. The makespan values generated by P_1 from start till time t, $[m^{P_1,0}, \ldots, m^{P_1,t}]$, are used to predict the makespan value of the planner at $t + \Delta t$. In this paper, we initialize $m^{P_1,0}$

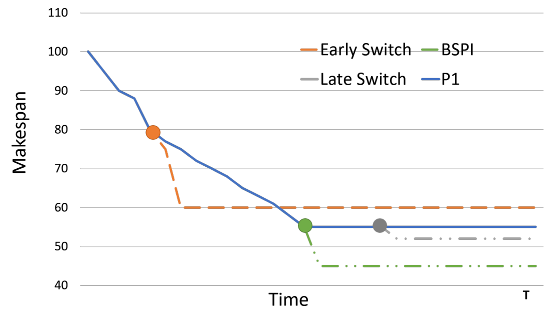


Figure 2: An *idealized* illustration of different switching decisions based on local monitoring—switching after P_1 has produced the best makespan possible, within T, reduces the final makespan.

to some large constant K and use linear regression for future makespan prediction. In practice, other prediction approaches may be used. Naturally, the prediction accuracy improves as the metareasoner collects more data. The online performance prediction does not involve preprocessing and hence is compatible for use with any novel instance.

Cost-Sensitive Switching When using online prediction with myopic lookahead, it is important to reason about whether the projected reduction in makespan, if any, is significant enough to postpone switching, by taking into account the time spent by the solver to achieve that improvement. Intuitively, a solution with a particular makespan computed in a minute has higher utility than a solution with the same or slightly better makespan computed in an hour.

Figure 2 illustrates an ideal setting where switching with the best makespan of P_1 reduces the final makespan. In this figure, P1 indicates the performance of the planner P_1 when executed until time T with no switching. The various switching decisions and their corresponding performances are indicated by different colors. BSPI, indicated by the green line, switches as soon as the makespan matches that of the best (or last) solution that can be produced by P_1 within T. Premature switching (earlier than BSPI switching) with a makespan value that is higher than the best value of P_1 may offer more computational time to P_2, but often at the price of starting with a significantly inferior solution. In practice, however, we observe that some problems benefit from switching earlier with more time allotted to P_2. In our experiments, the 'First' switch performs better than BSPI in 5/9 problems for the TFD+CP hybrid. Switching later than BSPI, after giving the metareasoner sufficient time to confirm that the current makespan is indeed the best makespan value P_1 can produce within T, also affects the final makespan value produced since switching later allocates too little time to P_2.

Therefore, the value of continuing with the current solver, P_1, is formalized with two components: (1) the expected makespan improvement and (2) the cost of time.

Let r denote the rate of makespan improvement, calculated as $r = \frac{(m^{P_1,t} - m')}{m^{P_1,t}}$ with m' denoting the predicted makespan of P_1 for time $t' = t + \Delta t$, using linear regression or any other approach that facilitates predicting the planner performance. Let C_t denote the cost of time, which represents the computation cost for running the planner P_1 until t. The cost of time also indirectly represents the opportunity cost of not using P_2 and continuing with P_1. Naturally, the benefits of continuing with P_1 diminishes as (1) r decreases and (2) C_t increases. The loss function, which determines the switching decision, is then defined as:

$$L(P_1) = (1 - r) * C_t.$$

Definition 1 *In cost-sensitive switching, the metareasoner switches from P_1 to P_2 when the loss incurred by continuing with P_1 exceeds a predefined threshold τ: $L(P_1) \geq \tau$.*

Note that the metareasoner switches only when the loss of continuing with P_1 exceeds the predefined threshold, and not necessarily because P_2 is guaranteed to improve the performance. In fact, the infeasibility of accurately estimating the performance of P_2 when warm-started with a particular solution is a key motivation for the local monitoring approach. Online performance prediction with myopic lookahead and cost-sensitive switching form the basis of and support the design of our local monitoring approach—**L**ocal **M**onitoring with **M**yopic **L**ookahead (LMML).

5 Empirical Evaluation

The main objective of our experiments is to validate the performance of switching guided by local monitoring with a myopic lookahead and cost-sensitive switching (LMML), compared to the existing approaches.

Setup We evaluate our approach on the same benchmark of QCC of QAOA to solve MaxCut problems used in previous work (Booth et al. 2018). Specifically, we evaluate performance on QCC problems for random MaxCut problems on quantum chips with either 8 or 21 qubits arranged in a grid pattern. For each chip size ($N = 8$, $N = 21$), there are three problem configurations: (1) base (B); (2) base + initialization (I); and (3) base + cross-talk constraints (C). For $N = 8$, we also vary the number of times the circuit is repeatedly executed, denoted by $\mathcal{P} = \{1, 2\}$. Problems with $\mathcal{P} = 2$ have more than twice the number of goals and much longer plans[2]. We report results for the first 30 of the 50 instances in each benchmark. The total allowed time (T) for solving the problem is 120 seconds for $N = 8$ and 600 seconds for $N = 21$ problems (same as in (Booth et al. 2018)). For P_1 we use TFD (Eyerich, Mattmüller, and Röger 2009) and POPF (Coles et al. 2010). For P_2, we use the CP Optimizer with model from Booth et al. (2018). LMML is implemented in Python and the experiments were run on an Ubuntu machine with 16GB of RAM.

Baselines The performance of the local monitoring strategy for switching is compared with four baselines:

[2]Between the two "phases", indicated by $\mathcal{P} = 2$, there are additional goals that need to be achieved (Venturelli et al. 2019).

		TFD					POPF				
Problem Set	$\mathcal{P}$	BSPI	Alone	First	Fixed	LMML	BSPI	Alone	First	Fixed	LMML
N8 B	1	0.961	0.891	**0.978**	0.956	0.972	0.951	0.870	*0.961*	0.946	0.949
N8 I	1	0.976	0.897	0.805	0.968	**0.974**	0.936	0.857	0.928	0.924	*0.932*
N8 C	1	0.972	0.899	0.963	0.956	**0.971**	0.963	0.766	0.961	*0.965*	0.957
N8 B	2	0.967	0.904	0.907	**0.959**	0.946	0.942	0.837	*0.939*	0.932	0.938
N8 I	2	0.969	0.860	0.887	0.963	**0.964**	0.871	0.743	0.855	0.872	*0.875*
N8 C *	2	0.893	0.873	**0.913**	0.879	0.901	0.888	0.634	*0.896*	0.879	0.890
N21 B	1	0.641	0.539	*0.644*	0.631	0.640	0.990	0.903	0.967	0.976	**0.981**
N21 I	1	0.508	0.434	*0.513*	0.442	0.502	0.993	0.933	0.951	**0.979**	0.976
N21 C **	1	0.814	0.428	*0.816*	0.297	0.801	0.974	0.619	**0.969**	0.823	0.949
Summed Score		7.701	6.725	7.424	7.050	**7.671**	9.508	7.162	8.427	8.296	**8.447**

Table 1: Performance comparison with four baselines, including the "perfect-information" BSPI approach. **Bold** values indicate best performance across all approaches that can be implemented in practice (i.e., all except BSPI), while *italics* values indicate best performance among all implementable approaches for the 'runner up' planner. Note: plan scores are averaged only over those instances solved by all approaches. * POPF solved only 26 instances. ** POPF solved only 16 instances.

- P_1 *alone:* P_1 solves the problem until T and does not switch to P_2;
- *First:* switch to P_2 as soon as P_1 produces a solution;
- *Equal time:* each solver is allocated equal time to solve the problem and P_2 is warm-started with the best solution produced by P_1 in time $T/2$; and
- *Best solution from* P_1 *with perfect information (BSPI):* in our implementation, we first run P_1 alone until T and record time t^*, we then run the hybrid setting that switch from P_1 to P_2 at t^*.

Algorithm Parameters The metareasoner monitors the performance of P_1 every $\theta = 1$ second and the decision to switch or to continue is made every $\Delta t = 5$ seconds. Motivated by the work on metareasoning for anytime algorithms (Hansen and Zilberstein 2001), we use an exponential function for cost of time, $C_t = e^{\alpha \cdot t}$, with normalization $\alpha = \frac{1}{30}$ for $N = 8$ and $\alpha = \frac{1}{60}$ for $N = 21$ problems with $\tau = e^{0.33}$ for $N = 8$ and $\tau = e^3$ for $N = 21$. The main purpose of α is to convert seconds to minutes for C_t calculation. The values for τ and α are based on our observation of the average time taken, in minutes, for all the planners to produce at least the first solution across all problem variants (B, I, C) on sample problems from the data set. No extensive pre-processing is involved in computing these values.

The temporal planners typically solve the $N = 8$ problems within the first 30 seconds and no new solution is produced thereafter and hence we use a relatively larger α and a smaller τ. We also tested with higher values of T and lower Δt (increasing the frequency of the option to switch), which did not significantly improve the solution quality. Tuning τ and α for each problem configuration and the planner, using standard hyperparameter tuning methods, improved the performance. However, we test and report results with the same τ and α for all planners and variants for each chip size (N=8, 21). This somewhat-conservative value allows us to compare the performance of each approach uniformly across settings.

Makespan prediction: Every $\Delta t = 5$ seconds, the metareasoner predicts the makespan of P_1 at the next decision epoch, $t + 5$ seconds. The prediction is performed using linear regression over the set of samples collected at every 1 second until t. Therefore, as the number of samples increase, the prediction is expected to be better. We use the Python `sklearn` package for linear regression.

6 Results Analysis

We evaluate the performance of our proposed switching strategy based on its effectiveness and adaptability. The effectiveness is measured using plan score metric. The plan score (max = 1.00) is calculated as: if the best-known makespan for instance i is M_i, then for a given solver X that returns a plan m^i_X, Score(i, X) = $\frac{M_i}{m^i_X}$. This metric offers a principled approach to evaluate the consistency and relative performance of the different techniques and has been used in the International Planning Competition (IPC) to grade planners' performance on the IPC benchmarks. The adaptability of our approach is analyzed empirically using the switching times for different planners and problem configurations, along with the resulting makespan values.

6.1 Effectiveness of local monitoring

Table 1 shows the plan scores of the different switching techniques considered. The objective is to evaluate if and when our switching strategy improves the performance, compared to the the existing approaches, irrespective of the planner used as P_1. In Table 1 we highlight in **bold** the best performance across planners, and in *italics* the best performance achieved for each individual planner. We highlight in bold and italics for a technique that can be implemented in practice for each planner, which excludes BSPI since it is infeasible to implement in practice due to lack of information regarding the best makespan value of P_1 a priori.

LMML is best on 4 of the 9 problem classes, while Fixed is better on 3 classes. LMML is better than other strategies for either planner when considering performance across all problem classes; more so for TFD than POPF. While BSPI has the highest overall score, which indicates that waiting for P_1 to find the best possible plan within T in general is the best strategy for the hybrid planner + CP setting for

Problem	$\mathcal{P}$	Hybrid	Fixed Strategy	First	BSPI	LMML
N8-B	1	TFD+CP	17.63 ± 1.54	17.23 ± 1.76	17.56 ± 1.84	17.33 ± 1.64
		POPF+CP	17.87 ± 2.03	17.57 ± 1.93	17.77 ± 2.03	17.8 ± 2.02
N8-I	1	TFD+CP	14.94 ± 1.93	18.00 ± 3.53	14.8 ± 1.83	14.83 ± 1.84
		POPF+CP	15.80 ± 2.66	15.70 ± 2.58	15.57 ± 2.51	15.63 ± 2.55
N8-C	1	TFD+CP	22.87 ± 1.65	22.73 ± 2.24	22.47 ± 1.48	22.51 ± 1.54
		POPF+CP	22.77 ± 2.22	22.80 ± 2.27	22.77 ± 2.33	22.93 ± 2.61
N8-B	2	TFD+CP	35.47 ± 4.49	37.80 ± 6.04	35.13 ± 4.53	35.93 ± 4.60
		POPF+CP	36.57 ± 5.03	36.30 ± 5.26	36.17 ± 5.11	36.33 ± 5.09
N8-I	2	TFD+CP	30.80 ± 4.80	33.83 ± 6.14	30.56 ± 4.49	30.73 ± 4.56
		POPF+CP	34.06 ± 4.82	34.70 ± 4.55	34.06 ± 4.71	33.93 ± 4.75
N8-C	2	TFD+CP	53.84 ± 12.92	50.94 ± 7.07	52.37 ± 11.26	51.91 ± 11.26
		POPF+CP	52.69 ± 7.40	51.69 ± 7.51	52.19 ± 7.52	51.96 ± 7.11
N21-B	1	TFD+CP	57.87 ± 11.44	57.01 ± 12.21	57.17 ± 11.86	57.24 ± 11.86
		POPF+CP	36.47 ± 4.48	36.80 ± 4.51	35.94 ± 4.43	36.24 ± 4.36
N21-I	1	TFD+CP	64.93 ± 11.69	63.24 ± 11.92	63.93 ± 12.21	64.35 ± 11.64
		POPF+CP	32.07 ± 5.18	33.07 ± 5.12	31.54 ± 4.83	32.24 ± 5.48
N21-C	1	TFD+CP	162.47 ± 38.54	68.44 ± 12.35	68.62 ± 12.31	68.89 ± 12.76
		POPF+CP	71.81 ± 36.87	58.44 ± 8.71	57.54 ± 8.79	58.67 ± 9.37

Table 2: Average makespan, along with standard deviation, of different switching techniques.

solving QCC, plan scores for dynamic switching are comparable to that of BSPI in most problem sets, and better than that of BSPI in some cases (for instance TFD with First outperforms BSPI on $N = 8, B, \mathcal{P} = 1$). This highlights the problem in ignoring performance of P_2, and also the potential that spending more time on P_2 may be beneficial in some instances. Overall, dynamic switching (either First or LMML) is better than Alone or Fixed in all but two problem classes and is comparable to that of BSPI which is infeasible to implement in practice.

6.2 Adaptability of local monitoring

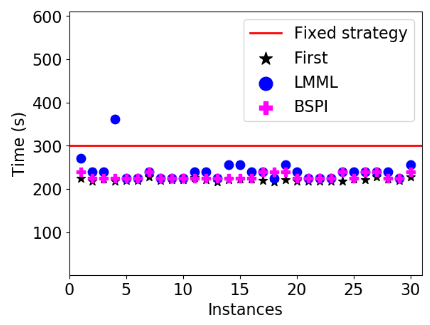
Table 2 shows the average makespan values of the different techniques on all problem variants, along with the standard deviation. Figure 3 plots the switching times for $N = 21, \mathcal{P} = 1$ problems. These results enable us to understand the performance and behavior of local monitoring strategy, beyond the plan score metric.

Each planner explores the search space differently. As a result, the frequency of producing solutions and their quality can be very different for different planners acting on a given problem. In general, we observe that the difficulty of solving the problem increases with N and more constraints. That is, I and C settings are more challenging to solve than B and $N = 21$ is more difficult than solving $N = 8$. This was evident in the frequency and number of solutions produced by P_1. The performances of the First switch and the BSPI strategies in Tables 1, 2 and Figure 3 illustrate this.

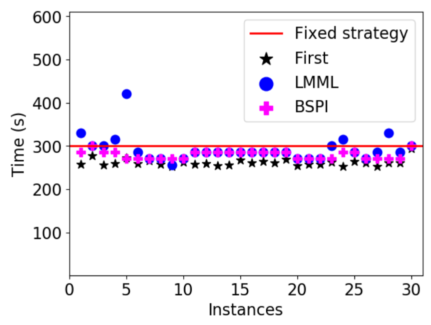
The differences in switching times affect the time allocated to P_2, as well as the quality of the solution used to warm-start P_2, which affects the overall performance. Figure 3 clearly shows that LMML adapts the switching time to each planner's best behavior. TFD switches at close to $T/2$ for all approaches, while POPF switches mostly before $T/2$ except for the 'C' (crosstalk) instances. Similar trends were observed for other instance classes.

Figure 3 indicates that First and BSPI are nearly indistinguishable from each other most of the time, but on occasion BSPI switches a little later than First. Instances where BSPI and First switch at the same time indicate that the planner produces only one solution in the allotted time (favoring First, which can switch immediately, over LMML, which must wait for the cost of time to dominate and prompt switching). In other instances, BSPI switches when the solution with best makespan possible has been found. The switching times of BSPI and LMML are close for TFD+CP; LMML switches a bit later on the 'I' and 'B' instances and a bit earlier on the 'C' instances. For POPF+CP, there is a $\sim$100 second interval between the switching times of BSPI and LMML, in most problems. However, we observe that the overall performances of BSPI and LMML, in terms of makespan, are comparable (Table 2). This happens because POPF does not produce any new solutions in this time interval, or the solutions produced have very similar makespan values.

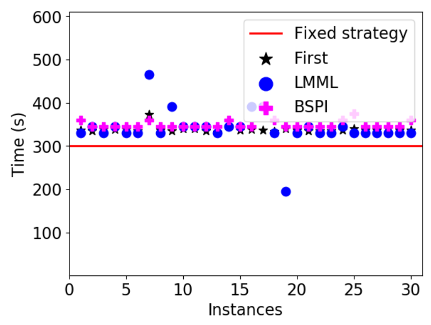
These results show that local monitoring approach adapts the switching time to the performance of P_1 and that predicting the performance of P_1 online works well in practice, but it may not always be the best strategy. For some problems, switching early is better, favoring First, and LMML is able to adapt and switch earlier in those problems (not as early as First, but close). For other problems, switching late is bet-



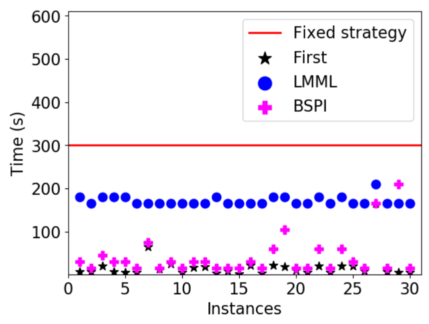
(a) TFD+CP, B



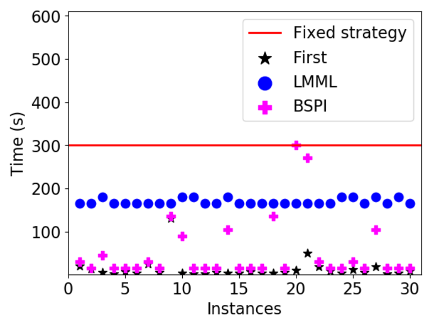
(b) TFD+CP, I



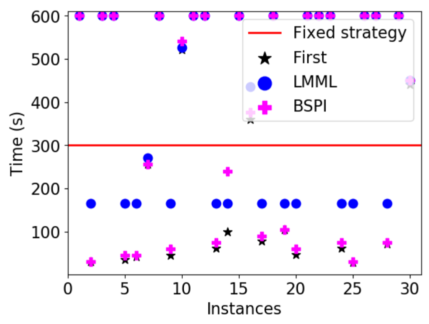
(c) TFD+CP, C



(d) POPF+CP, B



(e) POPF+CP, I



(f) POPF+CP, C

Figure 3: Switching times on $N = 21$, $\mathcal{P} = 1$ problems.

ter and favoring BSPI; again, LMML is also able to adapt and switch late and trail closely the best strategy in those scenarios (not as good as BSPI, but close).

7 Conclusion and Future Work

We present a metareasoning approach for deciding when to switch between solvers for the QCC problem. Our local monitoring approach with limited lookahead, *LMML*, monitors the decrease in makespan and decreasing time remaining to run the second solver, and switches when a loss function parameterized by these values reaches a critical threshold. This approach approximates the idealized strategy in which perfect information is available, BSPI, and can be easily implemented in practice. Our results demonstrate the benefits of switching using metareasoning approaches, compared to the fixed strategy. Tables 1 and 2 demonstrate the effectiveness of dynamic switching over the fixed strategy. Figure 3 shows that LMML adapts the switching time based on the planner performance.

Future Work There are a number of interesting directions for future work. First, we assumed that the makespan value of a solution is a good indicator for deciding when to switch. There could be other features of plans that are better indicators of good plans to warm-start the next solver, which are especially useful when planners produce only one solution in the allotted time. Any future developments in identifying characteristics of a solution that boosts the performance of P_2 can be leveraged by our approach. Second, we focus in this paper on the planner+CP setting for QCC of MaxCut problem but the overall framework does not limit itself to a particular solver or a particular planning domain. Candidate solvers for P_2 that can be warm-started with a complete plan and which can effectively handle other planning domains will allow us to expand our evaluation benchmarks. Third, generalizing our approach to multi-solver portfolio may require developing metareasoning techniques that can effectively monitor multiple solvers, and a more principled approach to determine the parameters for the metareasoner. Finally, designing a metareasoning strategy specifically for stochastic planners such as LPG will help expand the potential candidates for P_1.

Acknowledgments

We thank Davide Venturelli and Eleanor Rieffel for the insightful discussions on the QCC problem, and Kyle Booth for providing us the source code for CP solver.

References

Amy, M.; Maslov, D.; Mosca, M.; and Roetteler, M. 2013. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 32:6, 818–830.

Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J. C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F. G.; Buell, D. A.; et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574(7779):505–510.

Booth, K. E. C.; Do, M.; Beck, J. C.; Rieffel, E.; Venturelli, D.; and Frank, J. 2018. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling*.

Carchrae, T., and Beck, J. C. 2005. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

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*.

Corcoles, A.; Kandala, A.; Javadi-Abhari, A.; McClure, D.; Cross, A.; Temme, K.; Nation, P.; Steffen, M.; and Gambetta, J. 2019. Challenges and opportunities of near-term quantum computing systems. *arXiv preprint arXiv:1910.02894*.

Cowtan, A.; Dilkes, S.; Duncan, R.; Krajenbrink, A.; Simmons, W.; and Sivarajah, S. 2019. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*.

Cserna, B.; Ruml, W.; and Frank, J. 2017. Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling*.

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*.

Gokhale, P.; Ding, Y.; Propson, T.; Winkler, C.; Leung, N.; Shi, Y.; Schuster, D. I.; Hoffmann, H.; and Chong, F. T. 2019. Partial compilation of variational algorithms for noisy intermediate-scale quantum machines. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 266–278. ACM.

Hansen, E. A., and Zilberstein, S. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence* 126(1-2):139–157.

Hay, N., and Russell, S. J. 2011. Metareasoning for monte carlo tree search. *Technical Report UCB/EECS-2011-119, EECS Department, University of California, Berkeley*.

Horvitz, E., and Rutledge, G. 1991. Time-dependent utility and action under uncertainty. In *Proceedings of the 7th conference on Uncertainty in Artificial Intelligence*.

Itoko, T.; Raymond, R.; Imamichi, T.; and Matsuo, A. 2019. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*.

Li, G.; Ding, Y.; and Xie, Y. 2019. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*.

Lieder, F.; Plunkett, D.; Hamrick, J. B.; Russell, S. J.; Hay, N.; and Griffiths, T. 2014. Algorithm selection by rational metareasoning as a model of human strategy selection. In *Advances in neural information processing systems*.

Lye, A.; Wille, R.; and Drechsler, R. 2015. Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits. In *Proceedings of the IEEE 20th Asia and South Pacific Design Automation Conference*.

Maslov, D.; Falconer, S. M.; and Mosca, M. 2008. Quantum circuit placement. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 27:4, 752–763.

Murali, P.; Linke, N. M.; Martonosi, M.; Abhari, A. J.; Nguyen, N. H.; and Alderete, C. H. 2019. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. *arXiv preprint arXiv:1905.11349*.

Oddi, A., and Rasconi, R. 2018. Greedy randomized search for scalable compilation of quantum circuits. In *Proceedings of the 15th Integration of Constraint Programming, Artificial Intelligence, and Operations Research*.

Rasconi, R., and Oddi, A. 2019. An innovative genetic algorithm for the quantum circuit compilation problem. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.

Russell, S., and Wefald, E. 1991. Principles of metareasoning. *Artificial intelligence* 49(1-3):361–395.

Shafaei, A.; Saeedi, M.; and Pedram, M. 2013. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Proceedings of the 50th Annual Design Automation Conference*.

Svegliato, J.; Wray, K. H.; and Zilberstein, S. 2018. Meta-level control of anytime algorithms with online performance prediction. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*.

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

Venturelli, D.; Do, M.; O'Gorman, B.; Frank, J.; Rieffel, E.; Booth, K. E. C.; Nguyen, T.; Narayan, P.; and Nanda, S. 2019. Quantum circuit compilation: An emerging application for automated reasoning. In *Scheduling and Planning Applications Workshop, ICAPS*.

Wille, R.; Lye, A.; and Drechsler, R. 2014. Optimal swap gate insertion for nearest neighbor quantum circuits. In *Proceedings of the 19th IEEE Asia and South Pacific Design Automation Conference*.

Zilberstein, S. 2008. Metareasoning and bounded rationality. *Metareasoning: Thinking about Thinking, MIT Press*.

Zulehner, A.; Paler, A.; and Wille, R. 2018a. An efficient methodology for mapping quantum circuits to the ibm qx architectures. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Zulehner, A.; Paler, A.; and Wille, R. 2018b. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.