Synthetiq: Fast and Versatile Quantum Circuit Synthesis

ANOUK PARADIS*, ETH Zurich, Switzerland
JASPER DEKONINCK*, ETH Zurich, Switzerland
BENJAMIN BICHSEL, ETH Zurich, Switzerland
MARTIN VECHEV, ETH Zurich, Switzerland

To implement quantum algorithms on quantum computers it is crucial to decompose their operators into the limited gate set supported by those computers. Unfortunately, existing works automating this essential task are generally slow and only applicable to narrow use cases. We present Synthetiq[1], a method to synthesize quantum circuits implementing a given specification over arbitrary finite gate sets, which is faster and more versatile than existing works. Synthetiq utilizes Simulated Annealing instantiated with a novel, domain-specific energy function that allows developers to leverage partial specifications for better efficiency. Synthetiq further couples this synthesis method with a custom simplification pass, to ensure efficiency of the found circuits.

We experimentally demonstrate that Synthetiq can generate better implementations than were previously known for multiple relevant quantum operators including RCCCX, CCT, CCiSWAP, C$\sqrt{\text{SWAP}}$, and C$\sqrt{\text{iSWAP}}$. Our extensive evaluation also demonstrates Synthetiq frequently outperforms a wide variety of more specialized tools in their own domains, including (i) the well-studied task of synthesizing fully specified operators in the Clifford+T gate set, (ii) ϵ-approximate synthesis of multi-qubit operators in the same gate set, and (iii) synthesis tasks with custom gate sets. On all those tasks, Synthetiq is typically one to two orders of magnitude faster than previous state-of-the-art and can tackle problems that were previously out of the reach of any synthesis tool.

CCS Concepts: • **Hardware** → **Quantum computation**; • **Software and its engineering** → **Compilers**.

Additional Key Words and Phrases: Quantum Circuits, Synthesis, Clifford+T

1 INTRODUCTION

Quantum computing promises to gain a significant advantage over classical computing by leveraging the principles of quantum mechanics [Arute et al. 2019; De Wolf 2017; Shor 1999]. However, for such an advantage to be realized in practice, quantum algorithms must be implemented and executed on a quantum computer. This requires bridging the gap between the high level constructs used in the description of those quantum algorithms, and the limited set of operations that can be executed on a quantum computer.

Decomposing Arbitrary Operators. Typically, quantum algorithms rely on arbitrary operators, described by input-output pairs. Such operators may act on many qubits simultaneously. In contrast, quantum computers, both existing [Alvarez-Rodriguez et al. 2018; Arute et al. 2019; Steffen et al. 2011] and planned [Bravyi et al. 2022; Gill et al. 2022] can only execute a limited set of operations acting on one or two qubits, often referred to as gates. Hence, implementing a quantum algorithm requires decomposing its operators into these supported gates. We refer to the resulting decomposition as a circuit. Crucially, the performance of quantum algorithms hinges on the efficiency of the circuits resulting from the decomposition.

*Equal Contribution
[1]Our implementation is available at https://github.com/eth-sri/synthetiq.

Authors' addresses: Anouk Paradis*, ETH Zurich, Switzerland, anouk.paradis@inf.ethz.ch; Jasper Dekoninck*, ETH Zurich, Switzerland, jasper.dekoninck@inf.ethz.ch; Benjamin Bichsel, ETH Zurich, Switzerland, benjamin.bichsel@inf.ethz.ch; Martin Vechev, ETH Zurich, Switzerland, martin.vechev@inf.ethz.ch.

Compilation to Finite Gate Sets. We focus on the problem of operator decomposition to a *finite* gate set. This is in particular relevant for fault-tolerant quantum computing, which invariably depends on finite gate sets, most frequently the Clifford+T gate set. Almost all existing synthesis tools [Di Matteo and Mosca 2016; Gheorghiu et al. 2022a,b; Mosca and Mukhopadhyay 2021; Niemann et al. 2020] for finite gate sets only support decomposition to this gate set. Further, they are generally slow and limited to decomposing fully specified operators, adding unnecessary constraints when an operator is only specified on selected inputs. Only three works [Amy et al. 2013; Chou et al. 2022; Kang and Oh 2023] allow for custom gate sets or partially specified operators. They are however prohibitively slow, and can therefore only handle really short decompositions of at most 20 gates.

This Work: Simulated Annealing for Circuit Synthesis. We present Synthetiq, a method to synthesize circuits over arbitrary finite gate sets using Simulated Annealing (SA), which iteratively modifies circuits until they meet a given specification. Our key insight is that SA is particularly well-suited for this problem when instantiated with a novel, domain-specific energy function that allows for partial specifications of quantum operators. Due to the inherent parallelizability of SA, Synthetiq is not only more versatile but also significantly faster than existing works, typically by one to two orders of magnitude on common synthesis tasks.

Versatility. Unlike prior works, Synthetiq can be employed in a wide range of different scenarios. First, as it does not rely on properties specific to the Clifford+T gate set, it supports adding custom gates or even changing the gate set completely. Crucially, *composite* gates, that is small operators whose decomposition in the original gate set is already known, can be added to the gate set in order to speed up Synthetiq.

Second, Synthetiq naturally supports partially specified operators. This is essential, as requiring developers to arbitrarily refine an underspecified operator restricts the search space and results in sub-optimal decompositions.

Third, Synthetiq can synthesize efficient circuits for many performance metrics, from total gate count to a specific gate depth. We note that, like most existing methods, Synthetiq cannot guarantee optimal circuits, but empirically succeeds in finding them given enough time.

Finally, for the cases where an operator cannot be exactly decomposed into the chosen finite gate set, Synthetiq can synthesize quantum circuits that approximate the given operator with a specified accuracy level.

Results. Leveraging Synthetiq, we can effectively synthesize a broader range of operators than any previous work. This allows us to uncover more efficient implementations of the operators RCCCX, CCT, CCiSWAP, C$\sqrt{\text{SWAP}}$, and C$\sqrt{\text{iSWAP}}$ using the Clifford+T gate set. When applied to five problems from StackExchange, Synthetiq outperforms the expert-written accepted answer twice and matches its efficiency otherwise.

Additionally, our evaluation shows that Synthetiq surpasses more specialized tools in performance and results, despite addressing a broader range of synthesis tasks. First, Synthetiq outperforms synthesis tools for both partially specified operators in custom gate sets [Kang and Oh 2023] and fully specified operators in the Clifford+T gate set [Gheorghiu et al. 2022a; Mosca and Mukhopadhyay 2021] by a factor of one to two orders of magnitude, often producing more efficient circuits. Second, Synthetiq's performance is on par with the state-of-the-art approach for ϵ-approximate synthesis of fully specified multi-qubit operators in the Clifford+T gate set [Gheorghiu et al. 2022b]. Finally, Synthetiq stands out as the first tool to successfully synthesize relative phase operators, an important case of incomplete specification. Those operators are in particular crucial for the efficient implementation of operators with multiple controls [Maslov 2016], and are used by Qiskit as one of the standard decompositions of the MCX operator [Qiskit 2023].

Main Contributions. To summarize, our main contributions are:

- Synthetiq, a fast and versatile synthesis algorithm for quantum operators over finite gate sets based on Simulated Annealing (§3–§4),
- a natural framework for partial specifications addressing common synthesis tasks (§5),
- an implementation and thorough evaluation of Synthetiq, showing that it outperforms more specialized tools and can tackle synthesis problems that were previously out of reach of any tool (§6).

In the following, we present the necessary background (§2), exemplify Synthetiq on an example (§3), formally describe Synthetiq (§4) and how it handles partial specifications (§5), evaluate Synthetiq (§6) and discuss related work (§7).

2 BACKGROUND

We now present the necessary background on quantum computation and Simulated Annealing.

Qubit. A qubit is the quantum counterpart of a classical bit. The state φ of a qubit x is a linear combination of the basis states $|0\rangle_x$ and $|1\rangle_x$, which we can write as $\varphi = \gamma_0 |0\rangle_x + \gamma_1 |1\rangle_x$ with $\gamma_0, \gamma_1 \in \mathbb{C}$. In the following, we often omit the subscript x indicating the qubit name when it is not relevant. The state of qubit x can equivalently be described by a state vector $\varphi = \left[\begin{smallmatrix}\gamma_0\\ \gamma_1\end{smallmatrix}\right]$.

To describe a system with multiple qubits, we use the tensor product $\otimes$. For instance, we can write the state of a system with two qubits x and y as

$$\varphi = \gamma_0 |0\rangle_x \otimes |0\rangle_y + \gamma_1 |1\rangle_x \otimes |0\rangle_y + \gamma_2 |0\rangle_x \otimes |1\rangle_y + \gamma_3 |1\rangle_x \otimes |1\rangle_y, \text{ or } \varphi = \begin{bmatrix}\gamma_0\\ \gamma_1\\ \gamma_2\\ \gamma_3\end{bmatrix}$$

with $\gamma_i \in \mathbb{C}$. We often abbreviate $|a\rangle_x \otimes |b\rangle_y$ to $|ab\rangle_{xy}$. We say the state vector describes φ in the *computational basis* $\{|00\rangle, |10\rangle, |01\rangle, |11\rangle\}$.

Quantum Gates and Circuits. Quantum compiling aims to produce circuits to be run on a quantum computer. Quantum circuits consist of a fixed number of qubits, and *gates* to be applied to some of those qubits. For instance, the X gate acts on one qubit and flips its value, or more formally $\mathrm{X}|a\rangle = |a \oplus 1\rangle$. More generally, X maps the state $\gamma_0 |0\rangle + \gamma_1 |1\rangle$ to $\gamma_0 |1\rangle + \gamma_1 |0\rangle$. Using the state vector representation, this operation can be described by the following matrix in $\mathbb{C}^{2^1 \times 2^1}$, which we refer to as $[\![\mathrm{X}]\!]$:

$$[\![X]\!] = \left[\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right]$$

Applying gate X to a qubit described by the state vector φ yields the new state vector $[\![\mathrm{X}]\!]\,\varphi$. Some quantum gates act on multiple qubits at the same time. For instance, the controlled X gate CX maps $|ab\rangle$ to $|a\rangle\,\mathrm{X}^a\,|b\rangle = |a\rangle\,|a \oplus b\rangle$; the second qubit is flipped iff the first one is 1. Again, the effect of CX can be described as a matrix. We finally introduce the identity gate I. It is the no-op of quantum gates, and its matrix representation when applied to n qubits is simply the identity matrix in $\mathbb{C}^{2^n \times 2^n}$.

Quantum Circuit Semantics. We can think of a quantum circuit as a list of gates and qubits to which each gate is applied. For instance, we can consider the circuit C on two qubits which applies the X gate to its first qubit, followed by the CX gate on both of its qubits. The effect of this circuit on two qubits can again be described by a matrix, which is simply the product of the matrices of each of its gates: $[\![C]\!] = [\![\mathrm{CX}]\!] \cdot ([\![\mathrm{X}]\!] \otimes [\![\mathrm{I}]\!])$. Note how we used the tensor product of $[\![\mathrm{X}]\!]$ with the identity I to extend the semantics of this one qubit gate to two qubits. In slight abuse of notation, we will typically write C instead of $[\![C]\!]$ throughout this work.

Quantum Operators. In the above, we described how the effect of a quantum gate or circuit on n qubits can be described by a matrix in $\mathbb{C}^{2^n \times 2^n}$. It is worth noting that all matrices representing the action of gates or circuits are unitary[2]. We say a *quantum operator* is an operation on qubits described by some unitary matrix U. We then say that some circuit C implements this operator if $[\![C]\!] = U$.

Clifford+T Gate Set. To implement a general quantum operator on a given quantum computer, we must decompose it into gates from the gate set $\mathcal{G}$ supported by this computer. Such gate sets $\mathcal{G}$ are usually *universal*, meaning that every quantum operator can be decomposed into gates from $\mathcal{G}$, with arbitrary precision $\epsilon > 0$. More formally, for any unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$ and $\epsilon > 0$, there exists some circuit C on n qubits using only gates from $\mathcal{G}$ such that $D(U, C) \leqslant \epsilon$, for some distance metric D. Fault-tolerant quantum computers will likely rely on the so-called Clifford+T gate set [Terhal 2015], which consists of the following gates:

$$\mathrm{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \mathrm{S} = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{bmatrix}, \quad \mathrm{S}^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -\mathrm{i} \end{bmatrix},$$
$$\mathrm{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\mathrm{i}\pi/4} \end{bmatrix}, \quad \mathrm{T}^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-\mathrm{i}\pi/4} \end{bmatrix}, \text{ and } \quad \mathrm{CX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

The Clifford+T gate set is known to be universal [Nielsen and Chuang 2002]. Further, Giles and Selinger [2013] showed that based on the coefficients of an operator U we can decide whether it can be decomposed exactly in the Clifford+T gate set, meaning there is a circuit C such that $D(U, C) = 0$. Based on the determinant of U, we can decide if this decomposition requires an *ancilla*, that is if U acts on n qubits, its decomposition will be a circuit C acting on $n + 1$ qubits, where the last extra qubit is assumed to be in state $|0\rangle$ initially, and must be returned to this state at the end.

Simulated Annealing. Simulated Annealing (SA) allows to efficiently find a state x which approximately minimizes an energy function $E(x)$, often referred to as an *energy*. Starting from some initial state, each SA step picks a randomly sampled neighbor state $x' \sim \mathcal{N}(x)$ of the current state x, and selects x' as the current state with some probability $P(x, x', T)$. Here, $T \in \mathbb{R}_{>0}$ denotes a progressively decreasing *temperature*. In this work, we set the acceptance probability P using the common approach $P(x, x', T) = \min\left(1, \exp\left(-\frac{E(x') - E(x)}{T}\right)\right)$. Thus a better x' (meaning $E(x') < E(x)$) ensures $P(x, x', T) = 1$ and therefore is always accepted. In contrast, a worse x' has $P(x, x', T) < 1$ and thus can be rejected or accepted, where acceptance is particularly likely initially, at high temperatures T.

3 OVERVIEW

We now explain the approach of Synthetiq by synthesizing a circuit over the Clifford+T gate set for an example operator. First, we introduce this operator and translate it to a partial specification (§3.1). We then show how Synthetiq finds an implementation for it (§3.2) and how composite gates can be used to speed up this search (§3.3). We describe Synthetiq in more detail in later sections.

3.1 Creating a Partial Specification

Controlled-T. The Controlled-T (CT) operator acts on two qubits, and applies the T gate to the second qubit if and only if the first qubit is 1. More formally, for $p, q \in \{0, 1\}$:

$$\mathrm{CT}\,|p\rangle\,|q\rangle = |p\rangle\,\mathrm{T}^p\,|q\rangle\,.$$

[2]We say a matrix A is unitary if $AA^\dagger = I$, where $A^\dagger$ is the conjugate transpose of A.

In matrix notation, CT corresponds to the operator shown below:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi/4} \end{bmatrix}$$

We note that CT can be represented exactly by a Clifford+T circuit [Giles and Selinger 2013, Theorem 1], but only if its circuit can make use of an ancilla [Giles and Selinger 2013, Corollary 2]. We now show how to encode the CT specification, taking into account this ancilla.

Encoding Ancillae. The effect of CT when given an ancilla is the following, again for $p, q \in \{0, 1\}$:

$$\mathrm{CT}\,|p\rangle\,|q\rangle\,|0\rangle = |p\rangle\,\mathrm{T}^p\,|q\rangle\,|0\rangle \tag{1}$$

$$\mathrm{CT}\,|p\rangle\,|q\rangle\,|1\rangle = ? \tag{2}$$

Note how the operator is undefined when the ancilla is not in state $|0\rangle$. This specification corresponds to the following underspecified matrix[3]:

$$\begin{array}{l} \qquad\qquad\qquad \mathrm{CT}\,|110\rangle = e^{i\pi/4}\,|110\rangle \\ \mathrm{CT}\,|000\rangle = |000\rangle \qquad\qquad \mathrm{CT}\,|101\rangle = ? \\ \begin{bmatrix} 1 & 0 & 0 & 0 & ? & ? & ? & ? \\ 0 & 1 & 0 & 0 & ? & ? & ? & ? \\ 0 & 0 & 1 & 0 & ? & ? & ? & ? \\ 0 & 0 & 0 & e^{i\pi/4} & ? & ? & ? & ? \\ 0 & 0 & 0 & 0 & ? & ? & ? & ? \\ 0 & 0 & 0 & 0 & ? & ? & ? & ? \\ 0 & 0 & 0 & 0 & ? & ? & ? & ? \\ 0 & 0 & 0 & 0 & ? & ? & ? & ? \end{bmatrix} \end{array} \tag{3}$$

Here we denote all unspecified elements with "?". When comparing such an underspecified matrix to the operator induced by a given circuit, Synthetiq only takes the specified elements into account. More precisely, we say a circuit C satisfies the specification if the matrix of the circuit operator matches all specified coefficients of the underspecified matrix. Although our example matrix consists of only fully known or fully unknown columns (i.e., *isometries*), we note that Synthetiq can also handle partially specified columns.

3.2 Running Synthetiq

We now describe how Synthetiq builds a quantum circuit from a gate set and a partial specification, following Fig. 1.

Sampling an Initial Circuit. For a given input, Synthetiq executes multiple separate runs of SA. Each run starts by sampling a random circuit C. To this end, Synthetiq first samples a circuit size within the circuit size bounds $\ell_{\min}$ and $\ell_{\max}$, and then a gate from the gate set (augmented with the identity gate) for each position in the circuit. Then, Synthetiq runs SA starting from C.

Simulated Annealing. Each SA step randomly replaces one gate in the current circuit C, yielding the new circuit C'. Comparing both circuits using a custom energy function E, Synthetiq decides whether to keep C or replace it by C'. Specifically, if C' is strictly better, we always replace C, otherwise we do so probabilistically.

[3]We use the little-endian convention, i.e., input state $|pqr\rangle$ corresponds to column $r \cdot 4 + q \cdot 2 + p \cdot 1$ in Eq. (3). This is analogous to existing tools such as Qiskit [Abraham et al. 2019].

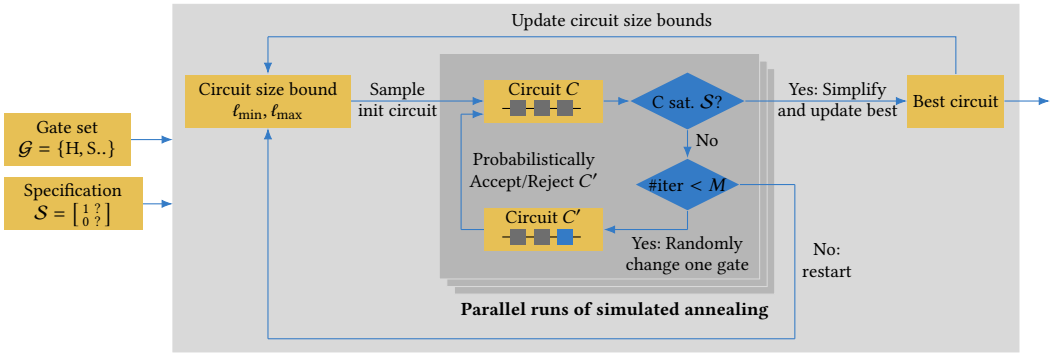


Fig. 1. Synthetiq overview. Here $M = 200 \cdot \text{len}(C)$ is the maximum number of iterations of each SA run.

Energy Function. We describe our energy function in detail in §4.4. In a nutshell, it measures how close the operator implemented by the circuit is to the given specification. The key insight of Synthetiq is the extension of an energy function used in e.g. [Chou et al. 2022; Khatri et al. 2019; Meister et al. 2022] to work for partial specifications. Such an extension is non-trivial, as it must gracefully decrease for circuits that "almost" satisfy a specification in order to guide SA efficiently. Further, it must be scaled to work for all possible underspecified operator sizes without requiring tuning of Synthetiq to each underspecified case.

Found Circuit. At every step of the SA algorithm, we check whether the current circuit satisfies this specification. If so, we run a simplification pass on it. This simplification is a simple and fast algorithm that heuristically optimizes both the total gate cost and the depth of user specified gates. Here, the total gate cost is the sum of the implementation cost for each gate, which is specified by the user. If the simplified circuit is the best one found so far, we record it. Finally, we use this best circuit to update the circuit size bounds for the next initial circuit. If no circuit is found in a reasonable number of steps, we abort SA and start a new run from a fresh random circuit.

Parallelization. As Synthetiq executes many short runs of SA, we can easily parallelize it. This leads to an almost linear speedup in the number of cores, greatly increasing Synthetiq's speed and allowing it to synthesize larger circuits than previous work, as demonstrated in §6.

3.3 Speeding up Synthetiq with Composite Gates

The versatility of Synthetiq extends beyond the Clifford+T gate set, as it is designed to work with any finite gate set. This includes the ability to integrate *composite* gates, that is operators whose decomposition in the gate set is already known. This *stratified synthesis* [Heule et al. 2016] significantly broadens the size of operators that Synthetiq can decompose as it can dramatically boost the speed of the search process. This is because a single circuit mutation can introduce a complex operator that would otherwise require a large amount of precise mutations. For instance, the inclusion of the RCCX operator as a composite gate enabled us to find an optimal implementation of the CT-operator within seconds, while finding one without the composite gate took 12 hours.

However, it is important to note that while the direct insertion of a complex gate can speed up the search process, it may not always yield the most efficient circuits. There could be a simpler, more efficient implementation that could only be discovered after a more thorough run of Synthetiq.

4 SYNTHETIQ

We now describe our method in more detail. For more details on hyperparameter optimization and values, we refer to Tab. 3 in §6.1.

Algorithm 1 Our main algorithm, Synthetiq.

```
1:  function SYNTHETIQ(𝒢: Gate set, 𝒮: Circuit specification)
2:      C_best ← ∅
3:      for i = 1, ..., m_iter do
4:          ℓ ← UNIFORM(ℓ_min, ℓ_max)
5:          C ← RANDOMCIRCUIT(𝒢, ℓ)
6:          for j = 1, ..., n_steps do
7:              C′ ← RANDOMMUTATION(C, 𝒢)
8:              e, e′ ← E(C, 𝒮), E(C′, 𝒮)
9:              if ACCEPT(e, e′) then
10:                 C ← C′
11:             if C satisfies 𝒮 then
12:                 C ← SIMPLIFY(C)
13:                 if COST(C) < COST(C_best) then
14:                     C_best ← C
15:                 ℓ_min, ℓ_max ← UPDATEBOUNDS(C_best)
16:                 break
17:     return C_best
```

End-to-End Procedure. Alg. 1 describes our main algorithm, Synthetiq. It takes as input a gate set (§4.1) and a (partial) specification (§4.2). Lin. 3–16 then execute multiple separate runs of SA. The starting point of each run is a fresh random circuit C (Lin. 5) made of ℓ randomly chosen gates. We select the number of gates ℓ uniformly at random within the current circuit size bounds ℓ_{min} and ℓ_{max} (Lin. 4). Lin. 6–16 then run n_{steps} SA steps from this initial circuit.

At each SA step, Lin. 7 creates a new candidate circuit C' by randomly changing one gate in the current circuit C (§4.3) and scores the two circuits using the energy function E (Lin. 8). This energy function captures how close the circuit is to the given specification (see §4.4). Lin. 9 then accepts the new circuit C' with a probability depending on the energy of both C and C' (see §4.5).

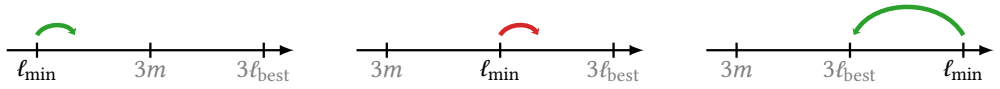
If the selected circuit does not satisfy the specification, Synthetiq proceeds to the next SA step (Lin. 11). Otherwise, Lin. 12 simplifies it (§4.6). Then, if it is the best circuit found so far (Lin. 13), Lin. 14 records it. Finally, Lin. 15 uses the best circuit found so far to update the circuit size bounds ℓ_{min} and ℓ_{max} (§4.3).

4.1 Gate Sets

Synthetiq searches for a circuit implementing the given specification using only the gates in the input gate set $\mathcal{G}$. This set can be any finite set of gates, for instance the Clifford+T gate set or any user supplied custom gate set. As discussed in §3.3, we can also add composite gates to $\mathcal{G}$ in order to speed up Synthetiq.

4.2 Expressing Partial Specifications

A partial specification $\mathcal{S} = (U, M)$ consists of two matrices $M \in \{0,1\}^{2^n \times 2^n}$ and $U \in \mathbb{C}^{2^n \times 2^n}$. U is the operation we aim to implement and M is a boolean mask specifying which elements of U should be matched (marked with 1 in M) and which can be ignored (marked with 0). Note that elements of U corresponding to a 0 in M can be omitted in the specification—we typically write them as "?". We say a unitary matrix $V \in \mathbb{C}^{2^n \times 2^n}$ matches the specification $\mathcal{S} = (U, M)$ if and only if $M \cdot U = \exp(i\theta)M \cdot V$ for some $\theta \in \mathbb{R}$, where "$\cdot$" denotes element-wise multiplication and θ is an arbitrary global phase difference. Of course, there may be many such matrices. We will show in §5 how this natural framework for underspecification can be used to specify various useful applications in quantum computing.



(a) Case: $\ell_{\text{min}} \leq 3m \leq 3\ell_{\text{best}}$. (b) Case: $3m \leq \ell_{\text{min}} \leq 3\ell_{\text{best}}$. (c) Case: $3m \leq 3\ell_{\text{best}} \leq \ell_{\text{min}}$.

Fig. 2. Updating bounds on circuit size.

4.3 Building Circuits

We now explain how Synthetiq builds and modifies circuits.

Randomly Mutating Gates. To mutate a given circuit C into a new candidate C', Synthetiq picks a gate position uniformly at random, and replaces it with a randomly selected gate.

To this end, Synthetiq first decides whether or not to replace the selected gate by a placeholder "identity" gate, with probability $P_{\text{Id}} \in [0, 1]$. This replacement step is analogous to deleting a gate, while replacing an identity gate by another is analogous to inserting a gate. Hence, this approach allows us to work with fixed size circuits, while keeping the flexibility of gate insertion and deletion.

If the identity gate was not selected, Synthetiq chooses which gate to insert. It picks a gate in $\mathcal{G}$ uniformly at random and then select the qubits it acts on. Further, we multiply the probability of sampling composite gates by $P_{\text{comp}} \in [0, 1]$ in order to avoid inserting these more expensive gates into the circuit too frequently.

Building the Initial Circuit. To build the initial circuit, we select a circuit size ℓ, and generate a circuit by randomly selecting ℓ gates as described above.

We have found empirically that Synthetiq performs best when ℓ is around 3 times the length of the optimal circuit for the given specification, which we denote here by m. As m is not known when running Synthetiq, we use an adaptive scheme to pick ℓ. To this end, we define minimal and maximal sizes ℓ_{min} and ℓ_{max}, and sample ℓ uniformly between the two for each new initial circuit. If a circuit is found that implements the specification in an SA step, we use the length of the current optimal circuit ℓ_{best} to move ℓ_{min} closer to $3m$, and analogously for ℓ_{max}:

$$\ell_{\text{min}} = \begin{cases} f_{\text{min}}\,\ell_{\text{best}} & \text{if } f_{\text{min}}\,\ell_{\text{best}} \leq \ell_{\text{min}} \\ \ell_{\text{min}} + \max\left(1, \left\lfloor \beta(f_{\text{min}}\,\ell_{\text{best}} - \ell_{\text{min}}) \right\rfloor\right) & \text{otherwise} \end{cases} \tag{4}$$

$$\ell_{\text{max}} = \begin{cases} f_{\text{max}}\,\ell_{\text{best}} & \text{if } f_{\text{max}}\,\ell_{\text{best}} \leq \ell_{\text{max}} \\ \ell_{\text{max}} + \max\left(1, \left\lfloor \beta(f_{\text{max}}\,\ell_{\text{best}} - \ell_{\text{max}}) \right\rfloor\right) & \text{otherwise} \end{cases} \tag{5}$$

We now explain how Eq. (4) moves ℓ_{min} closer to $3m$; the intuition behind Eq. (5) is analogous.

Suppose for now that we set $f_{\text{min}} = 3$, then Fig. 2 illustrates three possible situations. In all three situations, $3m \leq 3\ell_{\text{best}}$, as m is the theoretical best circuit size, while ℓ_{best} is the best size found so far. Fig. 2a shows the typical case in the first steps of SA: as we pick ℓ_{min} to be small, it is typically smaller than $3m$. Then, the second case in Eq. (4) increases ℓ_{min} slightly, where the gray part accounts for rounding and increments smaller than 1. Fig. 2b shows a case where ℓ_{min} was increased to surpass $3m$, but still lies below $3\ell_{\text{best}}$. In this case, the second case in Eq. (4) further increases ℓ_{min}, which moves us further from $3m$, but eventually Synthetiq will find better circuits, thus decreasing $3\ell_{\text{best}}$. Finally, Fig. 2c shows a case where ℓ_{min} is larger than $3\ell_{\text{best}}$. Then, we know for sure that $3\ell_{\text{best}}$ is closer to the optimal value $3m$, so we directly update ℓ_{min} to $3\ell_{\text{best}}$.

We note that since our estimate ℓ_{best} of m becomes better with every circuit found (no matter whether the current circuit improved ℓ_{best} or not), we apply this update rule every time a circuit is found. When Synthetiq is run on multiple threads, ℓ_{min} and ℓ_{max} are synchronized across threads.

4.4 Evaluating Circuits

To evaluate a circuit C with respect to a specification $\mathcal{S}$, we need to define an energy function $E(\mathcal{S}, C)$ that measures the distance between C and $\mathcal{S}$.

Various works have used measures inspired by *fidelity* to compare the matrices of quantum operators [Chou et al. 2022; Khatri et al. 2019; Meister et al. 2022]. Such measures are typically of the shape[4] $D(U,C) = 1 - \frac{|\mathrm{Tr}(U^\dagger C)|}{2^n}$ and have the important property that if U and C differ by a global phase, i.e., if $U = e^{i\theta}C$ for some $\theta \in \mathbb{R}$, then $D(U,C) = 0$.

Intuitively, we want to generalize $D(U,C)$ to account for partial specifications by replacing U and C by $M \cdot U$ and $M \cdot C$, respectively. Unfortunately, the resulting energy $\tilde{D}$ is useless if only 0s are specified in U, which is relevant, e.g., when specifying relative phase operators (§5). In such a case, since $M \cdot U = 0$, the suggested energy $\tilde{D}$ is constant regardless of the current circuit:

$$\tilde{D}(U,C) = 1 - \frac{|\mathrm{Tr}((M\cdot U)^\dagger (M\cdot C))|}{2^n} = 1 - \frac{|\mathrm{Tr}(0)|}{2^n} = 1.$$

To address this problem, we first rewrite $D(U,C)$ to[5] (derived in App. A.1):

$$D(U,C) = \frac{1}{2}\frac{\left\| U - \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} C \right\|_F^2}{2^n}. \tag{6}$$

After this rewrite, we generalize D to account for partial specifications by replacing U and C:

$$\bar{D}(\mathcal{S}, C) = \frac{1}{2}\frac{\left\| M\cdot U - \frac{\mathrm{Tr}\left((M\cdot C)^\dagger (M\cdot U)\right)}{\left|\mathrm{Tr}\left((M\cdot C)^\dagger (M\cdot U)\right)\right|} M\cdot C \right\|_F^2}{2^n}. \tag{7}$$

Here, we define $0/0 := 1$ to account for the fact that $\mathrm{Tr}((M\cdot C)^\dagger (M\cdot U))$ could be zero. If $\mathcal{S}$ only specifies 0s, that is $M \cdot U = 0$, $\bar{D}$ simplifies to $\frac{1}{2}\frac{\|M\cdot C\|_F^2}{2^n}$, which still provides valuable information; at the entries where U is 0 (and M is 1), C might not be and the higher its values, the higher $\bar{D}$ is.

Further, we also adapt the normalization factor of 2^n in $\bar{D}$. This is crucial, as an incorrect normalization would make the magnitude of $D(U,C)$ sensitive to the number of specified elements $||M||_F^2$. To this end, we note that 2^n is equal to $\|1_n\|_F$, where 1_n is the matrix of dimension $2^n \times 2^n$ with all ones, that is the boolean mask for a complete specification. Therefore, we replace the normalization of 2^n by $||M||_F$. Taking the square root of all squared norms and dropping the factor of $\frac{1}{2}$ for simplicity then yields:

$$\underline{D}(\mathcal{S}, C) = \frac{\left\| M\cdot U - \frac{\mathrm{Tr}\left((M\cdot C)^\dagger (M\cdot U)\right)}{\left|\mathrm{Tr}\left((M\cdot C)^\dagger (M\cdot U)\right)\right|} M\cdot C \right\|_F}{\sqrt{\|M\|_F}}. \tag{8}$$

Measuring Distance Between Circuit and Specification. To speed up the search for circuits, we always evaluate not only the current circuit C against the specification $\mathcal{S} = (M, U)$, but also all the circuits C_σ that can be built from C by permuting its qubits following some permutation σ. Note that this is equivalent to evaluating C against any permutation $\mathcal{S}_\sigma = (M_\sigma, U_\sigma)$ of the specification. Here M_σ (resp. U_σ) is defined as $B_\sigma^{-1} M B_\sigma$ (resp. $B_\sigma^{-1} U B_\sigma$) where B_σ is the change-of-basis matrix from the original qubit order to their permutation. This gives finally:

$$E(\mathcal{S}, C) = \min_{\sigma \in S_n} \underline{D}(\mathcal{S}_\sigma, C),$$

[4] For a matrix M, $\mathrm{Tr}(M)$ is its trace and $M^\dagger$ its conjugate transpose.
[5] For a matrix M, $\|M\|_F = \mathrm{Tr}(MM^\dagger)$ is its Frobenius norm.

where S_n is the set of all permutations of $\{1, ..., n\}$. This is inspired by the equality metric for classical programs presented in [Schkufza et al. 2013]. There, if a program gives the correct result in the wrong register, the penalty is much smaller than if the result is not present at all.

Efficiently Computing the Circuit Matrix. To compute E, we need the matrix of the operator implemented by C. To compute it efficiently, we maintain a binary tree over the matrices of the list of gates in C. Hence, as each mutation only modifies one gate, we can update the complete matrix in only $O(\log(\text{len}(C)))$ matrix multiplications. This comes at the cost of an extra memory requirement, but this is not a limiting factor in practice.

Approximate Synthesis. Synthetiq can be readily adapted for approximate circuit synthesis. We simply treat a circuit as discovered once the condition $\underline{D}(S, C) \leqslant \sqrt{2}\epsilon$ is met (see Lin. 11 in Alg. 1). In the context of a complete specification, this corresponds exactly to the global phase invariant distance employed in previous studies, such as Gheorghiu et al. [2022b].

4.5 Updating the Current Circuit

As mentioned in §2, SA accepts a new circuit C' with a certain probability depending on a temperature function T. Modifications leading to a better circuit ($E(\mathcal{S}, C) < E(\mathcal{S}, C')$) are therefore always accepted, whereas modifications leading to a worse circuit are only accepted occasionally. The temperature function $T(i)$ governs the acceptance rate of such worse modifications: increasing it means that worse modifications are more likely to be accepted. We define T as:

$$T(i) = T(0) \exp\left(\frac{-n_{\text{accept}}}{\ell n_{\text{norm}}}\right),$$

where n_{accept} is the total number of accepted modifications since the start of the SA run, ℓ is the number of gates in the circuit currently under consideration, and $T(0)$ and n_{norm} are hyperparameters. Intuitively, after many modifications were accepted, acceptance of a worse circuit becomes less likely, allowing to focus on a local optimum.

4.6 Simplifying a Circuit

SA allows Synthetiq to discover many new circuits implementing the given specification. However, in many cases, we are specifically looking for *efficient* circuits implementing this specification. The circuits found by SA can often be trivially simplified, for instance by replacing two consecutive gates that cancel out by the no-op identity gate I. We therefore developed a fast simplification pass to remove such inefficiencies from the found circuits. We first discuss two ways of measuring the efficiency of a circuit and then describe our simplification pass.

Cost of a Circuit. The first way of measuring efficiency is by looking at the number of gates the circuit is made of. Typically, if each gate has a cost of execution on the quantum computer (be it in time or loss of precision), the cost of a circuit is simply the sum of the cost of each of its gates. For some applications (see §6.3), we assume the cost of all gates is the same and equal to 1. In contrast, for fault-tolerant quantum computing, the T gate is much more expensive to implement than any of the other gates in the Clifford+T gate set. To compute the cost of a circuit in this gate set, we use the following gates costs, roughly reflecting gate complexity on hardware:

$$\textsc{Cost}(\text{T}) = \textsc{Cost}(\text{T}^\dagger) = 1, \textsc{Cost}(\text{CX}) = 0.1 \text{ and } \textsc{Cost}(\text{H}) = \textsc{Cost}(\text{S}) = \textsc{Cost}(\text{S}^\dagger) = 0.01. \quad (9)$$

In all gate sets, the identity gate has cost 0, as it does not apply any operation to the qubits.

Algorithm 2 Simplification applied as post-processing. C_i denotes the i-th gate in C, $C_{[i,k]}$ is the subset of C consisting of its i-th to k-th gates, and $C_{i \leftrightarrow k}$ is C after swapping the i-th and k-th gates.

```
1:  function SIMPLIFY(C: Circuit, 𝒢: Gate set)
2:      i, j ← 0, 0
3:      while i < len(C) do
4:          for k = 1, 2, ..., 12 do
5:              if ⟦C_[i,i+k]⟧ = ⟦g⟧ with g ∈ 𝒢 and COST(C_[i,i+k]) > COST(g) then
6:                  C_[i,i+k] ← [g, I, ..., I]
7:                  i ← i − 1
8:          if C_i and C_{i+1} commute and C_{i+1} ≺ C_i then
9:              C ← C_{i↔i+1}
10:             i ← i − 2
11:         i ← i + 1
12:     C ← reverse(C)
13:     while j < len(C) do
14:         if C_j and C_{j+1} commute then
15:             d_diff = DEPTH(C_{j↔j−1}) − DEPTH(C)
16:             if d_diff < 0 or (d_diff = 0 and C_{j+1} ≺ C_j) then
17:                 C ← C_{j↔j+1}
18:                 j ← j − 2
19:         j ← j + 1
20:     return reverse(C)
```

Depth of a Circuit. The second way of defining a circuit cost takes parallelism in its execution into account. If a circuit applies one gate on its first qubit and another on its second qubit, those two gates can often be executed *at the same time*. Therefore, the cost of the execution is only the cost of one gate, and not the sum of the two. The *depth* of a circuit reflects this cost. It is the length on the execution of the circuit, assuming all operations that can be are parallelized[6]. Further, in cases where some gates take much longer to execute than others, we may use as cost for the circuit its depth when only considering those expensive gates. This is typically the case for the Clifford+T gate set, where we measure T-depth.

Optimizing Found Circuits. We show our simplification pass in Alg. 2. It consists of two main parts. The first aims at minimizing the gate cost of the circuit (Lin. 3–11). More specifically, it replaces sequences of gates in the circuit with gates from $\mathcal{G}$ that have the same semantics if this gate has a lower cost than the complete sequence (Lin. 4–7). Note that we only consider sequences of up to 12 gates since higher values did not result in more efficient circuits. The second part aims at minimizing the depth of the circuit (Lin. 13–19). Here, it swaps gates that commute[7] if doing so would reduce the depth of the circuit. Finally, to create more opportunities for both optimizations, Alg. 2 also swaps any gates that commute, in both parts (Lin. 8–10 and Lin. 17). To ensure we don't endlessly swap gates back and forth, we only do so according to a custom total order on gates $\prec$. We give more details about this order in App. A.2.

A Custom Pass. We note that this simplification pass is specifically tuned to our SA algorithm. It is both fast and focused on simple optimizations that are easily missed by SA and can be applied for any finite gate set. Further, we found in practice that for a given circuit, optimizing for gate count and depth were not at odds. We therefore always optimize for both. This simplification pass is an essential part of the algorithm and is not meant to be used as a standalone procedure. Indeed, when applied to circuits found by other synthesis tools, it most often does not find any simplifications.

[6]This can be computed in time linear in the number of gates in the circuit.
[7]That is applying one then the other has the same effect on qubits as doing the opposite.

Table 1. Translating common tasks to incomplete specifications.

Task	Full	Isometry	Element-wise
State Preparation		●	
Relative Phase Operators			●
Ancillae		●	
Oracles		●	
Dirty Qubits	●		
Isometry + Ancilla			●

5 LEVERAGING PARTIAL SPECIFICATIONS

We now show how to leverage our partial specification framework to express common tasks when implementing quantum algorithms.

Classification. Recall that in a partial specification $\mathcal{S} = (U, M)$, the boolean matrix M specifies which elements of matrix U should be matched. When each column of M is either all ones or all zeros (i.e., each column is fully specified or not at all), the partial specification is an *isometry*. Otherwise, we refer to the specification as *element-wise*.

Tasks. Tab. 1 summarizes the tasks discussed in §5.1–§5.5, and whether they can be expressed as a full specification, an isometry, or require element-wise specification in the general case. We note that multiple tasks can be combined. For instance, allowing an isometry to use an ancilla yields a new, element-wise specification.

5.1 State Preparation

The task of state preparation asks to implement an operator that brings qubits from some initial state (typically $|0...0\rangle$) to some target state φ. This operator is only specified for the input $|0...0\rangle$ and can be written as $U = \left[\begin{array}{c|c} \varphi & ? \end{array}\right]$, where φ denotes the vector representation of the target state. State preparation applications include quantum chemistry ([Cao et al. 2019]), quantum machine learning ([Araujo et al. 2021]), and solving systems of linear equations ([Harrow et al. 2009]). For example, the specification to prepare the GHZ state for two qubits $|\text{GHZ}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ \frac{1}{\sqrt{2}} & ? & ? & ? \end{bmatrix}.$$

5.2 Relative Phase Operators

We say an operator U' is a relative phase operator for operator U if for any input state in the computational basis $|k\rangle$ there exists some phase θ_k such that $U'|k\rangle = e^{i\theta_k} U|k\rangle$. Such relative phase operators often have a shorter circuit implementation than their non-relative original. Therefore, it can be interesting to replace U with its relative counterpart when it is used in a bigger computation, if this replacement does not change the overall computation. Common use cases for relative phase operators include their use in more efficient implementations of their non-relative counterpart [Maslov 2016], replacing the CCX gate by a relative RCCX gate when it is later uncomputed [Paradis et al. 2021], or when the non-relative counterpart is used in a circuit that is measured directly after the application of the operator.

When U can be described classically, i.e., when it maps all computational basis states to another basis state, we know that its matrix representation consists of 0s and 1s. In this case, we can simply

replace each "1" with a question mark. As any operator built by Synthetiq is unitary, any circuit it produces matching the specification will have values of norm 1 in place of the question marks.

For example, the partial specification for a relative phase operator of CCX transforms:

$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0 \end{bmatrix} \text{ to } \begin{bmatrix} ?&0&0&0&0&0&0&0\\ 0&?&0&0&0&0&0&0\\ 0&0&?&0&0&0&0&0\\ 0&0&0&?&0&0&0&0\\ 0&0&0&0&?&0&0&0\\ 0&0&0&0&0&?&0&0\\ 0&0&0&0&0&0&0&?\\ 0&0&0&0&0&0&?&0 \end{bmatrix}.$$

5.3 Operators with Ancillae

As discussed in §3, an ancilla is an extra qubit used to help implementing an operator on the other qubits. We assume the ancilla is initially in state $|0\rangle$ and must be returned to the same state. In some cases, such an ancilla is necessary to implement the given operator using the chosen gate set [Giles and Selinger 2013]. In other cases, ancillae are not necessary but may allow for a shorter circuit implementation, e.g., CCX has lower T-depth when implemented with one ancilla [Amy et al. 2013]. To represent an operator U with ancillae A, we observe that any state $\varphi \otimes |0\rangle_A$ maps to $(U\varphi) \otimes |0\rangle_A$, while the result on any state $\varphi \otimes |1\rangle_A$ is unspecified. The resulting specification is thus: $\left[\begin{smallmatrix} U & ? \\ \mathbf{0} & ? \end{smallmatrix}\right]$, where $\mathbf{0}$ is the null matrix.

More generally, for an already partial specification $\mathcal{S} = (U, M)$, adding an ancilla changes the specification to $\mathcal{S}' = \left(\left[\begin{smallmatrix} U & ? \\ \mathbf{0} & ? \end{smallmatrix}\right], \left[\begin{smallmatrix} M & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \end{smallmatrix}\right]\right)$, where $\mathbf{1}$ is the all ones matrix of appropriate size. For example, adding an ancilla to an isometry gives an element-wise underspecification. An example of ancilla underspecification for the CT operator can be found in §3.

5.4 Oracles

Quantum algorithms often require computing a classical boolean function f. For instance, in Grover's algorithm [Grover 1996] f decides if a candidate value x is a solution to a given task. The quantum circuit implementation of f is called an oracle. Typically, this oracle O is assumed to be such that for any computational basis state $|x\rangle$, we have $\mathrm{O}\,|x\rangle\,|-\rangle = (-1)^{f(x)}\,|x\rangle\,|-\rangle$ where $|-\rangle$ is the shorthand notation for the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Using that $(I \otimes H)\,|x\rangle\,|-\rangle = |x\rangle\,|1\rangle$, we can build this oracle as $\mathrm{O} = \mathrm{O}' \circ (I \otimes H)$, where O' is such that $\mathrm{O}'\,|x\rangle\,|1\rangle = (-1)^{f(x)}\,|x\rangle\,|-\rangle$. Such an O' can easily be specified in our framework.

Take for instance the classical identity function $f\colon x \mapsto x$, where the corresponding O' is undefined on inputs $|x\rangle\,|0\rangle$ for $x \in \{0, 1\}$. For the other computational basis states, we have that $\mathrm{O}'\,|0\rangle\,|1\rangle = (-1)^0\,|0\rangle\,|-\rangle = \frac{1}{\sqrt{2}}\,|0\rangle\,(|0\rangle - |1\rangle)$, and $\mathrm{O}'\,|1\rangle\,|1\rangle = (-1)^1\,|1\rangle\,|-\rangle = \frac{1}{\sqrt{2}}\,|1\rangle\,(-|0\rangle + |1\rangle)$. This corresponds to the following incomplete specification:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} ?&?&1&0\\ ?&?&0&-1\\ ?&?&-1&0\\ ?&?&0&1 \end{bmatrix}.$$

5.5 Dirty Qubits

Dirty qubits are similar to ancillae, with the difference that they can initially be in any state and must return to that same state after the computation. Therefore, they allow for less underspecification, but have the advantage of not requiring any preparation for the extra qubit. Dirty qubits are for example used in Low et al. [2018] to do state preparation, allowing them to achieve significantly shorter circuits.

Given a specification $\mathcal{S} = (U, M)$, we can allow for an extra dirty qubit by using the specification $\mathcal{S}' = \left(\left[\begin{smallmatrix} U & \mathbf{0} \\ \mathbf{0} & U \end{smallmatrix}\right], \left[\begin{smallmatrix} M & \mathbf{1} \\ \mathbf{1} & M \end{smallmatrix}\right]\right)$, where $\mathbf{1}$ is the all-ones matrix of appropriate size. The null matrices $\mathbf{0}$ in $\mathcal{S}$

ensure that the dirty qubits remain in the same state before and after the computation, since a state flip of the dirty qubits would require a non-zero element at any of the positions of the null matrices.

Note that dirty qubits only lead to underspecification if $\mathcal{S}$ is already underspecified, and are therefore especially useful in this case. When we allow the use of a dirty qubit in the implementation of the example in §5.4, we get the specification:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} ? & ? & 1 & 0 & 0 & 0 & 0 & 0 \\ ? & ? & 0 & -1 & 0 & 0 & 0 & 0 \\ ? & ? & -1 & 0 & 0 & 0 & 0 & 0 \\ ? & ? & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ? & ? & 1 & 0 \\ 0 & 0 & 0 & 0 & ? & ? & 0 & -1 \\ 0 & 0 & 0 & 0 & ? & ? & -1 & 0 \\ 0 & 0 & 0 & 0 & ? & ? & 0 & 1 \end{bmatrix}.$$

6 EXPERIMENTAL EVALUATION

We now experimentally evaluate Synthetiq. We first explain our process for optimizing the hyper-parameters occurring in the SA algorithm (§6.1). We then demonstrate that due to its versatility and speed, Synthetiq can push the limits of circuit synthesis, synthesizing previously unknown decompositions of relevant quantum operators (§6.2). We finally evaluate the versatility of Synthetiq by running it in different modes (§6.3–§6.6) and comparing the results to synthesis tools specialized for each of these modes. Overall, our main findings are:

§6.2 Better operator decompositions. We show that Synthetiq finds better implementations than the currently best known ones for RCCCX, CCT, CCiSWAP, C$\sqrt{\text{SWAP}}$, and C$\sqrt{\text{iSWAP}}$.

§6.3 Custom gates. We show that Synthetiq can efficiently synthesize circuits with user-supplied custom gate sets, and outperforms the state-of-the-art [Kang and Oh 2023] in 50% of the cases (including 27% of cases where Kang and Oh [2023] fails to return any decomposition) while being equally optimal in all other cases.

§6.4 Clifford+T gate set. We show that when synthesizing completely specified operators over the Clifford+T gate set, Synthetiq outperforms the specialized state-of-the-art [Gheorghiu et al. 2022a; Mosca and Mukhopadhyay 2021]. Synthetiq is able to find circuits for more operators and those circuits are often more efficient and typically found one to two orders of magnitude faster.

§6.5 Approximate synthesis for the Clifford+T gates set. We show that for approximate synthesis on the Clifford+T gate set, Synthetiq is 6 times faster than the state-of-the-art approach specialized to this task [Gheorghiu et al. 2022b] for complex multi-qubit operators and, while slower, can find circuits that are on par with Gheorghiu et al. [2022b] for simpler single qubit operators.

§6.6 Relative phase gates. Finally, we show how using Synthetiq to synthesize small components of a bigger circuit allows for more efficient implementations. Specifically, by synthesizing a relative phase `carry` operator, we can reduce the T-count of the Cirq Adder [Cirq 2023] by more than 3x.

Implementation. We implemented Synthetiq using C++17 with the Eigen matrix library [Guennebaud et al. 2010] and openMP [OpenMP Architecture Review Board 2021] for parallelization. All experiments are conducted on a Linux machine with 500 GB RAM and two AMD EPYC 7601 2200MHz processors, with a total of 64 cores. In the practical implementation of Synthetiq, we do not specify m_{iter} (see Alg. 1). Instead, we report the average runtime averaged over 100 runs, where each run finishes as soon as a circuit with the desired property (e.g., T-count optimal) is found. For particularly time-consuming tasks, we instead average on as many runs as fit within a set time-limit (12 hours per task). Unless specified otherwise, we run Synthetiq on all 64 cores. Finally, for Tab. 4, we do not average over multiple runs and instead mention the total timeout instead, as well as the best circuit found within this time-out.

Table 2. Ablation study of Synthetiq. We report the average speedup of Synthetiq compared to Synthetiq with a specific component removed.

	Full Specification	Isometry	Element-Wise	Averaged
Circuit Cost Rewrite	0.94	1.30	47.01	16.42
Permutation of Qubits in Cost	1.12	1.25	1.41	1.26
Simplification Pass	3642.10	2056.13	2387.31	2695.18

Results Format and Correctness. The energy function of Synthetiq naturally checks correctness of the synthesized circuits, as we only consider a circuit to be found if its distance to the specification is 0. Further, Synthetiq explicitly produces the found circuit in the standard OpenQASM 2.0 language [Cross et al. 2017] and can therefore easily be imported to other frameworks such as Qiskit [Abraham et al. 2019]. Note that this is in contrast to other tools which often only output a resource count [Gheorghiu et al. 2022a,b; Mosca and Mukhopadhyay 2021]. Finally, all synthesized implementations from this section are made available with our implementation.

6.1 Hyperparameter Optimization

We describe how we validated Synthetiq's design choices and fine-tuned its hyperparameters, using a randomly generated benchmark of operators.

Generating Random Operators. To optimize Synthetiq's hyperparameters without overfitting to a specific domain, we built a set of random operators covering the many use cases of Synthetiq. The benchmark consists of 90 operators acting on 2, 3, or 4 qubits, whose shortest decomposition contains 10 gates in the Clifford+T gate set[8]. A third of these operators have full specifications, another third are isometries, and the last third have element-wise specifications. The performance metric for circuits is T-count for the remainder of this section.

Ablation Study. To evaluate our design choices, we ran Synthetiq on the benchmark described above (i) without rewriting the energy function (instead using Eq. (6)), (ii) without using qubit permutations to speed up the search (outlined in §4.4) and (iii) without the simplification pass (outlined in §4.6). The results, shown in Tab. 2, demonstrate the significant impact on runtime of each of these choices. In particular, the simplification pass is essential, increasing Synthetiq speed by orders of magnitude. Note that the speedup is less than one in only one case, namely not applying cost rewriting for full specification. As the two energy functions (Eq. (6) and Eq. (8)) are equivalent for fully specified operators, the slowdown is solely due to the slightly higher computational complexity of the rewritten energy function.

Optimizing Hyperparameters. We optimize the hyperparameters for Synthetiq, recalled in Tab. 3.

We first optimize the optimal number of starting gates ℓ on our random benchmark where all operators have shortest decomposition length 10. $\ell = 30$ was the optimal value. Optimizing ℓ on a few other random operators of different lengths, we confirmed that the optimal ℓ was consistently around three times the decomposition length size. As we aim to synthesize decompositions of 10 to 40 gates, we use this factor 3 and set $\ell_{\text{min,init}}$ to 30 and $\ell_{\text{max,init}}$ to 120.

Since f_{min} and f_{max} only start playing a large role for bigger operators, we could not optimize them efficiently on our benchmark. We chose to set f_{min} to 2.5 and f_{max} to 3.5, to achieve a higher variety of initial number of starting gates and while staying close to the optimal value 3.

Subsequently, we conduct a grid search for every parameter but β, scanning over a range of plausible values for each parameter and optimizing the average time taken to solve the random operators introduced above.

[8]More precisely, we ran Synthetiq for 10 minutes for each of those operators, and made sure that no decomposition shorter than 10 gates was found. It is hence likely that no shorter one exists.

Table 3. Hyperparameter values used in Synthetiq. n is the number of qubits (fixed by the operator specification) and ℓ the number of starting gates in a run (randomly sampled in $[\ell_{\min}, \ell_{\max}]$ for each run).

Parameter name	Parameter value	Description
n_{steps}	$40n\ell$	number of SA steps
P_{ld}	0.3	probability of replacing a gate by the identity gate
$T(0)$	$\frac{0.1}{2^n}$	initial temperature in SA
n_{norm}	80.0	normalization constant in the temperature function
$[\ell_{\min,\text{init}}, \ell_{\max,\text{init}}]$	$[30, 120]$	initial circuit size bounds
$[f_{\min}, f_{\max}]$	$[2.5, 3.5]$	factors in the update of the circuit size bounds
β	0.05	moving average factor in the update of the circuit size bounds
P_{comp}	0.2	factor for reduced mutation selection of composite gates

For the hyperparameters P_{ld} and P_{comp} we follow a slightly adjusted grid search procedure to ensure the found values perform well for larger operators too. First, as small operators do not require composite gates, optimizing P_{comp} directly is impossible. Instead, we add the RCCX gate as a composite gate and set P_{comp} to the highest value that does not slow down the synthesis speed by more than a factor of 2. This ensures that the inclusion of a unneeded composite gate does not slow down the synthesis process too much, while ensuring composite gates are still likely to be used for operators that do require this additional gate. In the case of P_{ld}, we observe that its optimal value is heavily influenced by the ratio of the optimal circuit size to the number of initial gates, ℓ. Indeed, as ℓ increases, the proportion of identity gates in the optimal circuit also increases, which in turn raises the optimal value of P_{ld}. Therefore, when optimizing P_{ld}, we set ℓ to 30 for all operators, which is the optimal value of ℓ for operators with 10 gates.

Lastly, we optimize β, the moving average factor used in updating circuit size bounds. As β is largely dependent on the size of the found circuits and is not significantly influenced by the specifications, we use one larger operator - the 4-qubit adder operator - to optimize this parameter.

6.2 Better Operator Decompositions

Using Synthetiq, we were able to provide new and more efficient decompositions of multiple relevant operators, shown in Tab. 4.

Operators. We briefly describe each of the operators in Tab. 4. The first is RCCCX, that is a relative controlled X with three controls. For any a, b, c, d in $\{0, 1\}$, it maps $|abc\rangle\,|d\rangle$ to $|abc\rangle\,e^{\mathrm{i}\phi_{abcd}}\mathrm{X}^{abc}\,|d\rangle$, flipping the last qubit d with unspecified phase $\phi_{abcd} \in \mathbb{R}$ if and only if all three controls are 1. This gate is extremely useful to decompose controlled X operators with more than three controls, as described in Maslov [2016]. Hence, finding a better implementation of RCCCX directly gives a better implementation of the controlled X with four controls, when using only Clifford+T gates. The next operator is CCT, that is the T gate with two controls, mapping $|ab\rangle\,|c\rangle$ to $|ab\rangle\,\mathrm{T}^{ab}\,|c\rangle$. CCiSWAP maps $|ab\rangle\,|cd\rangle$ to $\mathrm{i}\,|ab\rangle\,|dc\rangle$ if both a and b are 1, and to $|ab\rangle\,|cd\rangle$ otherwise. C$\sqrt{\text{SWAP}}$ is $\sqrt{\text{SWAP}}$ controlled by one qubit and C$\sqrt{\text{iSWAP}}$ is $\sqrt{\text{iSWAP}}$ controlled by one qubit where we used:

$$\sqrt{\text{SWAP}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}(1+i) & \frac{1}{2}(1-i) & 0 \\ 0 & \frac{1}{2}(1-i) & \frac{1}{2}(1+i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \sqrt{\text{iSWAP}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \mathrm{i}\frac{1}{\sqrt{2}} & 0 \\ 0 & \mathrm{i}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Baselines. No existing circuit synthesis tool could synthesize the operators in Tab. 4. First, Kang and Oh [2023] is the only tool that can express the partially specified synthesis problems of RCCCX, CCT, and C$\sqrt{\text{SWAP}}$. It is however too slow to obtain results within any reasonable timeframe, as it times out for the much simpler circuit CCX after 1 day. The remaining operators (CCiSWAP and C$\sqrt{\text{iSWAP}}$) are beyond the capabilities of all existing tools due to their size: the fastest synthesis

Table 4. New operator decompositions found with Synthetiq, using 64 cores. For each operator we gave Synthetiq the composite gate RCCX, except for CCiSWAP where we used CCX. Previous Best is the result of a best-effort search either found in previous work or using a standard decomposition as discussed in §6.2.

Operator	Synthetiq			Previous Best	
	Ancillae	T-depth	Time	Ancillae	T-depth
RCCCX	**0**	**5**	8h	**0**	8
CCT	**1**	9	2h	2	9
CCiSWAP	**0**	9	4h	1	6
C$\sqrt{\text{SWAP}}$	**1**	**6**	4h	**1**	29
C$\sqrt{\text{iSWAP}}$	**0**	**6**	4h	1	27

tool for T-depth, Gheorghiu et al. [2022a] again fails to find any result in 2 days for CCiSWAP, and yields incorrect results for C$\sqrt{\text{iSWAP}}$[9]. We hence had to manually combine existing operator decompositions and generic decomposition techniques for each of the operators in Tab. 4. We describe this manual effort in App. A.3.

Results. Qubits are often the scarcest resource in quantum computers. Reducing the number of ancillae, and hence qubits used by a quantum operator is crucial. For three of the operators shown in Tab. 4, Synthetiq was able to find a decomposition using fewer ancillae than previous state of the art[10]. Only for the CCiSWAP operator does this come at the cost of a slightly higher T-depth. For the two operators where state of the art decompositions already used the minimum amount of ancillae, Synthetiq was able to significantly reduce the T-depth of the operators: from 8 to 5 for RCCX, and 29 to 8 for C$\sqrt{\text{SWAP}}$. Further, note that all those results were obtained in only a few hours. Finally, now that those decompositions are known, they can easily be reused by any quantum compiler.

Exploiting Versatility. To generate the decompositions in Tab. 4, we heavily relied on the versatility of Synthetiq. First, incomplete specification was necessary for all operators requiring an ancilla and for RCCCX. Further, we used composite gates to speed up synthesis and hence boost the chances of success. More precisely, we added to the Clifford+T gate set the RCCX gate for all operators but CCiSWAP, where we instead added CCX. This allows the synthesis to directly leverage those complex gates, and hence speeds up the search. To pick which composite gate to add to the gate set, we consistently used the following procedure. If after running for one minute Synthetiq could not find any circuit satisfying the specification, we added RCCX to the gate set. If no circuit was found after running one more minute with RCCX, we replaced RCCX with CCX in the gate set. Note that these intermediate runs take at most 2 minutes, which is negligible compared to the total runtime for each operator.

6.3 Mode: Custom Gates

As mentioned above, Synthetiq can synthesize circuits using any finite custom gates set. The most recent work on quantum circuit synthesis also allowing for custom finite gate sets is Kang and Oh [2023]. We evaluate the applicability of both tools on the benchmark described below.

[9]More precisely, the implementation requires a circuit of the given operator, but different circuits all representing C$\sqrt{\text{iSWAP}}$ are encoded differently. This indicates something in the encoding phase of the implementation is wrong.

[10]We used the characterization from Giles and Selinger [2013] (mentioned in §2) to find out which operators can be implemented on Clifford+T with 0 (resp. one) ancilla. We then ran Synthetiq with the corresponding complete (resp. partial) specification.

Fig. 3. Comparison between Synthetiq and [Kang and Oh 2023] on the benchmark used by [Kang and Oh 2023]. A time-out of 1 day was set for both tools, but Synthetiq always reached a solution within 1 hour. N/A means that the tool returned an empty solution set.

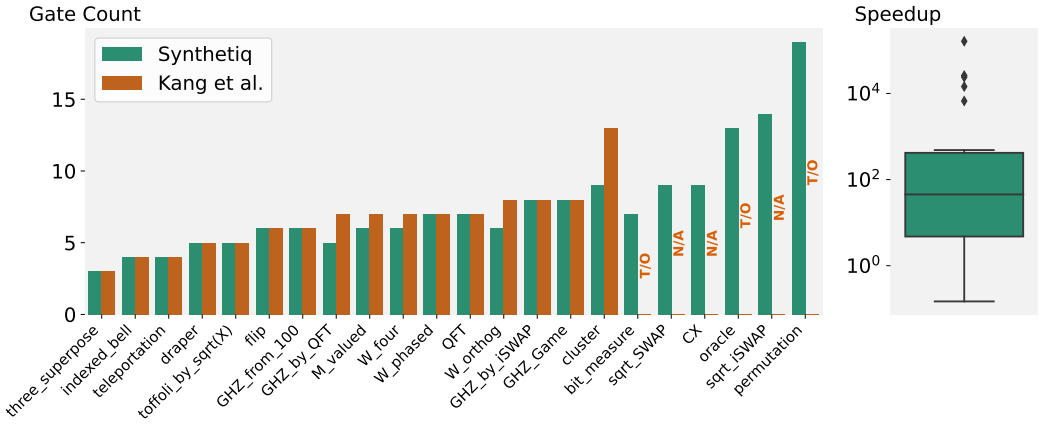


Table 5. Operators synthesis for several questions on StackExchange. Expensive gate count is the count of the most expensive gate (highlighted in bold). Time for Synthetiq is on 64 cores. [Kang and Oh 2023] was left out as it timed out (1 day) or returned an empty solution set for every problem.

Source	Operator	Qubits	Gate Set	Expensive Gate Count		Time Ours (s)
				Ours	StackExchange	
[StackExchange 2018]	$\sqrt{\text{SWAP}}$	2	H, S, S^{-1}, CX, **T**, $\mathbf{T}^{-1}$	**3**	**3**	0.04
[StackExchange 2021]	CX	2	H, S, S^{-1}, T, T^{-1}, **U**	**1**	**1**	0.02
[StackExchange 2022]	Oracle	4	H, S, S^{-1}, CX, **T**, $\mathbf{T}^{-1}$	**4**	8	3.08
[StackExchange 2023]	$\sqrt{\text{iSWAP}}$	2	H, S, S^{-1}, T, T^{-1}, $\mathbf{\sqrt{CZ}}$	**2**	4	0.09
[StackExchange 2020]	Permutation	3	H, S, S^{-1}, CX, **T**, $\mathbf{T}^{-1}$	**7**	**7**	0.32

Benchmark. Each element in the benchmark consists of a circuit specification and a custom gate set. The first part of the benchmark (`three_superpose` to `bit_measure` in Fig. 3) is the evaluation benchmark from Kang and Oh [2023]. However, these synthesis problems are not entirely realistic. They assume the gates required to build a circuit for the specification are known ahead of time, and supply exactly those gates in the gate set. This results in small gate sets (three or less gates for 11 of the 17 problems), and hence easier synthesis. We therefore complete the benchmark with real-world problems taken directly from Quantum Computing Stack Exchange. Gate sets and specifications are taken directly from the questions, resulting in bigger gate sets, where some gates are not used in the optimal decomposition.

Results. We show the results in Fig. 3 and Tab. 5. Fig. 3 shows the results when optimizing for gate count. We see that Synthetiq outperforms Kang and Oh [2023] in 50% of cases, and matches it on the rest. Further, Kang and Oh [2023] is not able to find any decomposition for one of their problems as well as the more complex problems we added to the benchmark, even with a one day time out. This shows that Kang and Oh [2023] is not scalable to those new complex problems, whereas Synthetiq still easily handles those in less than 4 seconds.

In Tab. 5, we focus on the questions taken from StackOverflow, for the more realistic objective of minimizing the use of the most expensive gate in the gate set. We compare Synthetiq results to the

Table 6. Synthesis of common fully specified operators using Clifford+T. We denote Mosca and Mukhopadhyay [2021] as Mosca and Gheorghiu et al. [2022a] as Gheorghiu. Speedup is the ratio of the time taken by the other tool to the time taken by Synthetiq. Times were measured on 64 cores for Synthetiq and Gheorghiu, and on a single core for Mosca.

	T-count		Time		T-depth		Time	
Operator	Synthetiq	Mosca	Synthetiq (s)	Speedup	Synthetiq	Gheorgiu	Synthetiq (s)	Speedup
CCX	**7**	**7**	0.1	**107.5**	**3**	**3**	0.3	**449.1**
CCH	**9**	-	6.4	-	**4**	**4**	55.9	0.5
Adder	**7**	**7**	11.0	**78.9**	**2**	-	43 200.0	-
U_1	**11**	**11**	2273.7	**6.2**	**5**	6	6171.4	**6.5**
U_1 var.	**11**	14	2700.0	**6.0**	**5**	-	6171.4	-
U_2	**7**	**7**	12.9	**62.3**	**3**	-	111.0	-

expert accepted answer on StackExchange. Note that we do not show results from Kang and Oh [2023] as they could not find any of the circuits within a day. We find that Synthetiq outperforms the expert answer in two out of five cases, and matches it in the remaining three cases. Further, all results were found within a few seconds, confirming the usefulness of Synthetiq for quantum programmers.

6.4 Mode: Clifford+T Gate Set

We now compare Synthetiq to the state of the art for the well-studied problem of synthesizing fully specified quantum operators over the Clifford+T gate set. When optimizing T-count, the current state of the art is Mosca and Mukhopadhyay [2021], while for T-depth it is Gheorghiu et al. [2022a]; we provide a broader overview of existing tools in §7.

Overall, we find that Synthetiq is generally faster than both tools, and finds strictly better or equally good implementations compared to either of them.

Benchmarks. Tab. 6 shows the comparison of Synthetiq to both works on a benchmark of common quantum operators, which is based on the original benchmark of Mosca and Mukhopadhyay [2021]. CCX, Adder, U_1, and U_2 are taken directly from their benchmark, where U_1 is defined as $\mathrm{CCX}(a,b,c);\mathrm{CCX}(c,b,d)$ and U_2 as $\mathrm{CCX}(a,b,c);\mathrm{CCX}(a,d,b);\mathrm{CCX}(a,b,c)$. We exclude the other 3-qubit operators from the original benchmark as they are affine equivalents of CCX and add the CCH operator to the benchmark instead. Based on U_1, we additionally introduce U_1 var. which we define as $\mathrm{CCX}(a,b,c);\mathrm{CCX}(b,c,d)$, allowing us to evaluate the sensitivity of all tools to simple changes of specifications.

We additionally show in Fig. 4 the comparison of Synthetiq with both tools on a benchmark of 3-qubits permutations, following Gheorghiu et al. [2022a][11]. We built this benchmark by clustering all 40320 permutations on 3 qubits by Clifford equivalence[12], and picking one representative for each of the resulting 30 equivalence classes.

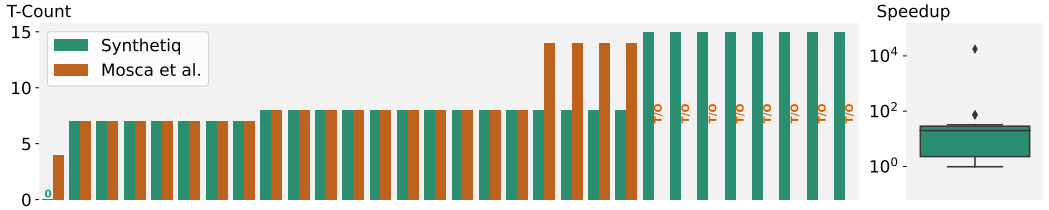
Results for Common Operators. We find that Synthetiq consistently finds the best implementation for each operator in the benchmark, outperforming Mosca and Mukhopadhyay [2021] in 33% of the cases and Gheorghiu et al. [2022a] in 66% of the cases. Further, Synthetiq finds these implementations faster than both tools in every example but one (CCH for T-depth).

[11] Gheorghiu et al. [2022a] only evaluated their tool on a few randomly generated permutations. We here systematize this benchmark.

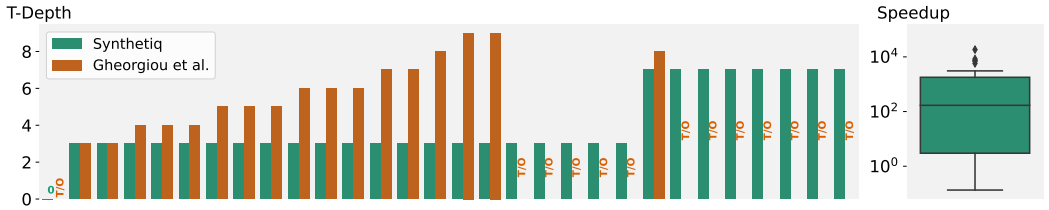
[12] We say two permutations P_1 and P_2 are Clifford equivalent iff there exists some circuit C made of the Clifford gates H, S, CX and $S^\dagger$ such that $P_1 = [\![C]\!]\, P_2$.

Fig. 4. Comparison between Synthetiq and previous works on 3-qubit permutation synthesis. Each bar represents one of the 30 evaluated permutations. Speedups are only included for cases where both tools return a result. Time-out set at 1h per example for all tools.

(a) Comparison with Mosca and Mukhopadhyay [2021] for the T-count of permutations.



(b) Comparison with Gheorghiu et al. [2022a] for the T-depth of permutations.



More importantly, Synthetiq does not time out (> 2 days compute) on any of the examples whereas Mosca and Mukhopadhyay [2021] and Gheorghiu et al. [2022a] do, showing that Synthetiq can handle more difficult problems than what could previously be done. We also note that Synthetiq is the first to automatically synthesize a T-depth 2 circuit for the Adder operator.

Results for Permutations. The results for all tools are displayed in Fig. 4. For the largest eight operators, we added RCCX to the gate set allowed for Synthetiq, following the procedure described in §6.2. Note that neither Mosca and Mukhopadhyay [2021] nor Gheorghiu et al. [2022a] allow for such composite gates, and hence cannot be extended when used for complex operators. We find that Synthetiq significantly outperforms both tools. Synthetiq finds a better T-count than Mosca and Mukhopadhyay [2021] in 43% cases, including 27% where Mosca and Mukhopadhyay [2021] times out. Further, Synthetiq is around one order of magnitude faster on the problems where Mosca and Mukhopadhyay [2021] does not time out.

For T-depth, Synthetiq finds more efficient circuits than Gheorghiu et al. [2022a] in 93% of cases, including 50% where Gheorghiu et al. [2022a] fails to find any circuit. Excluding the cases where Gheorghiu et al. [2022a] times out, Synthetiq is around two orders of magnitude faster than Gheorghiu et al. [2022a].

6.5 Mode: Approximate Circuit Synthesis

We compare Synthetiq with Gheorghiu et al. [2022b], the state-of-the-art method for approximate synthesis of multi-qubit operators in the Clifford+T gate set. Approximate synthesis is important as many operators cannot be implemented exactly with Clifford+T gates, but all can be approximated up to an arbitrary distance ϵ (see for instance [Nielsen and Chuang 2002, Chap. 4.5.3]).

Table 7. Results for approximate synthesis compared to [Gheorghiu et al. 2022b]. We run Synthetiq and [Gheorghiu et al. 2022b] for an hour on 1 or 2 qubit tasks and for two hours on the 3 qubit task and report the best found circuit. Synthetiq is run on 64 cores and [Gheorghiu et al. 2022b] is run on one. We tried running the code of [Gheorghiu et al. 2022b] on 64 cores, but this resulted in runtimes about 50 times **slower**.

Operator	Qubits	ϵ	k	T-Count		Time	
				Synthetiq	Gheorgiu	Synthetiq	Speedup
$R_Z(2\pi/2^k)$	1	0.05	4	**7**	8	0.059	0.085
			5	**9**	**9**	0.825	0.015
		0.01	4	**16**	18	3600.000	0.004
			5	-	**17**	-	x
			6	**16**	**16**	1800.000	0.001
			7	**11**	**11**	19.883	0.003
$cR_Z(2\pi/2^k)$	2	0.05	2	**2**	**2**	0.001	11.048
		0.01	2	**2**	**2**	0.004	4.847
cR_k	2	0.05	2	**3**	**3**	0.001	8.981
		0.01	2	**3**	**3**	0.005	2.296
$ccR_Z(2\pi/2^k)$	3	0.001	2	**12**	-	1800.000	-

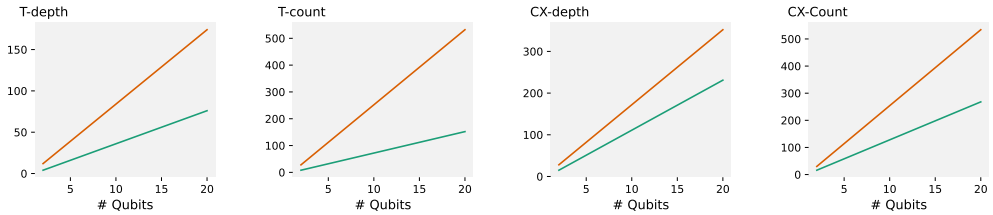


Fig. 5. Adder circuit performance with the original `carry` operator (–) and with the Synthetiq synthesized relative `carry` operator (–).

Tab. 7 shows the results of this comparison on the benchmark from Gheorghiu et al. [2022b]. We report results for all operators present in their evaluation, except for trivial operators with T-count less than 2 or for operators where neither tool reported any result.[13]

For operators on one qubit, Synthetiq finds more efficient circuits than Gheorghiu et al. [2022b] for two out of six operators, despite their claim of optimality[14]. Synthetiq is only outperformed once, when it fails to find any circuit. It is however several orders of magnitude slower than Gheorghiu et al. [2022b]. For operators on two qubits, Synthetiq is five times faster than Gheorghiu et al. [2022b] and finds circuits as efficient as Gheorghiu et al. [2022b] does. Further, Gheorghiu et al. [2022b] could not synthesize the three qubits operator, whereas Synthetiq succeeds.

6.6 Mode: Relative Phase Operators

We now showcase the use of Synthetiq for relative phase operators, and their use in bigger circuits. We do so using the Adder implementation from Cirq [Cirq 2023]. This implementation relies on the

[13]Further, we note that Gheorghiu et al. [2022b] reports finding a circuit with 26 gates for the operators $R_z\left(\frac{2\pi}{2^{10}}\right)$ and $R_z\left(\frac{2\pi}{2^{11}}\right)$ with $\epsilon = 0.001$, but we were not able to reproduce this result using their implementation. We confirmed with the authors that their reported result may indeed have been erroneous and we therefore dropped these operators as well.
[14]We confirmed with the authors that their implementation may indeed return sub-optimal results, which they conjecture may be due to various optimizations implemented but not discussed in the publication.

interleaving of three small operators: `sum`, `carry`, and `uncarry` (which uncomputes `carry`). To build an adder for two operands of n qubits with n ancilla qubits, this implementation uses n of each of the three operators. As we explained in §5.2, relative phase operators can be used to replace their non-relative counterpart in a circuit when this counter-part is later uncomputed. We can hence replace the `carry` operator by a relative implementation, and `uncarry` by the inverse of the relative operator, without changing the semantics of the resulting Adder circuit.

We used Synthetiq to synthesize such relative operators. This yielded two circuits: one optimized for T-count and T-depth, and one optimized for CX-count and CX-depth, each synthesized in less than 1h. Using those relative operators, we built the complete Adder circuit for different number of qubits. Note that all Adder circuits, no matter the number of qubits, use the same `carry` and `uncarry` operators. We therefore only synthesized two operators, and were able to use them for all adder operators.

We show the resulting circuit performance in Fig. 5. Using the relative operators allowed for significantly more efficient circuits; with a reduction in T-count by 3.5, in T-depth by 2.3, in CX-count by 2 and in CX-depth by 1.5. This demonstrates the usefulness of relative phase operators, and the need for a synthesis tool that can easily synthesize such operators for any specification.

7 RELATED WORK

We now discuss works related to Synthetiq.

Clifford+T Synthesis. Meet-in-the-middle (MITM) algorithms have been extensively explored for synthesizing circuits with finite gate sets. The original implementation by Amy et al. [2013] ensures gate-depth or T-depth optimality but is much slower than other methods, taking over four days to find a T-depth optimal CCX circuit. It supports ancillae by treating specifications that allow for ancillae as isometries, but does not discuss extending it to element-wise specifications. A later iteration of MITM [Gosset et al. 2013] focuses on optimizing T-count rather than T-depth, but sacrifices the use of ancillae. Di Matteo and Mosca [2016] improved upon Gosset et al. [2013] by introducing a parallel framework, thereby reducing runtime. Overall, those three MITM algorithms are extremely slow. For instance, the most efficient among them, Di Matteo and Mosca [2016], requires approximately 30 seconds to execute on 4096 cores for the smallest operator we considered, which is the CCX gate. Due to these excessive runtimes, they were not incorporated in the evaluation conducted in §6.4.

The more recent Mosca and Mukhopadhyay [2021] and Gheorghiu et al. [2022a] further refined the original MITM algorithm, optimizing for T-count and T-depth, respectively. However, they lost the original algorithm's optimality guarantees and cannot deal with ancillae[15]. It is worth noting that the former cannot be parallelized, and the latter gains only marginal benefits from parallelization. In §6.4, we demonstrate that Synthetiq outperforms both of those works, in terms of runtime and efficiency of the generated circuits.

Another work [Giles and Selinger 2013] suggests an algorithm to synthesize *any* circuit that can be exactly synthesized over Clifford+T. However, this work does not target efficient decompositions, instead often producing expensive ones. Niemann et al. [2020] implement and evaluate an improved version of this approach. Unfortunately, we were unable to compare their results to Synthetiq because we could not run their implementation and their publication does not report results on the circuits we consider here. [16]

[15]The *ancillae* mentioned by Gheorghiu et al. [2022a]; Mosca and Mukhopadhyay [2021] correspond to our notion of dirty qubits—any work that can handle full specification can trivially handle this.

[16]While the authors shared their implementation with us, we were unable to compile it, despite being in contact with them for over two months.

Synthesis on Other Gate Sets. We now discuss methods capable of handling gate sets other than Clifford+T and compare their capabilities with Synthetiq. Kang and Oh [2023] recently proposed a new circuit synthesis method focusing on finite gate sets and provide a framework for specifying isometries in any basis. Synthetiq, on the other hand, naturally handles partial specifications like relative phase operators that cannot be specified as isometries in any basis. More importantly, we demonstrated in §6.3 that Synthetiq significantly outperforms Kang and Oh [2023] in all tasks, both in terms of speed and efficiency of the generated circuits, even when restricted to a single core. Allowing Synthetiq to use multiple cores would only increase the performance gap further.

Chou et al. [2022] suggested an evolutionary algorithm that incrementally modifies a circuit to meet a specification. Even though it exploits known aspects of an optimal CCX gate decomposition, its reported runtime on CCX is orders of magnitude higher than Synthetiq, namely 600s. Unfortunately, its implementation is not available, so we were unable to compare it to Synthetiq. [17] Its publication does not address parallelization, partial specifications, or ancillae. Further, it assumes incorrect definitions of gate depth and T-depth. [18]

ϵ-approximate Clifford+T Synthesis. Since not all unitaries can be implemented exactly in the Clifford+T gate set, some works have focused on implementing circuits up to a distance ϵ, where the distance can be measured using distances that allow for a global phase difference (e.g., [Gheorghiu et al. 2022b; Kliuchnikov et al. 2016]) or not (e.g., [Ross and Selinger 2016; Selinger 2014]). Most of these works [Kliuchnikov et al. 2016; Ross and Selinger 2016; Selinger 2014] focus on single qubit operators; only Gheorghiu et al. [2022b] considers multi-qubit operators. While the latter claims its algorithm produces optimal circuits, our experiments demonstrate its implementation is not optimal for all tasks. As shown in §6.5, Synthetiq performs similarly to Gheorghiu et al. [2022b] on the subdomain of ϵ-approximate Clifford+T synthesis.

Synthesis in Other Settings. In contrast to the finite gate sets assumed by Synthetiq, various works have studied synthesis using parametrized gate sets such as CX+Rot [Davis et al. 2019; Khatri et al. 2019; Meister et al. 2022; Smith et al. 2023; Younis et al. 2021]. However, synthesis over CX+Rot relies on the optimization of the parameters in the rotational gates, which is not possible for finite gate sets such as Clifford+T. State preparation synthesis for the CX+Rot gate set has also been studied extensively, see e.g., [Araujo et al. 2021; Iten et al. 2016; Plesch and Brukner 2011].

A plethora of works synthesizes circuits for specific use cases. Various works decompose classical oracles into quantum circuits [Amy et al. 2017; Biswal et al. 2018; Green et al. 2013; Parent et al. 2015, 2017; Rand et al. 2019], or help with this task [Bhattacharjee et al. 2019; Paradis et al. 2021]. In contrast to these specialized algorithms, Synthetiq synthesizes general circuits.

8 CONCLUSION

We presented Synthetiq, a novel method and tool to synthesize quantum circuits over finite gate sets. Synthetiq is based on Simulated Annealing (SA) and allows us to solve a wide range of synthesis tasks from relative phase operators over Clifford+T to operators with ancillae over custom gates.

Our evaluation shows that Synthetiq (i) is able to synthesize more efficient implementations of relevant quantum operators, (ii) frequently outperforms more specialized synthesis tools such as synthesis for complete specification in the Clifford+T gate set, and (iii) can use relative phase operators to build more efficient implementations large n qubit operators.

We believe there are many more applications of Synthetiq worth exploring, such as topology-aware synthesis [Davis et al. 2020] or incomplete specifications of operators in different bases.

[17] We asked the authors for the implementation but did not receive a response within six months.

[18] The T-depth is measured as the number T-gates in different vertical layers in a drawing of the circuit. We tried to confirm this mistake with the authors but did not receive a response within six months.

DATA-AVAILABILITY STATEMENT

The implementation of Synthetiq and all evaluations results are available on github[19].

REFERENCES

Héctor Abraham, AduOffei, Rochisha Agarwal, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Eli Arbel, Arijit02, Abraham Asfaw, Artur Avkhadiev, Carlos Azaustre, AzizNgoueya, Abhik Banerjee, Aman Bansal, Panagiotis Barkoutsos, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Arjun Bhobe, Lev S. Bishop, Carsten Blank, Sorin Bolos, Samuel Bosch, Brandon, Sergey Bravyi, Bryce-Fuller, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Jerry M. Chow, Spencer Churchill, Christian Claus, Christian Clauss, Romilly Cocking, Filipe Correa, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Tareq El Dandachi, Marcus Daniels, Matthieu Dartiailh, DavideFrr, Abdón Rodríguez Davila, Anton Dekusar, Delton Ding, Jun Doi, Eric Drechsler, Drew, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Grant Eberle, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Axel Hernández Ferrera, Romain Fouilland, FranckChevallier, Albert Frisch, Andreas Fuhrer, Bryce Fuller, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Tanya Garg, Shelly Garion, Austin Gilliam, Aditya Giridharan, Juan Gomez-Mosquera, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, John A. Gunnels, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Frank Harkins, Vojtech Havlicek, Joe Hellmers, \Lukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Junye Huang, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Raban Iten, Toshinari Itoko, JamesSeaward, Ali Javadi, Ali Javadi-Abhari, Jessica, Madhav Jivrajani, Kiran Johns, Scott Johnstun, Jonathan-Shoemaker, Vismai K, Tal Kachmann, Naoki Kanazawa, Kang-Bae, Anton Karazeev, Paul Kassebaum, Josh Kelso, Spencer King, Knabberjoe, Yuri Kobayashi, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Prasad Kumkar, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Dennis Liu, Peng Liu, Yunho Maeng, Kahan Majmudar, Aleksei Malyshev, Joshua Manela, Jakub Marecek, Manoel Marques, Dmitri Maslov, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Thomas Metcalfe, Martin Mevissen, Andrew Meyer, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Jhon Montanez, Michael Duane Mooring, Renier Morales, Niall Moran, Mario Motta, MrF, Prakash Murali, Jan Müggenburg, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Johan Nicander, Pradeep Niroula, Hassi Norlen, NuoWenLei, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Raul Otaolea, Steven Oud, Dan Padilha, Hanhee Paik, Soham Pal, Yuchen Pang, Simone Perriello, Anna Phan, Francesco Piro, Marco Pistoia, Christophe Piveteau, Pierre Pocreau, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Daniel Puzzuoli, Jesús Pérez, Quintiii, Rafey Iqbal Rahman, Arun Raja, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Marcello La Rocca, Diego M. Rodríguez, RohithKarur, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, SamFerracin, Martin Sandberg, Hirmay Sandesara, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Joachim Schwarm, Ismael Faro Sertage, Kanav Setia, Nathan Shammah, Yunong Shi, Adenilton Silva, Andrea Simonetto, Nick Singstock, Yukio Siraichi, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, Igor Sokolov, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Shaojun Sun, Kevin J. Sung, Hitomi Takahashi, Tanvesh Takawale, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Davindra Tulsi, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Almudena Carrera Vazquez, Victor Villar, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Johannes Weidenfeller, Rafal Wieczorek, Jonathan A. Wildstrom, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, Steve Wood, James Wootton, Daniyar Yeralin, David Yonge-Mallo, Richard Young, Jessie Yu, Christopher Zachow, Laura Zdanski, Helena Zhang, Christa Zoufal, Zoufalc, a-kapila, a-matsuo, bcamorrison, brandhsn, nick bronn, chlorophyll-zz, dekel.meirom, dekelmeirom, dekool, dime10, drholmie, dtrenev, ehchen, elfrocampeador, faisaldebouni, fanizzamarco, gabrieleagl, gadial, galeinston, georgios-ts, gruu, hhorii, hykavitha, jagunther, jliu45, jscott2, kanejess, klinvill, krutik2966, kurarrr, lerongil, ma5x, merav-aharoni, michelle4654, ordmoj, sagar pahwa, rmoyard, saswati-qiskit, scottkelso, sethmerkel, strickroman, sumitpuri, tigerjack, toural, tsura-crisaldo, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and Mantas Čepulkovskis. 2019. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2562110

[19] https://github.com/eth-sri/synthetiq

Unai Alvarez-Rodriguez, Mikel Sanz, Lucas Lamata, and Enrique Solano. 2018. Quantum artificial life in an IBM quantum computer. *Scientific reports* 8, 1 (2018), 14793.

Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 6 (2013), 818–830.

Matthew Amy, Martin Roetteler, and Krysta M. Svore. 2017. Verified Compilation of Space-Efficient Reversible Circuits. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Vol. 10427. Springer International Publishing, Cham, 3–21. https://doi.org/10.1007/978-3-319-63390-9_1

Israel F Araujo, Daniel K Park, Francesco Petruccione, and Adenilton J da Silva. 2021. A divide-and-conquer algorithm for quantum state preparation. *Scientific reports* 11, 1 (2021), 1–12.

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

Debjyoti Bhattacharjee, Mathias Soeken, Srijit Dutta, Anupam Chattopadhyay, and Giovanni De Micheli. 2019. Reversible Pebble Games for Reducing Qubits in Hierarchical Quantum Circuit Synthesis. In *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*. 102–107. https://doi.org/10.1109/ISMVL.2019.00026 ISSN: 2378-2226.

Laxmidhar Biswal, Rakesh Das, Chandan Bandyopadhyay, Anupam Chattopadhyay, and Hafizur Rahaman. 2018. A template-based technique for efficient clifford+ t-based quantum circuit implementation. *Microelectronics Journal* 81 (2018), 58–68.

Sergey Bravyi, Oliver Dial, Jay M Gambetta, Darío Gil, and Zaira Nazario. 2022. The future of quantum computing with superconducting qubits. *Journal of Applied Physics* 132, 16 (2022).

Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Má ria Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. 2019. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews* 119, 19 (aug 2019), 10856–10915. https://doi.org/10.1021/acs.chemrev.8b00803

Yao-Hsin Chou, Shu-Yu Kuo, Yu-Chi Jiang, Ching-Hsuan Wu, Jyun-Yi Shen, Cheng-Yen Hua, Pei-Shin Huang, Yun-Ting Lai, Yong Feng Tong, and Ming-He Chang. 2022. A novel quantum-inspired evolutionary computation-based quantum circuit synthesis for various universal gate libraries. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2182–2189.

Cirq. 2023. Examples: Basic Arithmetic. https://github.com/quantumlib/Cirq/blob/master/examples/basic_arithmetic.py.

Gavin E. Crooks. 2023. *Gates, States, and Circuits*. Technical report. https://threeplusone.com/gates,https://github.com/gecrooks/on_gates Tech. Note 014 v0.9.0 beta.

Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. 2017. Open quantum assembly language. *arXiv preprint arXiv:1707.03429* (2017).

Marc Grau Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. 2019. Heuristics for Quantum Compiling with a Continuous Gate Set. arXiv:1912.02727 [cs.ET]

Marc G Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. 2020. Towards optimal topology aware quantum circuit synthesis. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 223–234.

Ronald De Wolf. 2017. The potential impact of quantum computers on society. *Ethics and Information Technology* 19 (2017), 271–276.

Olivia Di Matteo and Michele Mosca. 2016. Parallelizing quantum circuit synthesis. *Quantum Science and Technology* 1, 1 (2016), 015003.

Vlad Gheorghiu, Michele Mosca, and Priyanka Mukhopadhyay. 2022a. A (quasi-) polynomial time heuristic algorithm for synthesizing T-depth optimal circuits. *npj Quantum Information* 8, 1 (2022), 1–11.

Vlad Gheorghiu, Michele Mosca, and Priyanka Mukhopadhyay. 2022b. T-count and T-depth of any multi-qubit unitary. *npj Quantum Information* 8, 1 (2022), 141.

Brett Giles and Peter Selinger. 2013. Exact synthesis of multiqubit Clifford+ T circuits. *Physical Review A* 87, 3 (2013), 032332.

Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. 2022. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience* 52, 1 (2022), 66–114.

David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. 2013. An algorithm for the T-count. *arXiv preprint arXiv:1308.4134* (2013).

Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *PLDI'13*. ACM Press, Seattle, Washington, USA. https://doi.org/10.1145/2491956.2462177

Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.

Stefan Heule, Eric Schkufza, Rahul Sharma, and Alex Aiken. 2016. Stratified synthesis: automatically learning the x86-64 instruction set. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 237–250.

Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. 2016. Quantum circuits for isometries. *Physical Review A* 93, 3 (2016), 032318.

Chan Gu Kang and Hakjoo Oh. 2023. Modular Component-Based Quantum Circuit Synthesis. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 348–375.

Sumeet Khatri, Ryan LaRose, Alexander Poremba, Lukasz Cincio, Andrew T. Sornborger, and Patrick J. Coles. 2019. Quantum-assisted quantum compiling. *Quantum* 3 (may 2019), 140. https://doi.org/10.22331/q-2019-05-13-140

Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. 2016. Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and T circuits. *IEEE Trans. Comput.* 65, 1 (Jan. 2016), 161–172. https://doi.org/10.1109/TC.2015.2409842 arXiv:1212.6964 [quant-ph].

Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. 2018. Trading T-gates for dirty qubits in state preparation and unitary synthesis. *arXiv preprint arXiv:1812.00954* (2018).

Dmitri Maslov. 2016. Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *Physical Review A* 93, 2 (2016), 022311.

Richard Meister, Cica Gustiani, and Simon C Benjamin. 2022. Exploring ab initio machine synthesis of quantum circuits. *arXiv preprint arXiv:2206.11245* (2022).

Michele Mosca and Priyanka Mukhopadhyay. 2021. A polynomial time and space heuristic algorithm for T-count. *Quantum Science and Technology* 7, 1 (2021), 015003.

Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

Philipp Niemann, Robert Wille, and Rolf Drechsler. 2020. Advanced exact synthesis of Clifford+ T circuits. *Quantum Information Processing* 19, 9 (2020), 1–23.

OpenMP Architecture Review Board. 2021. OpenMP Application Program Interface Version 5.2. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf

Anouk Paradis, Benjamin Bichsel, Samuel Steffen, and Martin Vechev. 2021. Unqomp: synthesizing uncomputation in Quantum circuits. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. Association for Computing Machinery, New York, NY, USA, 222–236. https://doi.org/10.1145/3453483.3454040

Alex Parent, Martin Roetteler, and Krysta M. Svore. 2015. Reversible circuit compilation with space constraints. https://doi.org/10.48550/arXiv.1510.00377 arXiv:1510.00377 [quant-ph].

Alex Parent, Martin Roetteler, and Krysta M. Svore. 2017. REVS: A Tool for Space-Optimized Reversible Circuit Synthesis. In *Reversible Computation (Lecture Notes in Computer Science)*, Iain Phillips and Hafizur Rahaman (Eds.). Springer International Publishing, Cham, 90–101. https://doi.org/10.1007/978-3-319-59936-6_7

Martin Plesch and Časlav Brukner. 2011. Quantum-state preparation with universal gate decompositions. *Physical Review A* 83, 3 (2011), 032302.

Qiskit. 2023. Implement the multi-controlled X gate using a V-chain of CX gates. https://qiskit.org/documentation/stubs/qiskit.circuit.library.MCXVChain.html.

Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. 2019. ReQWIRE: Reasoning about Reversible Quantum Circuits. *Electronic Proceedings in Theoretical Computer Science* 287 (Jan. 2019), 299–312. https://doi.org/10.4204/EPTCS.287.17 arXiv: 1901.10118.

Neil J. Ross and Peter Selinger. 2016. Optimal ancilla-free Clifford+T approximation of z-rotations. http://arxiv.org/abs/1403.2975 arXiv:1403.2975 [quant-ph].

Eric Schkufza, Rahul Sharma, and Alex Aiken. 2013. Stochastic superoptimization. *ACM SIGARCH Computer Architecture News* 41, 1 (2013), 305–316.

Peter Selinger. 2014. Efficient Clifford+T approximation of single-qubit operators. arXiv:1212.6253 [quant-ph]

Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

Ethan Smith, Marc Grau Davis, Jeffrey Larson, Ed Younis, Lindsay Bassman Oftelie, Wim Lavrijsen, and Costin Iancu. 2023. LEAP: Scaling Numerical Optimization Based Synthesis Using an Incremental Approach. *ACM Transactions on Quantum Computing* 4, 1 (feb 2023), 1–23. https://doi.org/10.1145/3548693

Quantum Computing StackExchange. 2018. How to implement the "Square root of Swap gate" on the IBM Q (composer)? https://quantumcomputing.stackexchange.com/questions/2228/how-to-implement-the-square-root-of-swap-gate-on-the-ibm-q-composer.

Quantum Computing StackExchange. 2020. Decomposition of |110> <-> |000> Exchange Gate. https://quantumcomputing.stackexchange.com/questions/13644/decomposition-of-110-rangle-leftrightarrow-000-rangle-exchange-gate.

Quantum Computing StackExchange. 2021. How to create CX from an entangling gate and arbitrary single-qubit gates? https://quantumcomputing.stackexchange.com/questions/17656/how-to-create-cnot-from-an-entangling-gate-and-arbitrary-single-qubit-gates.

Quantum Computing StackExchange. 2022. Qiskit: How to implement a classical function? https://quantumcomputing.stackexchange.com/questions/26505/qiskit-how-to-implement-a-classical-function.

Quantum Computing StackExchange. 2023. How to decompose root iswap into root cz and single-qubit gates. https://quantumcomputing.stackexchange.com/questions/29617/how-to-decompose-root-iswap-into-root-cz-and-single-qubit-gates.

Matthias Steffen, David P DiVincenzo, Jerry M Chow, Thomas N Theis, and Mark B Ketchen. 2011. Quantum computing: An IBM perspective. *IBM Journal of Research and Development* 55, 5 (2011), 13–1.

Barbara M Terhal. 2015. Quantum error correction for quantum memories. *Reviews of Modern Physics* 87, 2 (2015), 307.

Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. 2021. QFAST: Conflating Search and Numerical Optimization for Scalable Quantum Circuit Synthesis. arXiv:2103.07093 [quant-ph]

A APPENDIX

A.1 Energy Function Derivation

Here, we demonstrate our rewrite of the typical energy function used by e.g. [Chou et al. 2022; Khatri et al. 2019; Meister et al. 2022].

Lemma 1. *Let $U, C \in \mathbb{C}^{2^n \times 2^n}$ be two unitary matrices. Then*

$$1 - \frac{|Tr(U^\dagger C)|}{2^n} = \frac{1}{2} \frac{\left\| U - \frac{Tr(C^\dagger U)}{|Tr(C^\dagger U)|} C \right\|_F^2}{2^n}$$

Proof. We show that

$$2^{n+1} - 2|\mathrm{Tr}(U^\dagger C)| = \left\| U - \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} C \right\|_F^2. \tag{10}$$

The lemma follows by multiplying Eq. (10) with $\frac{1}{2^{n+1}}$.

We first note that for any matrix $A, B \in \mathbb{C}^{2^n \times 2^n}$, the following well-known properties hold:

(1) $||A||_F^2 = \mathrm{Tr}(A^\dagger A)$
(2) $\mathrm{Tr}(A)^\dagger = \mathrm{Tr}(A^\dagger)$
(3) Linearity of the trace, $\mathrm{Tr}(A + \lambda B) = \mathrm{Tr}(A) + \lambda \mathrm{Tr}(B)$
(4) If A is also a unitary matrix, then $||A||_F^2 = 2^n$.

We can use these properties to prove the lemma. Starting from the right-hand side in Eq. (10) we get:

$$\begin{aligned}
&\left\| U - \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} C \right\|_F^2 \\
=&\mathrm{Tr}\left(\left(U - \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} C \right)^\dagger \left(U - \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} C \right) \right) \\
=&\mathrm{Tr}(U^\dagger U) - \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} \mathrm{Tr}(U^\dagger C) - \frac{\mathrm{Tr}(C^\dagger U)^\dagger}{|\mathrm{Tr}(C^\dagger U)|} \mathrm{Tr}(C^\dagger U) + \frac{\mathrm{Tr}(C^\dagger U)^\dagger}{|\mathrm{Tr}(C^\dagger U)|} \frac{\mathrm{Tr}(C^\dagger U)}{|\mathrm{Tr}(C^\dagger U)|} \mathrm{Tr}(C^\dagger C) \\
=&\|U\|_F^2 + \|C\|_F^2 - 2|\mathrm{Tr}(U^\dagger C)| \\
=&2 \cdot 2^n - 2|\mathrm{Tr}(U^\dagger C)|
\end{aligned}$$

which proves the equality. □

A.2 Simplification Order

We define the order $\prec$ that is used in Alg. 2 to determine if two gates should be swapped in a circuit C. The goal of $\prec$ is to ensure that as many gates as possible can be swapped and to define some strict order on the set of gates.

We therefore first define the functions N_{comm}, N_{act} and A. $N_{\text{comm}}(g, i, C)$ is the number of consecutive gates that commute with g starting from g_i and going down until g_1. Mathematically, this means

$$N_{\text{comm}}(g, i, C) = \max \left\{ i - j \middle| 0 \leqslant j \leqslant i \wedge g_i \text{ commutes with } g_j \right\}.$$

$N_{\text{act}}(g)$ is the number of qubits on which gate g acts and $A(g)$ is the indices of the qubits on which g acts. We then define $\prec_{\text{alpabet}}$ as the alphabetical order on the name of the gates (e.g. H, S, CX, ...) and $\prec_n$ as the standard order on $\mathbb{R}^n$. Alg. 3 shows the definition of $\prec$. As shown, the order prioritizes a difference between commutating gates, than acting qubits, than the alphabetical order of the gates and finally the order of the qubits on which the gates act.

Algorithm 3 Definition of the order $\prec$.

function $g_i \prec g_{i+1}$
  if $N_{\text{comm}}(g_i, i, C) \neq N_{\text{comm}}(g_{i+1}, i, C)$ **then**
    return $N_{\text{comm}}(g_i, i, C) < N_{\text{comm}}(g_{i+1}, i, C)$
  if $N_{\text{act}}(g_i) \neq N_{\text{act}}(g_{i+1})$ **then**
    return $N_{\text{act}}(g_i) < N_{\text{act}}(g_{i+1})$
  if $g_i \neq g_{i+1}$ **then**
    return $g_i <_{\text{alphabet}} g_{i+1}$
  if $A(g_i) \leq_n A(g_{i+1})$ **then**
    return `true`
  return `false`

A.3 Baseline for Operators Decomposition

We describe our best effort construction of the baselines for Tab. 4.

[Maslov 2016] gives the best decomposition of RCCCX we could find, with a T-depth of 8.

We were unable to find any explicit decomposition of CCT in published work. We therefore used the construction from [Maslov 2016] and the minimal implementation of the CT gate from [Amy et al. 2013], decomposing $\text{CCT}(a, b, c)$ as $\text{RCCX}(a, b, t); \text{CT}(t, c); \text{RCCX}(a, b, t)$ where t is an ancilla qubit. Altogether, this yields an implementation with T-depth 9 and 2 ancilla qubits, as CT requires its own extra ancilla.

For CCiSWAP, we used the same construction as above, with CiSWAP decomposed as in [Crooks 2023] instead of CT. This yields an implementation with 1 ancilla.

For $\text{C}\sqrt{\text{SWAP}}$ (resp. $\text{C}\sqrt{\text{iSWAP}}$), we could not find a better existing implementation than controlling every gate in the best known decomposition of $\sqrt{\text{SWAP}}$ (resp. $\sqrt{\text{iSWAP}}$). This yields for each of those two gates a decomposition with an ancilla and a T-depth higher than 25.