

Research



Cite this article: Soeken M, Meuli G, Schmitt B, Mozafari F, Riener H, De Micheli G. 2019 Boolean satisfiability in quantum compilation. *Phil. Trans. R. Soc. A* **378**: 20190161. <http://dx.doi.org/10.1098/rsta.2019.0161>

Accepted: 11 October 2019

One contribution of 13 to a theme issue 'Harmonizing energy-autonomous computing and intelligence'.

Subject Areas:

quantum computing, computer-aided design

Keywords:

quantum computing, quantum programming languages, quantum compilation, Boolean satisfiability, logic synthesis

Author for correspondence:

Mathias Soeken
e-mail: mathias.soeken@epfl.ch

Boolean satisfiability in quantum compilation

Mathias Soeken, Giulia Meuli, Bruno Schmitt, Fereshte Mozafari, Heinz Riener and Giovanni De Micheli

EPFL, Lausanne, Switzerland

 GDM, 0000-0002-7827-3215

Quantum compilation is the task of translating a quantum algorithm implemented in a high-level quantum programming language into a technology-dependent instructions flow for a physical quantum computer. To tackle the large gap between the quantum program and the low-level instructions, quantum compilation is split into a multi-stage flow consisting of several layers of abstraction. Several different individual tasks have been proposed for the layers in the flow, many of them are NP-hard. In this article, we will describe the flow and we will propose algorithms based on Boolean satisfiability, which is a good match to tackle such computationally complex problems.

This article is part of the theme issue 'Harmonizing energy-autonomous computing and intelligence'.

1. Introduction

Quantum computers promise to solve tasks such as quantum simulation [1] (e.g. quantum chemistry [2]) and quantum cryptanalysis [3] (e.g. Shor's algorithm [4]) computationally faster than a classical computer. Various practical prototypes have recently been proposed [5–7].

Although quantum computers have not reached the scale to tackle any unsolved open problem yet, researchers have been studying how to efficiently program them. Complete programming flows already exist, which take as input a *quantum algorithm* described in a quantum programming language and compile it into a set of instructions that describe the interactions with the physical device (e.g. [8–10]). This process is called *quantum compilation* and the generated set of instructions is referred to as a *quantum circuit*.

Programming flows allow users to describe their quantum algorithm through the use of symbolical representations of relevant sets of low-level instructions. In addition, they provide compilation methods to generate quantum circuits for each high-level instruction. A good quantum compilation method minimizes the number of resources used to perform the algorithm: number of qubits and number of expensive instructions (requiring complex interactions with the physical device).

Many quantum algorithms require arithmetic operations, e.g. addition, inversion and multiplication, which usually need large amounts of resources to be computed. For this reason, the quality of the compilation method provided by the programming flow is crucial: if this task is not performed efficiently, arithmetic operations will require too many qubits and/or instructions to be performed in practice.

In this work, we focus on a versatile method used to compile arithmetic operations specified as Boolean functions, called *hierarchical quantum compilation* [11–13]. It takes as input a Boolean function, represented by its most compact representation: a logic network. In addition, it allows us to specify the number of available qubits for the computation. With these data, it tries to find a quantum circuit minimizing the number of expensive quantum instructions. Hierarchical quantum compilation consists of three stages. First, the logic network is represented in terms of a k -LUT network (lookup-table network). In the second stage, each of the logic gates are mapped to qubits, which will store the temporary computed value of the gates. Finally, in the third stage, each mapped LUT logic gate is decomposed into quantum gates that are supported by the targeted physical device.

The choice of the targeted quantum hardware has an influence on the quantum compilation algorithms. Current quantum technology will likely not enable reliable fault-tolerant quantum computing within the next few years [14]. This is mainly due to the large amount of noise in the qubits and quantum operations, which requires a significant overhead cost to perform quantum error correction [15]. Until then, researchers investigate which applications can be performed with the quantum computers that will be available in the next few years, having about 50–100 noisy physical qubits unprotected by error correction. Such computers are termed *noisy intermediate-scale quantum computers* (NISQ) [14]. It has already been demonstrated that, despite the noise, practical applications such as quantum chemistry for small problem instances can already be solved on such devices [16]. In this scenario, hierarchical compilation is capable of synthesizing optimal quantum circuits, matching the limited available resources. Eventually, the resulting computation must fit the few qubits available in NISQ computers and perform a limited number of expensive operations, according to the hardware technology.

The advantage of hierarchical quantum compilation is that it is breaking down the complex task of finding a quantum circuit for a large Boolean function into several dedicated combinatorial problems. SAT-based methods are excellent candidates to efficiently tackle these subproblems, as they are able to find results of very good quality, with a positive impact on the entire flow. It is well-known that naive monolithic SAT-based approaches do not scale. Nevertheless, the hierarchical framework provides an excellent application for this class of algorithms since large problems are decomposed into smaller ones. In this paper, we present different SAT-based techniques used to tackle several subproblems in a LUT-based hierarchical quantum compilation flow.

2. Preliminaries

(a) Quantum computing

Quantum computers can be seen as an array of n qubits [17]. A single qubit φ can be in the state $|\varphi\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = a_0|0\rangle + a_1|1\rangle$, where $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ represent the classical states 0 and 1, and a_0 and a_1 are complex-valued amplitudes such that $|a_0|^2 + |a_1|^2 = 1$. In other words, a quantum

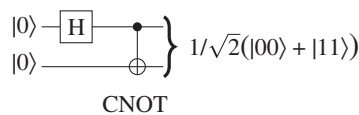
state is a *superposition* of the two classical states. As an example, the state $|+\rangle = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$ is a balanced superposition between 0 and 1. When measuring a single qubit, it collapses to the state 0 with a probability of $|a_0|^2$ and to the state 1 with a probability of $|a_1|^2$. One can also describe the joint state of all n qubits by $\sum_{i=0}^{2^n-1} a_i|i\rangle$, where $|i\rangle$ is one of the 2^n classical states over n bits and $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$. The basic states $|i\rangle$ are one-hot column vectors with 2^n elements which are all 0 except for row i (counting from row 0), which is 1. The probability that the quantum state collapses to the classical state i after measuring all qubits is $|a_i|^2$. Due to the quantum mechanical effect of *entanglement*, some quantum states cannot be described in terms of the qubits' individual quantum states. One example is the Bell state $1/\sqrt{2}(1\ 0\ 0\ 1)^T$, which collapses with the same probability to the states $00_2 = 0_{10}$ and $11_2 = 3_{10}$ after measurement. The measurement result of one qubit reveals the measurement result of the other one.

Quantum computations that change the quantum state of n qubits can be described in terms of $2^n \times 2^n$ unitary matrices. However, it is impractical to represent such unitary matrices explicitly and to physically realize these operations for large n . As an alternative one can describe quantum computations as a composition of small matrices, representing basic instructions, called *quantum gates*. Parallel composition is modelled using the Kronecker product and sequential composition as a matrix product. The Kronecker product between two matrices A of size $m \times n$ and B of size $p \times q$ is defined as:

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}.$$

A simple example for a quantum operation is the NOT gate, also called the X gate, described by the unitary matrix $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. It flips the amplitudes of a single-qubit quantum state, and in particular, $X|0\rangle = |1\rangle$ as well as $X|1\rangle = |0\rangle$. The following example illustrates how to describe quantum computations using small quantum operations in the quantum circuit model (see also figure 1).

Example 2.1. The following circuit shows a quantum computation on two qubits, which are each initialized to the state $|0\rangle$, i.e. they are in the joint state $|00\rangle$.



The horizontal direction denotes time. Each horizontal line corresponds to the life-line of a qubit, representing time, and one can place operations onto them. First, the Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$ is applied to the first qubit. Usually, a quantum gate is depicted as a framed box labelled with the name of the operation. No gate is applied to the second qubit, which implicitly corresponds to applying the identity operation $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. The joint operation to both qubits can be described as the Kronecker product of both matrices $H \otimes I$, which is a 4×4 unitary matrix. Afterwards, the two-qubit CNOT operation is applied to both qubits, where

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The CNOT gate has a special notation where the first qubit applied to the CNOT gate, called the control qubit, is drawn as a solid dot and the second qubit, called the target qubit, is drawn as a '⊕' symbol. A CNOT gate inverts the target qubit whenever the control qubit is assigned 1.

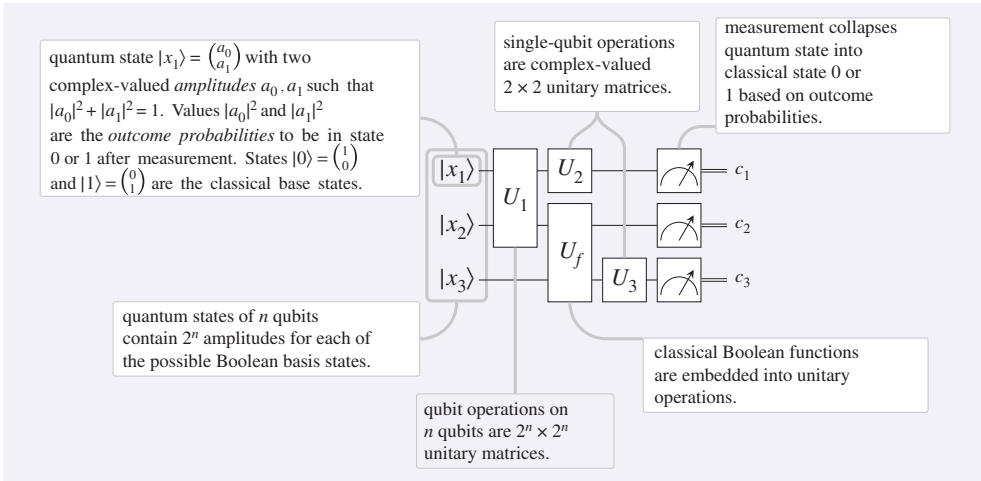


Figure 1. The quantum circuit model in a nutshell.

The overall quantum operation performed by the quantum circuit can be described as the product of all operations performed in each time step, i.e.

$$(\text{CNOT} \cdot (H \otimes I)) |00\rangle = 1/\sqrt{2}(|00\rangle + |11\rangle).$$

There are many quantum gates of interest. An important family of gates consists of the z-rotation gates $R_z(\theta) = \text{diag}(e^{-i\theta}, e^{i\theta})$. The gates $S = e^{i\pi/4}R_z(\pi/4)$ and $T = e^{i\pi/8}R_z(\pi/8)$ are frequently used gates of this family.¹ Similarly, one can define rotation gates R_x and R_y for rotations around the other two axes. A *gate library* is a set of gates which is universal, i.e. for any $2^n \times 2^n$ unitary matrix U there exists a finite sequence of gates to either exactly represent U or to approximate it as arbitrarily precise [17]. We call the gates in a gate library *elementary gates*. Typical gate libraries for current NISQ devices are for example $\{\text{CNOT}, U_3\}$, where U_3 is an arbitrary 2×2 unitary matrix, used, e.g. in IBM's quantum computers, or $\{\text{CZ}, R_z(\theta), R_x(k\pi/2)\}$, where $\text{CZ} = \text{diag}(1, 1, 1, -1)$ is a controlled Z gate, $R_z(\theta)$ is a z-rotation gate with arbitrary precision and $R_x(k\pi/2)$ is an x-rotation gate where the possible rotations are controlled by an integer $k \in \{0, 1, 2, 3\}$, used, e.g. in Rigetti's quantum computers. The commonly used gate library in fault-tolerant quantum computing is Clifford+T which consists of CNOT, H, and the T gate.

All the described gates are primitive operations on modern quantum devices. However, for the modelling and compilation of quantum circuits, more abstract gates are useful, which require further decomposition into elementary quantum gates, such as single-target gates. For some given Boolean *control function* $f(x_1, \dots, x_n)$, a single-target gate computes the unitary operation

$$U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle, \quad (2.1)$$

where $x = x_1 \dots x_n$, which permutes the state of the quantum state according to $f(x)$. The diagrammatic notation for single-target gates is as follows:

$$\begin{array}{c}
 f \\
 \begin{array}{c}
 x_1 \text{---} \square \text{---} x_1 \\
 \vdots \\
 x_n \text{---} \square \text{---} x_n \\
 \vdots \\
 y \text{---} \oplus \text{---} y \oplus f(x_1, \dots, x_n)
 \end{array}
 \end{array} \quad (2.2)$$

The X gate and the CNOT gate are special cases of single-target gates in which $f = 1$ and $f = x_1$, respectively. These gates are also generalized by multiple-controlled Toffoli gates, which are

¹The constant factors $e^{i\pi/4}$ and $e^{i\pi/8}$ are unobservable in quantum systems, but are a technicality to be consistent with the literature.

single-target gates where f can be described in terms of a single product term or $f = 1$ (the empty product term). For more details on quantum computing the reader is referred to the literature (e.g. [17–19]).

(b) Hierarchical quantum compilation

Given an n -input m -output Boolean function $f(x_1, \dots, x_n) = (y_1, \dots, y_m)$ with $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$, the mapping

$$O_f : |x\rangle|y\rangle|0^l\rangle \mapsto |x\rangle|y \oplus f(x)\rangle|0^l\rangle, \quad (2.3)$$

describes a $2^{n+m+l} \times 2^{n+m+l}$ unitary operation, which permutes the amplitudes of a quantum state according to $f(x)$. Such unitary operations are key in quantum algorithms for integer factorization [4] or Grover's search algorithm [20]. The task of quantum compilation for such unitary operations is to find a quantum circuit that realizes the operation without the need to explicitly represent O_f but only based on a symbolic representation of f , e.g. given as a logic network. It can be seen that the n inputs are passed through n qubits $|x\rangle$ that can be in an arbitrary quantum state. The outputs are computed onto m qubits, which can be in an arbitrary state $|y\rangle$. Note that ' \oplus ' refers to the bitwise XOR operation. Finally, compilation can make use of l additional helper qubits, called *ancillae*, which are all in state $|0\rangle$ before and after the computation, but can be used to store intermediate temporary results. Restoring the additional qubits to the $|0\rangle$ state is crucial, since intermediate computation results that remain in the output quantum state are perceived as noise by subsequent operations in the quantum algorithm, eventually leading to wrong results. The number of additional qubits is application- and hardware-specific. As an example, in order to realize an 8-input, 8-output Boolean function in this form on a 20-qubit quantum computer, one can use at most $l = 4$ ancillae. If $m = 1$ and $l = 0$, the unitary operation O_f is a single-target gate.

Hierarchical quantum compilation (also known as LUT-based hierarchical reversible logic synthesis [11]) is a framework to find quantum circuits for O_f when f is represented as a logic network. As illustrated in figure 2, the framework performs the following three steps:

- (i) Decompose f into a k -LUT network using k -LUT mapping (more details in §3a).
- (ii) Assign the logic gates of the k -LUT network to the $n + m + l$ qubits in the resulting quantum circuit, making sure that no more than l temporary values are stored at any time. Key algorithms to ensure this constraint in this step are based on reversible pebbling games [21,22], which will be introduced in §3b. The logic gates are computed in terms of single-target gates.
- (iii) Map each single-target gate into a quantum circuit of elementary gates supported by the targeted quantum device. Two possible algorithms for this step are illustrated in §§3c,d.

(c) Boolean satisfiability

The Boolean satisfiability (SAT) problem asks whether a given n -variable Boolean function f represented in conjunctive normal form (CNF) has a satisfying assignment, i.e. whether there exists an $x \in \mathbb{B}^n$ such that $f(x) = 1$. A CNF is a conjunction of clauses, a clause is a disjunction of literals, and a literal is a variable or its negation.

Example 2.2. The function $f(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ is satisfied when $x_1 = 0$, $x_2 = 0$, and $x_3 = 1$.

The SAT problem is NP-complete [23], yet many powerful SAT solvers [24,25] exist that can efficiently solve several problems of practical interest even when n is large. In order to use SAT solvers for practical applications, the decision problem to be solved must first be expressed in terms of a SAT formula in CNF. Such an encoding is crucial and can have a significant impact on the overall run-time of the SAT solver. In this paper, we illustrate several algorithms which require an encoding of the original domain problem into a SAT formula.

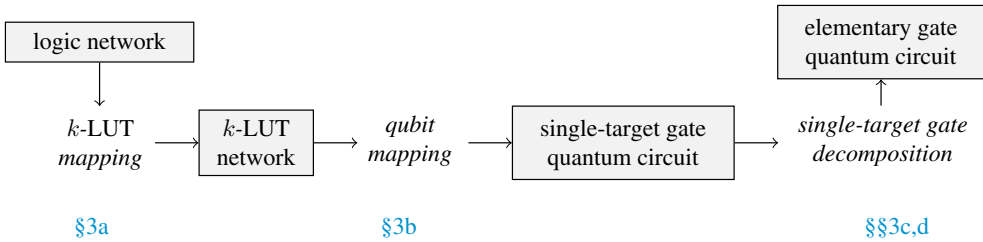


Figure 2. Hierarchical quantum compilation flow.

3. LUT-based hierarchical reversible logic synthesis

(a) LUT mapping

LUT mapping is an algorithm that transforms an arbitrary logic network into a k -LUT network. A necessary condition to find such a mapping is that each gate in the initial logic network has no more than k inputs. A logic network over the n primary inputs x_1, x_2, \dots, x_n is a sequence of r gates $x_{n+1}, x_{n+2}, \dots, x_{n+r}$, where each gate

$$x_i = f_i(x_{i_1} \oplus p_{i_1}, x_{i_2} \oplus p_{i_2}, \dots, x_{i_{k_i}} \oplus p_{i_{k_i}}) \quad \text{for } n < i \leq n + r \quad (3.1)$$

is defined by means of a k_i -input Boolean gate function f_i and fan-ins $x_{i_j} \oplus p_{i_j}$, with $0 \leq i_j < i$, which are either primary inputs, previous gates, or the constant $x_0 = 0$. The constant complementation flags p_{i_j} give the possibility to complement a fan-in. They are part of the gate definition, but not part of the input variables to the gate function. We refer to the set $\{x_0, x_1, \dots, x_{n+r}\}$ as nodes of the logic network. Finally, the multi-output Boolean function represented by the logic network is represented in terms of outputs y_1, \dots, y_m , where

$$y_j = x_{o_j} \oplus p_j, \quad (3.2)$$

with $0 \leq o_j \leq n + r$. The node x_{o_j} is called an *output driver*.

Cut enumeration [26,27] is a key technique used in LUT mapping. Cut enumeration finds for all nodes x_i in a logic network subcircuits, called *cuts*, with no more than k inputs that are rooted in x_i . The inputs of these subcircuits, called *leaves*, separate the root x_i from the primary inputs of the logic network. We refer to the set of all leaves corresponding to a cut to x_i as $CUTS(i)$ in the remainder. A LUT mapping [26,28] assigns some nodes in the logic network to a cut in a way such that

- all primary outputs in the logic network are mapped to a cut, and
- if a node is mapped to a cut, then each leaf of the cut must be mapped unless it is a primary input.

A SAT-based LUT mapping algorithm was first proposed in [29]. In general, in order to describe a problem as a SAT formula, we must find a suitable encoding in terms of Boolean variables in the SAT formula and clauses that restrict the satisfying solutions to solutions of the problem, if they exist. The problem is formulated as a SAT problem with variables m_i for each gate x_i and variables s_C for each cut C from all gates. The variables m_i are 1 if gate x_i has a mapping. The variables s_C are 1 if cut C has been chosen for a mapped gate. Three different types of clauses are needed to constrain the problem such that the solution corresponds to a valid mapping. First, if a gate x_i is mapped, then at least one of its cuts must be chosen:

$$\bar{m}_i \vee \bigvee_{C \in CUTS(i)} s_C. \quad (3.3)$$

Furthermore, if a cut C has been chosen, then each of its leaves must be mapped unless the leaf is a primary input:

$$(\bar{s}_C \vee m_l) \quad \text{for all } l \in C \text{ when } l > n \quad (3.4)$$

Finally, each output driver must be mapped:

$$m_{o_j} \quad \text{for all } 1 \leq j \leq m. \quad (3.5)$$

While these three types of clauses are sufficient in order to find a valid mapping, the resulting mappings may not be of good quality. For example, the SAT solver could simply assign all variables to 1 in order to satisfy all clauses. We need some way of adding a cost function to the SAT formula. While multiple cost functions are possible, we illustrate the idea by finding a mapping that minimizes the size. For this purpose, we further restrict the number of mapped nodes to L by adding the cardinality constraint

$$\sum_{i=n+1}^{n+r} m_i \leq L. \quad (3.6)$$

Cardinality constraints can be translated into CNF in various different ways (e.g. [25,30,31]). In order to find the smallest mapping, one can initialize L with a start value and then decrease it as long as a satisfying solution can be found. Possible start values are the number of gates r or the size of some initial mapping that has been found using a heuristic mapping algorithm. Optimized 6-LUT networks for the EPFL benchmark² obtained using this problem formulation are shown in [29].

Example 3.1. Figure 3a shows a logic network with primary inputs x_1, x_2, x_3 and characterized by a sequence of $r = 3$ gates x_4, x_5, x_6 . The SAT problem declares the variables m_0, m_1, m_2 , one for each gate in the network. If two possible cuts are stored for each gate: $\text{CUTS}(0) = \{c_{00}, c_{01}\}$, $\text{CUTS}(1) = \{c_{10}, c_{11}\}$ and $\text{CUTS}(2) = \{c_{20}, c_{21}\}$. The CNF formulae defining a valid solution with two LUTs are:

$$\begin{aligned} &(\bar{m}_0 \vee s_{c_{00}} \vee s_{c_{01}}) \wedge (\bar{m}_1 \vee s_{c_{10}} \vee s_{c_{11}}) \wedge (\bar{m}_2 \vee s_{c_{20}} \vee s_{c_{21}}) \wedge \\ &\bigwedge_{l \in c_{00}} (\bar{s}_{c_{00}} \vee m_l) \wedge \bigwedge_{l \in c_{01}} (\bar{s}_{c_{01}} \vee m_l) \wedge \cdots \wedge \bigwedge_{l \in c_{21}} (\bar{s}_{c_{21}} \vee m_l) \wedge \\ &(\bar{m}_0 \vee \bar{m}_1 \vee \bar{m}_2), \end{aligned}$$

where the last row defines the cardinality constraint with $L = 2$.

Figure 3 illustrates the quantum compilation process of the 2-LUT network obtained decomposing the 3-input function $f(x_1, x_2, x_3)$, that consists of two subfunctions g and h where $f(x_1, x_2, x_3) = h(g(x_1, x_2), x_3)$. Recall that we use unitary operations U_g and U_h to compute g and h using three qubits, where two qubits hold the input values and a third qubit that is initialized to 0 stores the output result. Figure 3c shows a quantum circuit that computes f as a composition of g and h using five qubits: three data qubits for $|x_1\rangle$, $|x_2\rangle$, and $|x_3\rangle$, and two ancillae, which are initialized to $|0\rangle$. The circuit in figure 3c does not realize a unitary operation such as described in (2.3), as the value for $|g\rangle$ still remains at the end of the computation. Since all unitary operations are reversible, a value can be uncomputed by applying the inverse operation, as long as all original input values for the computation are still available. Figure 3d illustrates how the inverse operation can uncompute the intermediate value for g .

(b) Quantum memory management

In the previous section, we illustrated how decomposed functions can be mapped into quantum circuits using ancillae. For a given Boolean function, there exist several decompositions, and for each decomposition, there exist several mappings, which might require a different number of

²<https://github.com/lsls/benchmarks>.

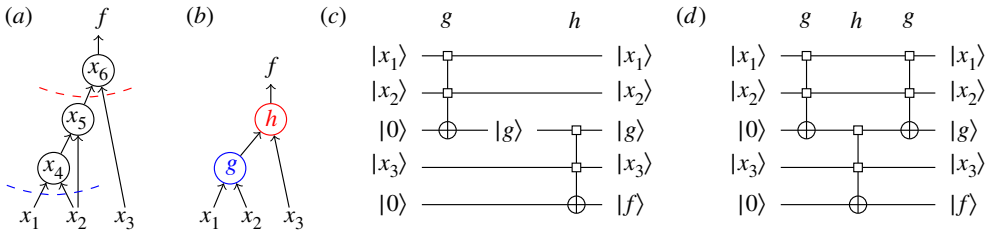


Figure 3. An example of mapping a function f into a quantum circuit; (a) logic network implementing f ; (b) functional decomposition of f as a directed acyclic graph; (c) quantum circuit implementing f that does not uncompute g , thereby leaving an intermediate value that can interfere with quantum states in superposition; (d) does uncompute g by applying it again in reverse order. (Online version in colour.)

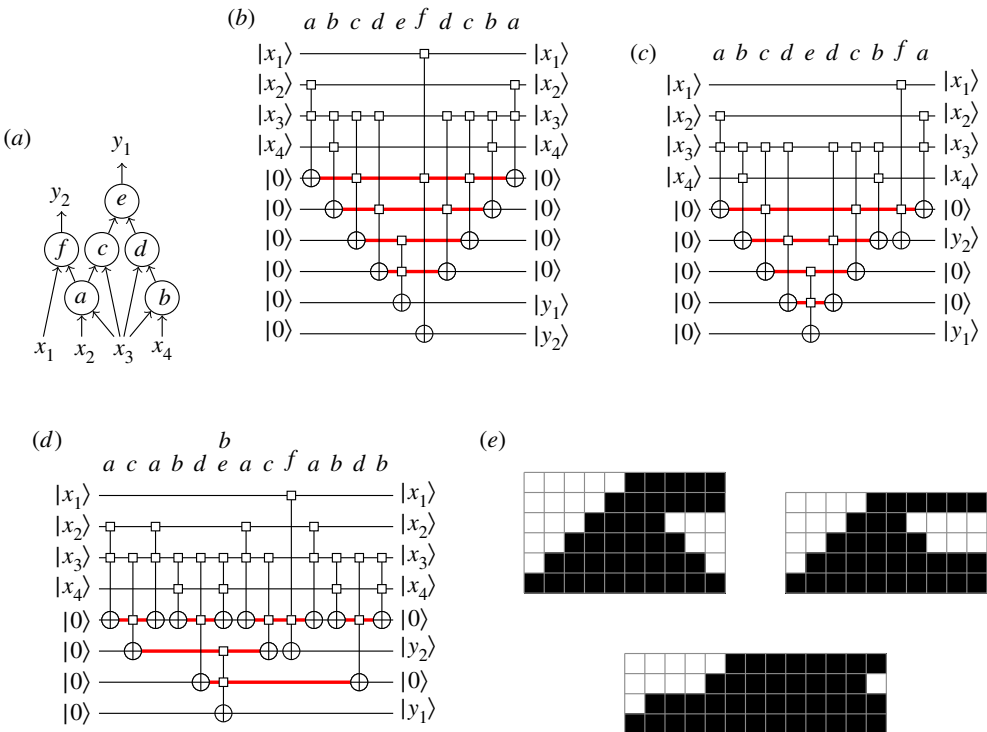


Figure 4. (a) Decomposition graph for 4-input/2-output function; (b) Bennett mapping [32]; (c) eager mapping [11]; (d) mapping using pebbling strategies [33]; (e) resource usage histograms [33]. (Online version in colour.)

ancillae. This motivates the need for effective quantum memory management strategies that map a decomposition into a quantum circuit using a given number of qubits.

Figure 4 illustrates three strategies of how one decomposition can be mapped into a quantum circuit using a different number of ancillae. The graph in figure 4a shows a decomposition of two Boolean functions y_1 and y_2 into six subfunctions a, \dots, f , where the subfunction a is shared by both functions. Figure 4b shows the mapping into a quantum circuit using the Bennett strategy [32], in which first all intermediate and function values are computed, and then all intermediate values are uncomputed. This strategy uses one ancilla for each subfunction, therefore leading to $l = 4$ ancillae and 10 qubits for the overall quantum circuit. We emphasize time spans in which an ancilla carries an intermediate value by drawing it with a thick red

line. As an alternative to the Bennett strategy, one can uncompute intermediate values *eagerly* as illustrated in figure 4c, where intermediate values for b , c and d are not needed after e has been computed. The uncomputation frees up an ancilla that can now be used to compute f , and therefore one less ancilla is needed. Mappings that use fewer qubits can be achieved by permitting to temporarily uncompute intermediate values. Such a strategy is illustrated in figure 4d. However, fewer qubits also result in an increase in computation and uncomputation steps. Consequently, the aim is to find a mapping sequence that leads to the fewest number of computation and uncomputation steps, while maximally using the available qubits. The checkerboards in figure 4e illustrate qubit utilization: each row corresponds to an ancilla, each column corresponds to a time step, and a cell is filled black if the qubit holds a computed value in that time step.

In order to find mapping strategies that lead to the fewest number of computation and uncomputation steps, we exploit an analogy to the *reversible pebble game* [21]. A solution to the reversible pebble game yields a feasible mapping strategy for the quantum memory management problem [33]. The reversible pebble game is played on a directed acyclic graph $G = (V, E)$ with a set of *outputs* $O \subseteq V$. Each state in the game is described in terms of a configuration $C \subseteq V$, which is a subset of vertices that are assigned pebbles. The goal of the game is to find a sequence of configurations $C_1 = \emptyset, C_2, \dots, C_k = O$ such that $v \in (C_i \Delta C_{i+1})$ implies that $w \in (C_i \cap C_{i+1})$ for all $(w, v) \in E$. In other words, a pebble can only be added or removed from a vertex v if all its children w are pebbled. We call such a sequence C_1, \dots, C_k a p -solution if $|C_i| \leq p$ for all i . We derive a sequence of computation and uncomputation steps from a valid p -solution to a pebble game if p ancillae are available. The input graph G is the decomposition graph. Operation j is computed at time step i , whenever $j \in \bar{C}_i \cap C_{i+1}$, and uncomputed at time step i , whenever $j \in C_i \cap \bar{C}_{i+1}$.

The formal description of the reversible pebble game can readily be expressed in terms of a SAT problem which solves the decision problem *Does there exist a p -solution with k steps?*, initialized with some lower bound for k that is incremented until a solution exists. The SAT problem is expressed over variables $v^{(s)}$ for each $v \in V$ and $0 \leq s \leq k$, where $v^{(s)}$ is supposed to be true, if and only if $v \in C_s$, i.e. whether v is pebbled at step k . Initially, no vertex is pebbled and eventually all the output vertices must be pebbled:

$$\bigwedge_{v \in V} \bar{v}^{(0)} \quad \text{and} \quad \bigwedge_{v \in V \setminus O} \bar{v}^{(k)} \wedge \bigwedge_{v \in O} v^{(k)}. \quad (3.7)$$

The following clauses ensure a correct transition from one configuration to the next one:

$$\bigwedge_{v \in V} \left((v^{(s)} \oplus v^{(s+1)}) \Rightarrow \bigwedge_{(w,v) \in E} w^{(s)} \wedge w^{(s+1)} \right) \quad (3.8)$$

for all $0 \leq s < k$. Finally, the cardinality constraint

$$\sum_{v \in V} v^{(s)} \leq p \quad (3.9)$$

for all $1 \leq s \leq k$ ensures that at each step no more than p vertices are pebbled. A comparison between the Bennett strategy and the SAT-based strategy is presented in [33], with a focus on the trade-off between qubits and quantum operations.

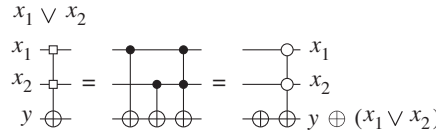
The above approach guarantees termination if there exists a p -solution. Otherwise, the SAT formula is unsatisfiable for all k . In this case, it is difficult to analyse whether the formula is unsatisfiable because k is still too large or whether there exists no solution. The problem lies in the underlying methodology of increasing k until a solution is found, which is also referred to as bounded model checking [34]. Alternative methods with the ability to prove that no solution can exist, are property-directed reachability [35,36] or fixed-point computation [37]. However, these methods cannot guarantee to find a solution with the smallest k .

(c) ESOP-based synthesis

An *exclusive sum-of-products* (ESOP) form is an exclusive sum (using the \oplus) operator of product terms. Each Boolean function can have several different ESOP forms, e.g. one can express $x_1 \vee x_2$ as $x_1 \oplus x_2 \oplus x_1x_2$ or as $1 \oplus \bar{x}_1\bar{x}_2$. ESOP forms are useful in quantum compilation since they allow us to decompose a single-target gate into a sequence of generalized Toffoli gates. More precisely, if f has the ESOP form $c_1 \oplus c_2 \oplus \dots \oplus c_k$, where each c_k is a product term, then

$$U_f = U_{c_1} U_{c_2} \dots U_{c_k}. \quad (3.10)$$

The above-mentioned example for the OR function can therefore be interpreted in the quantum circuit model as follows:



$$x_1 \vee x_2 \quad (3.11)$$

By finding an ESOP form for the control function of a single-target gate, one can use some of the several presented Toffoli gate compilation techniques (e.g. [38–40]). A good heuristic to reduce the compilation cost of a single-target gate using this method is to find a short ESOP with a fewer number of product terms.

A SAT-based algorithm to find an ESOP form for an n -variable Boolean function $f(x_1, \dots, x_n)$ with the minimum number of product terms has been presented in [41] based on ideas in [42]. We follow a similar strategy as in the previous section by solving the decision problem *Does there exist an ESOP form for f with k gates?* starting from some lower bound on k , increasing k until a satisfying solution has been found. The SAT problem is expressed over Boolean variables $p_{j,i}$ and $q_{j,i}$ for $1 \leq i \leq n$ and $1 \leq j \leq k$, where $p_{j,i}$ is true, if and only if x_i is contained in the j th product term and $q_{j,i}$ is true, if and only if \bar{x}_i is contained in the j th product term. The SAT formula further contains variables $z_{j,l}$ for $1 \leq j \leq k$ and $0 \leq l < 2^n$. For all values $l = (b_1 \dots b_n)_2$, the clauses

$$\bigwedge_{j=1}^k \bigwedge_{i=1}^n (\bar{z}_{j,l} \vee (b_i ? \bar{q}_{j,i} : p_{j,i})) \quad \text{and} \quad \bigwedge_{j=1}^k \left(z_{j,l} \vee \bigvee_{i=1}^n (b_i ? q_{j,i} : p_{j,i}) \right) \quad (3.12)$$

ensure that if $z_{j,l} = 1$, then the j th product term evaluates to 1 for assignment b_1, \dots, b_n and if $z_{j,l} = 0$, then the j th product term evaluates to 0 for assignment b_1, \dots, b_n , where the *if-then-else* operator $c ? t : e$ returns t if condition c is true and e otherwise. Note that in (3.12) all conditions of the if-then-else operators are constants. Finally, the XOR constraint

$$\bigoplus_{j=1}^k z_{j,l} = f(b_1, \dots, b_n) \quad (3.13)$$

guarantees that an odd number of $z_{j,l}$ s evaluate to 1 if function f evaluates to 1 for the input assignment b_1, \dots, b_n , and to 0, otherwise. The XOR constraint in (3.13) is the only constraint that involves the input function; it can be translated into clauses by adding auxiliary variables and applying encoding schemes such as the one presented by Tseytin [43]. Results relative to this encoding are reported in [41].

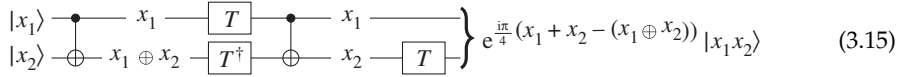
(d) Spectral-based synthesis

An interesting subclass of quantum circuits are those that can be constructed using only CNOT gates and R_z gates. Such circuits are characterized by the *phase polynomial representation* (e.g. [44–46]). A phase polynomial representation for a unitary $2^n \times 2^n$ matrix U over n qubits is a tuple $(A, (\theta_1, f_1), \dots, (\theta_k, f_k))$, where $A \in \text{GL}_n(\mathbb{B})$ is a linear Boolean matrix, θ_i are real-valued angles, and

f_i are linear Boolean functions over n variables. A phase polynomial representation describes the matrix

$$U : |x\rangle \mapsto \left(\prod_{i=1}^k e^{i\theta_i f_i(x)} \right) |Ax\rangle. \quad (3.14)$$

Conversely, it can be shown that matrices obtained from a phase polynomial representation can always be represented by a quantum circuit using only CNOT and R_z gates. A simple way to get familiar with the phase polynomial representation is to read a phase polynomial representation from a quantum circuit.



$$\left. \begin{array}{c} |x_1\rangle \text{---} \bullet \text{---} x_1 \text{---} \boxed{T} \text{---} \bullet \text{---} x_1 \text{---} \\ |x_2\rangle \text{---} \oplus \text{---} x_1 \oplus x_2 \text{---} \boxed{T^\dagger} \text{---} \oplus \text{---} x_2 \text{---} \boxed{T} \end{array} \right\} e^{\frac{i\pi}{4}(x_1 + x_2 - (x_1 \oplus x_2))} |x_1x_2\rangle \quad (3.15)$$

The phase polynomial representation for this circuit is

$$\left(I, \left(\frac{\pi}{4}, x_1 \right), \left(\frac{\pi}{4}, x_2 \right), \left(-\frac{\pi}{4}, x_1 \oplus x_2 \right) \right), \quad (3.16)$$

where I is the 2×2 identity matrix. This linear matrix can be obtained by considering the action on the qubits performed exclusively by all CNOT gates.

By observing that $x_1 \oplus x_2 = x_1 + x_2 - 2x_1x_2$, the operation performed by the quantum circuit in (3.15) is $e^{(i\pi/2)x_1x_2} |x_1x_2\rangle$, which is the controlled-S gate [17].

The controlled-controlled-Z operation, or CCZ in short, performs the action

$$\text{CCZ} |x_1x_2x_3\rangle = e^{i\pi x_1x_2x_3} |x_1x_2x_3\rangle \quad (3.17)$$

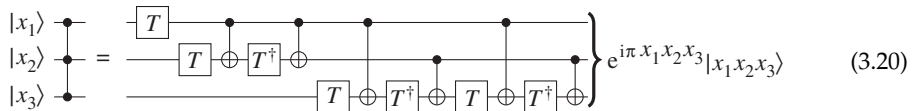
and is represented by the unitary matrix $\text{diag}(1, 1, 1, 1, 1, 1, -1)$. Since

$$4x_1x_2x_3 = x_1 + x_2 + x_3 - (x_1 \oplus x_2) - (x_1 \oplus x_3) - (x_2 \oplus x_3) + (x_1 \oplus x_2 \oplus x_3), \quad (3.18)$$

one can derive that a phase polynomial representation for the CCZ gate is [44]

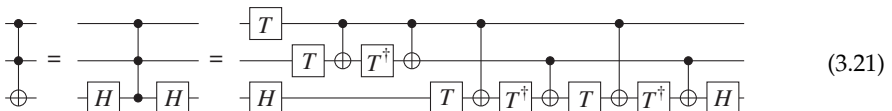
$$\left(I, \left(\frac{\pi}{4}, x_1 \right), \left(\frac{\pi}{4}, x_2 \right), \left(\frac{\pi}{4}, x_3 \right), \left(-\frac{\pi}{4}, x_1 \oplus x_2 \right), \left(-\frac{\pi}{4}, x_1 \oplus x_3 \right), \right. \\ \left. \times \left(\frac{\pi}{4}, x_2 \oplus x_3 \right), \left(\frac{\pi}{4}, x_1 \oplus x_2 \oplus x_3 \right) \right). \quad (3.19)$$

There are many possible quantum circuits that represent this phase polynomial representation, one example is



$$\left. \begin{array}{c} |x_1\rangle \text{---} \bullet \text{---} \boxed{T} \text{---} \bullet \text{---} \\ |x_2\rangle \text{---} \bullet \text{---} \boxed{T} \text{---} \oplus \text{---} \boxed{T^\dagger} \text{---} \oplus \text{---} \\ |x_3\rangle \text{---} \bullet \text{---} \boxed{T} \text{---} \oplus \text{---} \boxed{T^\dagger} \text{---} \oplus \text{---} \boxed{T} \text{---} \oplus \text{---} \boxed{T^\dagger} \end{array} \right\} e^{i\pi x_1x_2x_3} |x_1x_2x_3\rangle \quad (3.20)$$

The T and T^\dagger gates from left to right are applied to the polynomials x_1 , x_2 , $x_1 \oplus x_2$, x_3 , $x_1 \oplus x_3$, $x_1 \oplus x_2 \oplus x_3$ and $x_2 \oplus x_3$. Since $HZH = X$, the circuit construction can be used to find a quantum circuit over the Clifford+ T gate basis:



$$\left. \begin{array}{c} \bullet \text{---} \boxed{T} \text{---} \bullet \text{---} \\ \bullet \text{---} \boxed{T} \text{---} \oplus \text{---} \boxed{T^\dagger} \text{---} \oplus \text{---} \\ \oplus \text{---} \boxed{H} \text{---} \bullet \text{---} \boxed{H} \text{---} \boxed{H} \text{---} \boxed{T} \text{---} \oplus \text{---} \boxed{T^\dagger} \text{---} \oplus \text{---} \boxed{T} \text{---} \oplus \text{---} \boxed{T^\dagger} \text{---} \oplus \text{---} \boxed{H} \end{array} \right\} \quad (3.21)$$

In [47], the authors showed how to extract and realize phase polynomial representations for single-target gates based on the work presented in [48,49].

We next describe a SAT-based algorithm that finds a circuit with the smallest number of CNOT gates to realize a given phase polynomial representation $(A, (\theta_1, f_1), \dots, (\theta_k, f_k))$ over n qubits [50]. The algorithm solves decision problems of the form *Does there exist a CNOT+ R_z circuit with S CNOT gates?* starting from some lower bound on S , then incrementing S until a satisfying solution

is found. The SAT formula is expressed over variables $a_{i,j}^{(s)}$ with $1 \leq i, j \leq n$ and $0 \leq s \leq S$ that represent the linear operation of the quantum circuit after the s th CNOT gate. Furthermore, variables $c_i^{(s)}$ and $t_i^{(s)}$ for $1 \leq i \leq n$ and $1 \leq s \leq S$ are used to represent the position of the control and target line for the s th CNOT gate. Finally, variables $p_{i,l}^{(s)}$ for $1 \leq i \leq n$, $1 \leq l \leq k$, and $1 \leq s \leq S$ indicate whether the linear function on qubit i at step s matches the parity term f_l in the given phase polynomial representation; and variables $q_l^{(s)}$ for $1 \leq l \leq k$ and $1 \leq s \leq S$ indicate whether a parity term has been realized on any qubit at step s or before.

The following constraints need to be satisfied in order to find a valid solution. First, the initial linear transformation at step 0 is the identity transformation and the final linear transformation at step S should match A , i.e.

$$\bigwedge_{1 \leq i, j \leq n} [a_{i,j}^{(0)} = \delta_{ij}] \quad \text{and} \quad \bigwedge_{1 \leq i, j \leq n} [a_{i,j}^{(S)} = A_{i,j}] \quad (3.22)$$

where δ_{ij} is the delta function $\delta_{ij} = [i = j]$. Next, there can be exactly one control and one target at each step, and they must be on different qubits:

$$\bigwedge_{1 \leq i < j \leq n} [(\bar{c}_i^{(s)} \vee \bar{c}_j^{(s)})(\bar{t}_i^{(s)} \vee \bar{t}_j^{(s)})] \wedge (c_1^{(s)} \vee \dots \vee c_n^{(s)}) \wedge (t_1^{(s)} \vee \dots \vee t_n^{(s)}) \times \bigwedge_{1 \leq i \leq n} (\bar{c}_i^{(s)} \vee \bar{t}_i^{(s)}) \quad (3.23)$$

for all $1 \leq s \leq S$. More clauses ensure that the linear transformation at step s changes from the transformation at step $s - 1$ according to the CNOT gate at step s . First, no change takes place if the target is not on some qubit, i.e.

$$\bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq n} (\bar{t}_i^{(s)} \Rightarrow (a_{i,j}^{(s-1)} = a_{i,j}^{(s)})) = \bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq n} (t_i^{(s)} \vee \bar{a}_{i,j}^{(s-1)} \vee a_{i,j}^{(s)})(\bar{t}_i^{(s)} \vee a_{i,j}^{(s-1)} \vee \bar{a}_{i,j}^{(s)}), \quad (3.24)$$

for all $1 \leq s \leq S$. Second, the row of the control line needs to be added to the target line, i.e.

$$\begin{aligned} & \bigwedge_{\substack{1 \leq i, i' \leq n \\ i \neq i'}} \bigwedge_{1 \leq j \leq n} ((c_i^{(s)} \wedge t_{i'}^{(s)}) \Rightarrow (a_{i',j}^{(s)} = a_{i',j}^{(s-1)} \oplus a_{i,j}^{(s-1)})) \\ &= \bigwedge_{\substack{1 \leq i, i' \leq n \\ i \neq i'}} \bigwedge_{1 \leq j \leq n} (\bar{c}_i^{(s)} \vee \bar{t}_{i'}^{(s)} \vee \bar{a}_{i',j}^{(s)} \vee \bar{a}_{i',j}^{(s-1)} \vee \bar{a}_{i,j}^{(s-1)})(\bar{c}_i^{(s)} \vee \bar{t}_{i'}^{(s)} \vee \bar{a}_{i',j}^{(s)} \vee a_{i',j}^{(s-1)} \vee a_{i,j}^{(s-1)}) \\ & \times (\bar{c}_i^{(s)} \vee \bar{t}_{i'}^{(s)} \vee a_{i',j}^{(s)} \vee \bar{a}_{i',j}^{(s-1)} \vee a_{i,j}^{(s-1)})(\bar{c}_i^{(s)} \vee \bar{t}_{i'}^{(s)} \vee a_{i',j}^{(s)} \vee a_{i',j}^{(s-1)} \vee \bar{a}_{i,j}^{(s-1)}), \end{aligned} \quad (3.25)$$

for all $1 \leq s \leq S$.

The clauses so far guarantee that the linear function of the circuit according to the CNOT gates is computed correctly. Finally, we need to ensure that all parity terms f_l are realized at some step by some qubit. We do this in two steps. First, variables $p_{i,l}^{(s)}$ are true whenever the linear transformation after step s on qubit i matches the parity term f_l , ensured by

$$\bigwedge_{1 \leq i \leq n} \left(p_{i,l}^{(s)} \Rightarrow \bigwedge_{1 \leq j \leq n} (a_{i,j}^{(s)} = [x_j \in f_l]) \right) = \bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq n} \begin{cases} (\bar{p}_{i,l}^{(s)} \vee a_{i,j}^{(s)}) & \text{if } x_j \in f_l \\ (\bar{p}_{i,l}^{(s)} \vee \bar{a}_{i,j}^{(s)}) & \text{if } x_j \notin f_l \end{cases} \quad (3.26)$$

for all $1 \leq s \leq S$. Note that $x_j \in f_l$ means that x_j is in the parity term f_l . Finally, we use variables $q_l^{(s)}$ to assert that parity term l was realized on some qubit at step s or before, and then assert that all parity terms must have been realized at the final step S :

$$q_l^{(s)} = \bigvee_{1 \leq i \leq n} p_{i,l}^{(s)} \vee q_l^{(s-1)} \quad \text{for } 1 \leq s \leq S, \quad \text{and} \quad \bigwedge_{1 \leq l \leq k} q_l^{(S)}. \quad (3.27)$$

Circuits obtained solving the problem as encoded in this section can be found in [50].

4. Conclusion

In this work, we described a hierarchical compilation flow based on LUT-networks that is particularly efficient when the quantum system is constrained. We proposed different SAT-based algorithms that are used to find an optimal solution to several critical subproblems of the compilation flow. Because SAT methods are not scalable, they find a suited application in the hierarchical quantum compilation flow where complex problems are decomposed into smaller ones. Nevertheless, they can also be applied to large problems when combined with heuristics.

We provide open source implementations for all mentioned algorithms in the EPFL logic synthesis libraries [51], namely *tweedledum*,³ *caterpillar*,⁴ *easy*⁵ and *mockturtle*.⁶

The algorithms are able to solve several interesting problems. Nonetheless, there are open problems which still seek solutions. For SAT-based k -LUT mapping, it is interesting to investigate how associating a cost function with each LUT may improve the final result. For example, it is possible to consider the cost of each LUT after being decomposed into elementary quantum gates, as experimented in a heuristic solution in [52]. Similarly, it would be interesting to consider resource-constrained reversible pebble games, which would enable to prefer solutions where expensive computations are less often computed and uncomputed. In addition, the relation between ESOP minimality and actual implementation cost is worth being investigated, as, e.g. done in [53] by proposing a cost-minimal ESOP synthesis. Finally, to better address the hardware capabilities for the quantum devices, it is desirable to consider layout constraints such as qubit coupling constraints in the decompositions, as, e.g. done in [54,55].

Data accessibility. Github repositories to the software packages are mentioned in the paper.

Authors' contributions. All authors contributed equally to the paper.

Competing interests. We declare we have no competing interests.

Funding. This research was supported by the European Research Council in the project H2020-ERC-2014-ADG 669354 CyberCare and by the Swiss National Science Foundation (200021-169084 MAJesty).

References

1. Feynman RP. 1982 Simulating physics with computers. *Int. J. Theor. Phys.* **21**, 467–488. (doi:10.1007/BF02650179)
2. Low GH *et al.* 2019 Q# and NWChem: tools for scalable quantum chemistry on quantum computers. (<http://arxiv.org/1904.01131>)
3. Roetteler M, Naehrig M, Svore KM, Lauter K. 2017 Quantum resource estimates for computing elliptic curve discrete logarithms. In *Int. Conf. on the Theory and Applications of Cryptology and Information Security, Advances in Cryptology - ASIACRYPT 2017*, pp. 241–270.
4. Shor PW. 1997 Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**, 1484–1509. (doi:10.1137/S0097539795293172)
5. Castelvechi D. 2017 Quantum computers ready to leap out of the lab in 2017. *Nature* **541**, 9–10. (doi:10.1038/541009a)
6. IBM. 2017 *IBM builds its most powerful universal quantum computing processors*. Press release, 17 May 2017.
7. Intel. 2017 *Intel delivers 17-qubit superconducting chip with advanced packaging to QuTech*. Press release, 10 October 2017.
8. Soeken M, Häner T, Roetteler M. 2018 Programming quantum computers using design automation. *Design, automation and test in Europe* **2018**, 137–146. (doi:10.23919/DATE.2018.8341993)

³github.com/boschmitt/tweedledum.

⁴github.com/gmeuli/caterpillar.

⁵github.com/hriener/easy.

⁶github.com/lsils/mockturtle.

9. Häner T, Steiger DS, Svore K, Troyer M. 2016 A software methodology for compiling quantum programs. (<http://arxiv.org/1604.01401>)
10. Chong FT, Franklin D, Martonosi M. 2017 Programming languages and compiler design for realistic quantum hardware. *Nature* **549**, 180–187. (doi:10.1038/nature23459)
11. Soeken M, Roetteler M, Wiebe N, De Micheli G. In press. LUT-based hierarchical reversible logic synthesis. *IEEE Trans. on CAD of Integrated Circuits and Systems*.
12. Markov IL, Saeedi M. 2012 Constant-optimized quantum circuits for modular multiplication and exponentiation. *Quantum Inf. Comput.* **12**, 361–394.
13. Markov IL, Saeedi M. 2013 Faster quantum number factoring via circuit synthesis. *Phys. Rev. A* **87**, 012310. (doi:10.1103/PhysRevA.87.012310)
14. Preskill J. 2018 Quantum computing in the NISQ era and beyond. *Quantum* **2**, 79. (doi:10.22331/q-2018-08-06-79)
15. Gottesman D. 2010 An introduction to quantum error correction and fault-tolerant quantum computation. In *Symposia in applied mathematics*, vol. 68, pp. 13–58. (<http://arxiv.org/0904.2557>)
16. Kandala A, Mezzacapo A, Temme K, Takita M, Brink M, Chow JM, Gambetta JM. 2017 Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* **549**, 242–246. (doi:10.1038/nature23879)
17. Nielsen MA, Chuang IL. 2000 *Quantum computation and quantum information*. Cambridge, UK: Cambridge University Press.
18. Kitaev AY, Shen AH, Vyalyi MN. 2002 *Classical and quantum computation*, vol. 47. Graduate Series in Mathematics. American Mathematical Society.
19. Saeedi M, Markov IL. 2013 Synthesis and optimization of reversible circuits - a survey. *ACM Comput. Surv.* **45**, 21. (doi:10.1145/2431211.2431220)
20. Grover LK. 1996 A fast quantum mechanical algorithm for database search. In *STOC 1996 Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing, Philadelphia, PA, 22–24 May*, pp. 212–219.
21. Bennett CH. 1989 Time/space trade-offs for reversible computation. *SIAM J. Comput.* **18**, 766–776. (doi:10.1137/0218053)
22. Kráľovic R. 2001 Time and space complexity of reversible pebbling. In *Conf. on Current Trends in Theory and Practice of Informatics, Piestany, Slovakia, 24 November–1 December 2001*, pp. 292–303.
23. Cook SA. 1971 The complexity of theorem-proving procedures. In *STOC 1971 Proceedings of the third annual ACM symposium on Theory of computing, Shaker Heights, OH, 3–5 May*, pp. 151–158.
24. Biere A, Heule M, van Maaren H, Walsh T (eds). 2009 *Handbook of satisfiability*. Amsterdam, the Netherlands: IOS Press.
25. Knuth DE. 2015 *The art of computer programming, volume 4, fascicle 6: satisfiability*. Reading, MA: Addison-Wesley.
26. Cong J, Ding Y. 1994 On area/depth trade-off in LUT-based FPGA technology mapping. *IEEE Trans. VLSI Syst.* **2**, 137–148. (doi:10.1109/92.285741)
27. Cong J, Wu C, Ding Y. 1999 Cut ranking and pruning: enabling a general and efficient FPGA mapping solution. In *FPGA 1999 Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays, Monterey, CA, 21–23 February*, pp. 29–35.
28. Mishchenko A, Chatterjee S, Brayton RK. 2007 Improvements to technology mapping for LUT-based FPGAs. *IEEE Trans. CAD Integr. Circuits Syst.* **26**, 240–253. (doi:10.1109/TCAD.2006.887925)
29. Schmitt B, Mishchenko A, Brayton RK. 2018 SAT-based area recovery in structural technology mapping. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, South Korea, 22–25 January*, pp. 586–591.
30. Bailleux O, Bouffkhad Y. 2003 Efficient CNF encoding of Boolean cardinality constraints. In: *Principles and Practice of Constraint Programming – CP 2003* (ed. F Rossi), pp. 108–122. Lecture Notes in Computer Science, vol. 2833. Berlin, Heidelberg, Germany: Springer.
31. Sinz C. 2005 Towards an optimal CNF encoding of Boolean cardinality constraints. In: *Principles and Practice of Constraint Programming – CP 2003* (ed. F Rossi), pp. 827–831. Lecture Notes in Computer Science, vol. 2833. Berlin, Heidelberg, Germany: Springer.
32. Bennett CH. 1973 Logical reversibility of computation. *IBM J. Res. Dev.* **17**, 525–532. (doi:10.1147/rd.176.0525)

33. Meuli G, Soeken M, Roetteler M, Björner N, De Micheli G. 2019 Reversible pebbling game for quantum memory management. In *Proc. Design, Automation and Test in Europe*, pp. 288–291.
34. Biere A, Cimatti A, Clarke EM, Zhu Y. 1999 Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems. TACAS 1999* (ed. WR Cleaveland), pp. 193–207. Lecture Notes in Computer Science, vol. 1579. Berlin, Heidelberg, Germany: Springer.
35. Bradley AR. 2011 SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation. VMCAI 2011* (eds R Jhala, D Schmidt), pp. 70–87. Lecture Notes in Computer Science, vol. 6538. Berlin, Heidelberg, Germany: Springer.
36. Eén N, Mishchenko A, Brayton RK. 2011 Efficient implementation of property directed reachability. In *FMCAD 2011 Proceedings of the International Conference on Formal Methods in Computer-Aided Design, Austin, TX, 30 October–2 November*, pp. 125–134.
37. Hoder K, Björner N, de Moura LM. 2011 μZ - an efficient engine for fixed points with constraints. In *Computer Aided Verification. CAV 2011* (eds G Gopalakrishnan, S Qadeer), pp. 457–462. Lecture Notes in Computer Science, vol. 6806. Berlin, Heidelberg, Germany: Springer.
38. Barenco A, Bennett CH, Cleve R, DiVincenzo DP, Margolus N, Shor P, Sleator T, Smolin JA, Weinfurter H. 1995 Elementary gates for quantum computation. *Phys. Rev. A* **52**, 3457. (doi:10.1103/PhysRevA.52.3457)
39. Maslov D. 2016 Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *Phys. Rev. A* **93**, 022311. (doi:10.1103/PhysRevA.93.022311)
40. Abdessaied N, Amy M, Drechsler R, Soeken M. 2016 Complexity of reversible circuits and their quantum implementations. *Theor. Comput. Sci.* **618**, 85–106. (doi:10.1016/j.tcs.2016.01.011)
41. Riener H, Ehlers R, Schmitt B, Micheli GD. 2020 Exact synthesis of ESOP forms. In *Advanced boolean techniques* (eds R Drechsler, M Soeken). Berlin, Germany: Springer. (<http://arxiv.org/1807.11103>)
42. Kamath AP, Karmarkar N, Ramakrishnan KG, Resende MGC. 1992 A continuous approach to inductive inference. *Math. Program.* **57**, 215–238. (doi:10.1007/BF01581082)
43. Tseytin GS. 1970 On the complexity of derivation in propositional calculus. In *Studies in constructive mathematics and mathematical logic, part II, seminars in mathematics* (ed. AP Slisenko), pp. 115–125. Berlin, Germany: Springer.
44. Selinger P. 2013 Quantum circuits of T -depth one. *Phys. Rev. A* **87**, 042302. (doi:10.1103/PhysRevA.87.042302)
45. Amy M, Maslov D, Mosca M, Roetteler M. 2013 A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. CAD Integr. Circuits Syst.* **32**, 818–830. (doi:10.1109/TCAD.2013.2244643)
46. Amy M, Maslov D, Mosca M. 2014 Polynomial-time T -depth optimization of Clifford+ T circuits via matroid partitioning. *IEEE Trans. CAD Integr. Circuits Syst.* **33**, 1476–1489. (doi:10.1109/TCAD.2014.2341953)
47. Soeken M, Mozafari F, Schmitt B, De Micheli G. 2019 Compiling permutations for superconducting QPUs. In *2019 Design, Automation & Test in Europe Conference & Exhibition, Florence, Italy, 25–29 May*, pp. 1349–1354.
48. Welch J, Greenbaum D, Mostame S, Aspuru-Guzik A. 2014 Efficient quantum circuits for diagonal unitaries without ancillas. *New J. Phys.* **16**, 1–15. (doi:10.1088/1367-2630/16/3/033040)
49. Schuch N, Siewert J. 2003 Programmable networks for quantum algorithms. *Phys. Rev. Lett.* **91**, 027902. (doi:10.1103/PhysRevLett.91.027902)
50. Meuli G, Soeken M, De Micheli G. 2018 SAT-based {CNOT, T} quantum circuit synthesis. In *Reversible Computation. RC 2018* (eds J Kari, I Ulidowski), pp. 175–188. Lecture Notes in Computer Science, vol. 11106. Cham: Springer.
51. Soeken M, Riener H, Haaswijk W, De Micheli G. 2018 The EPFL logic synthesis libraries. (<http://arxiv.org/1805.05121>)
52. Meuli G, Soeken M, Roetteler M, De Micheli G. 2019 ROS: resource-constrained oracle synthesis for quantum computers. In *Quantum physics and logic*. (<http://qpl2019.org/ros-resource-constrained-oracle-synthesis-for-quantum-computers/>)

53. Meuli G, Schmitt B, Ehlers R, Riener H, De Micheli G. 2019 Evaluating ESOP optimization methods in quantum compilation flows. In *Reversible Computation. RC 2019* (eds M Thomsen, M Soeken). Lecture Notes in Computer Science, vol. 11497. Cham: Springer.
54. Nash B, Gheorghiu V, Mosca M. 2019 Quantum circuit optimizations for NISQ architectures. (<http://arxiv.org/1904.01972>)
55. Kissinger A, Meijer-van de Griend A. 2019 CNOT circuit extraction for topologically-constrained quantum memories. (<http://arxiv.org/1904.00633>)