Journal of Electronic Testing (2022) 38:39–62
https://doi.org/10.1007/s10836-022-05987-z

Synthesis of Reversible Circuits with Reduced Nearest-Neighbor Cost Using Kronecker Functional Decision Diagrams

Dengli Bu[1,2] · Junjie Yan[1] · Pengjie Tang[2] · Haohao Yuan[1]

Received: 13 October 2021 / Accepted: 16 February 2022 / Published online: 25 March 2022

Abstract
Motivated by the importance of fault tolerance in quantum computing, there has been renewed interest in quantum circuits that are realized with Clifford+T gates. Quantum computers that are based on ion-trap technology, superconducting, and quantum dots need to fulfill certain nearest-neighbor (NN) constraints. Fault-tolerant implementations of quantum circuits also require restricted interactions among neighboring quantum bits. The insertion of SWAP-gates is often deployed to make quantum circuits nearest-neighbor (NN) compliant. As quantum operations are prone to various errors, it is important to reduce the nearest-neighbor cost (NNC) which is a marker to the number of SWAP-gates needed to make a quantum circuit NN-compliant. Such an optimization problem arises while synthesizing reversible circuits using the Kronecker functional decision diagram (KFDD). In this work, we propose a method based on KFDD that reduces NNC during synthesis. Considering the Clifford+T quantum mapping for NOT, CNOT, and Toffoli (NCT) gates, and mixed-polarity Peres (MPP) gates, NNC metrics are defined for reversible circuits. Governed by NNC metrics, the nodes are then ranked for reducing NNC in resulting reversible circuits. Furthermore, local transformations are applied on node functions while mapping a node to a cascade of reversible gates. Experimental results on several benchmark functions reveal that the proposed synthesis technique reduces NNC in many cases while slightly impacting the number of qubits, T-depth, and T-count. Compared to prior methods based on functional decision diagrams or binary decision diagrams, the proposed synthesis technique reduces quantum cost for NCV-realizations (i.e., with NOT, CNOT, V, and $V^{\dagger}$ gates) in most of the cases.

Keywords Quantum computing · Fault tolerance · Nearest neighbor cost · Reversible circuit · Logic synthesis · Kronecker functional decision diagram

1 Introduction

With the shrinking of transistor size according to Moore's law, the ongoing miniaturization of integrated circuits will reach soon its limits [2, 5, 28]. Shrinking transistor sizes has become one of the major barriers in the development of circuits to provide an exponential increase of computing power [5, 28]. As a new computation paradigm, quantum computing, which performs computation using properties of quantum mechanics and processes information in terms of quantum bits (or, for short, qubits) instead of just classical bits, provides a promising alternative to further satisfy the needs for more computational power [2, 5, 16]. It has been shown that a quantum computer could efficiently solve certain problems (e.g., database search, integer factorization, graph problems) which have no efficient solution on a classical computer [2, 5, 16, 18, 26].

In recent years, the physical realizations of quantum computers have received significant attention [4]. Companies, such as IBM, Intel, Google, and Microsoft had started developments toward the realization of actual quantum computers for practical purposes [4]. Moreover, IBM, Intel, and Google have all announced their quantum devices with around 50-70 qubits [12]. Motivated by this, the synthesis of quantum circuits has become an active research area [2, 4, 5, 9, 12, 18]. As quantum logic synthesis is a complex and challenging

Responsible Editor: B. B. Bhattacharya

✉ Dengli Bu
dengli.bu@vip.163.com

[1] School of Electrical, Electronic and Computer Science, Guangxi University of Science and Technology, Liuzhou 545006, Guangxi, China

[2] School of Electronics and Information Engineering, Jinggangshan University, Ji'an 343009, Jiangxi, China

problem, Boolean functions, which constitute a major component in many quantum algorithms (e.g., the Oracle function in the Deutsch Algorithm or Grover's database search as well as the modular exponentiation in Shor's factorization algorithm), are usually treated separately using a two-step approach [2, 17, 18, 26]. First, a reversible circuit is designed for the desired Boolean function using established reversible gate libraries [23, 29]. Then, the resulting reversible circuit is mapped into a functionally equivalent quantum circuit by decomposing reversible gates into elementary quantum gates [4, 17, 18]. Accordingly, how to efficiently realize reversible circuits has received significant interest [4, 29].

There have been several functional and structural synthesis methods for reversible circuits proposed [29]. Such as transformation based synthesis methods [15], the positive-polarity Reed-Muller expansion based synthesis method [8], the one-pass synthesis method using a quantum multiple-valued decision diagram to represent the function matrix for improving the limited scalability of functional synthesis methods [29], exclusive-sum-of-products expansion based synthesis methods [7], as well as hierarchical synthesis methods including lookup-table networks based [21] or decision diagrams (DD) based synthesis methods [1, 3, 23, 24]. Although functional methods outperform others in terms of the cost of the synthesized circuits, they are limited to small functions [1, 29]. Consequently, structural or hierarchical synthesis methods which can offer satisfactory scalability have gained more attractions.

DD-based methods are intended for the synthesis of functions with a large number of variables [23, 24]. Compared to other structural methods, although DD-based methods incurs a large number of ancilla qubits, they can achieve low quantum cost, and thus can reduce the cost that makes the quantum realizations fault tolerant [14, 21]. Furthermore, for reducing the number of qubits required for the reversible circuits generated from DDs, techniques including using Davio decompositions [22], applying local transformations on the function represented by a node [3], as well as sorting the ordering of nodes to be mapped by using dependency matrices to express dependencies between nodes [23] or by using genetic algorithm [1] had been proposed.

The decoherence of quantum states while quantum systems interact with environment will result in error and consequent failure of computation, as a result, quantum circuits need to be fault-tolerant in a practical implementation [14, 16]. Because the gates can be implemented in a fault-tolerant way, and the fault-tolerant implementations of the gates are known for most technologies that are considered promising for large-scale quantum computing, there has been renewed interest in using Clifford+T library to realize quantum circuits [2, 21, 29]. Quantum computers that are based on ion-trap technology, superconducting, and quantum dots need to fulfill certain physical constraints [4, 5, 26, 27]. Fault-tolerant implementations of quantum circuits also require restricted interactions among neighboring quantum bits [4, 14]. While realizing a given quantum functionality to a given quantum architecture, in order to achieve high fidelity, the so-called nearest-neighbor (NN) constraints imposed by lattice models, which require that quantum operations can be performed only between adjacent qubits [5, 26, 27], or the coupling constraints imposed by IBM quantum architectures, which are also called CNOT-constraints and allow quantum operations applied only between certain pairs of qubits on the coupling graph [4], must be complied with.

Usually, to realize reversible circuits to a quantum architecture, gate decomposition is first performed to decompose reversible gates in the circuit into quantum gates from a particular gate library, and then, qubit placement or qubit mapping is conducted to convert the resulting quantum circuits to satisfy the NN-constraints or coupling constraints at the quantum circuit level [4, 5, 9, 10, 12, 19, 27]. However, by combining gate decomposition and qubit mapping, NN-constraints or coupling constraints can also be addressed at the reversible logic level by defining proper cost metrics for the nearest-neighbor cost (NNC) [11, 26], designing nearest-neighbor templates [19], or computing the optimal combination of SWAPs and templates [17].

Different from the binary decision diagram (BDD) or the functional decision diagram (FDD) which are built by carrying out only Shannon decompositions or Davio decompositions over the variables of a function, the Kronecker FDD (KFDD) is built by applying Shannon and Davio decompositions over the variables. The KFDD as a generalization of the BDD and the FDD always will be more compact than the two [6]. Hence, the reversible circuits synthesized using the KFDD are potentially better than which synthesized using the BDD or the FDD. While synthesizing reversible circuits using the KFDD, although how to reduce the quantum cost and the number of qubits has been extensively researched, the restricted interactions between quibts are rarely considered.

In this work, we focus on the NN-constraints. By using gates from the Clifford+T library which are also supported by IBM quantum architectures [4] to realize reversible circuits, we attempt to handle the NN-constraints at the reversible logic level while synthesizing reversible circuits from the KFDD using NOT, CNOT, Toffoli and mixed-polarity Peres (MPP) gates. A common way to make a quantum circuit nearest-neighbor (NN) compliant is to apply SWAP-gates for quantum gates over non-adjacent qubits [5, 10, 19]. The insertion of SWAP-gates increases the total number of quantum gates, and thus affects the operational reliability of quantum circuits [4, 9]. Therefore, it is necessary to keep the NNC, which is a marker to the number of SWAP-gates needed to make a quantum circuit NN-compliant, as low as possible.

The main contributions of this work are listed as follows.

1) The NN-constraints are handled at the reversible logic level while synthesizing reversible circuits using the KFDD. It is an attempt to combine reversible logic synthesis, gate decomposition, and qubit mapping in one synthesis flow.
2) The Clifford+T quantum mappings of different MPP gates are presented.
3) NNC metrics for the reversible logic level are defined by considering the Clifford+T quantum mapping for the Toffoli and MPP gates.
4) Strategies guided by NNC metrics are presented to rank the ordering of nodes to be mapped for reducing the NNC while synthesizing reversible circuits using the KFDD.

The rest of this paper is structured as follows: Section 2 first briefly introduces reversible and quantum circuits, and then presents the Clifford+T quantum mappings of MPP gates and the NNC metrics defined for the NOT, CNOT, Toffoli, and MPP gates. Section 3 describes the KFDD and dependency matrices. In Sect. 4, the mapping of nodes by node dependency and by using local transformations are first introduced to keep the paper self-contained, and then, the synthesis of reversible circuits using the KFDD by combining the dependency matrix and local transformations is described, at last, the proposed synthesis method is detailed. Finally, the obtained experimental results are summarized in Sect. 5 while the paper is concluded in Sect. 6.

2 Reversible and Quantum Circuits

A reversible gate realizes a reversible function. A reversible circuit is a cascade of reversible gates without fanout or feedback [28].

In this work, graphic forms are adopted to illustrate a reversible gate or circuit, also a quantum gate or circuit. In the graphic form of a gate or circuit, the horizontal lines are called circuit lines, or, for short, lines. The symbols located on the left of a line indicate the input of the line. Whereas the symbols located on the right of a line indicate the output of the line.

Figure 1 presents the graphic illustrations of the NOT, CNOT and Toffoli gates which compose the NCT library [2]. The symbol '⊕' in Fig. 1 represents the exclusive-or operation.

For evaluating the quality of reversible circuits, the number of lines (or, for short, #lines) and quantum cost which depends on the quantum gate library used while realizing reversible circuits are often taken into account. A frequently used quantum gate library is the NCV library

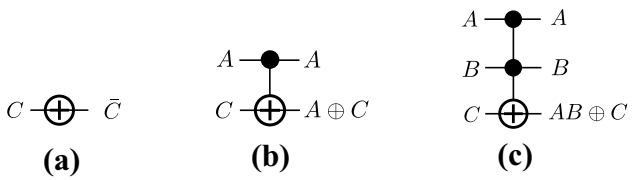

Fig. 1 NCT gate library: (**a**) NOT gate, (**b**) CNOT gate, (**c**) Toffoli gate

which is composed of the NOT, CNOT, V, and $V^\dagger$ gates [2]. Because the gates can be implemented in a fault-tolerant way and are supported by IBM quantum architectures, there has been renewed interest in using Clifford+T library to realize quantum circuits [2, 4, 21, 29].

When the NCV library is used, the quantum cost of a circuit is usually measured by NCV-cost. The NCV-cost of a reversible gate is equal to the number of elementary quantum operations required to implement its functionality [13]. Both the NOT gate and the CNOT gate have an NCV-cost of 1 [2]. The NCV-cost of the Toffoli gate is 5 [2]. For a reversible circuit, the NCV-cost is the cumulative NCV-cost sum of the reversible gates in the circuit.

When the Clifford+T library is used, the quantum cost of a circuit is often measured by T-count and also by T-depth [2, 21, 29]. This is due to the high cost of fault tolerant implementations of the T gate, exceeding the cost of Clifford group gates (CNOT, H, S gates) by as much as a factor of a hundred or more [18]. The T-count of a circuit is the total number of T and $T^\dagger$ gates in the circuit. Whereas the T-depth of a circuit is the number of T or $T^\dagger$ gates that have to be processed sequentially [2]. In addition, since quantum operations are prone to various errors, the number of quantum gates (or, for short, #QG) in the circuit is also an important cost metric for evaluating a quantum circuit [4, 9].

Figure 2 graphically illustrates the mixed-polarity Peres (MPP) gates. Using MPP gates to synthesize reversible circuits helps reduce the quantum cost [3]. The NCV-cost

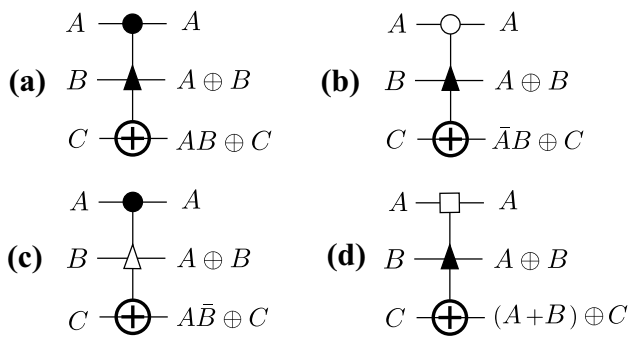

Fig. 2 MPP gate library: (**a**) Peres gate, (**b**) Peres gate with the first control negated, (**c**) inverse Peres gate, (**d**) or-Peres gate

of MPP gates are all 4 [3]. In the graphic forms of NCT or MPP gates as presented in Fig. 1 or Fig. 2, the lines on which symbols '•', '∘', '▲', '△', or '□' are located are considered as control lines. They take A or B as the input. The line on which the symbol '⊕' is located is the target line. It takes C as the input.

According to the Clifford+T quantum mappings of Toffoli gates with different polarities [2], by using equivalence and permutation, the Clifford+T quantum mappings of different MPP gates can be derived. Figure 3 presents the Clifford+T quantum mappings of different MPP gates. To keep the paper self-contained, Fig. 3(e) presents the Clifford+T quantum mapping of the Toffoli gate [2]. As can be seen from Fig. 3, the T-count and the T-depth of the Toffoli and MPP gates are 7 and 3, respectively. The #QG of the Peres gate, Peres gate with the first control negated, and inverse Peres gate are all 15. Whereas, the #QG of the or-Peres gate and Toffoli gate are both 16.

For making a quantum circuit NN-compliant, a common way is to apply SWAP-gates for two-qubit quantum gates over non-adjacent qubits [5, 10, 19]. Accordingly, for evaluating the effort to convert a quantum gate or quantum circuit to be NN-compliant, the NNC metrics for a quantum gate or quantum circuit which are defined as the number of SWAP-gates applied are proposed [10, 11]. The NNC of a quantum gate can directly be determined by considering the distance between the control line and the target line [11, 19]. Obviously, single-qubit quantum gates (e.g., the NOT, T, or H gates) have an NNC of 0.

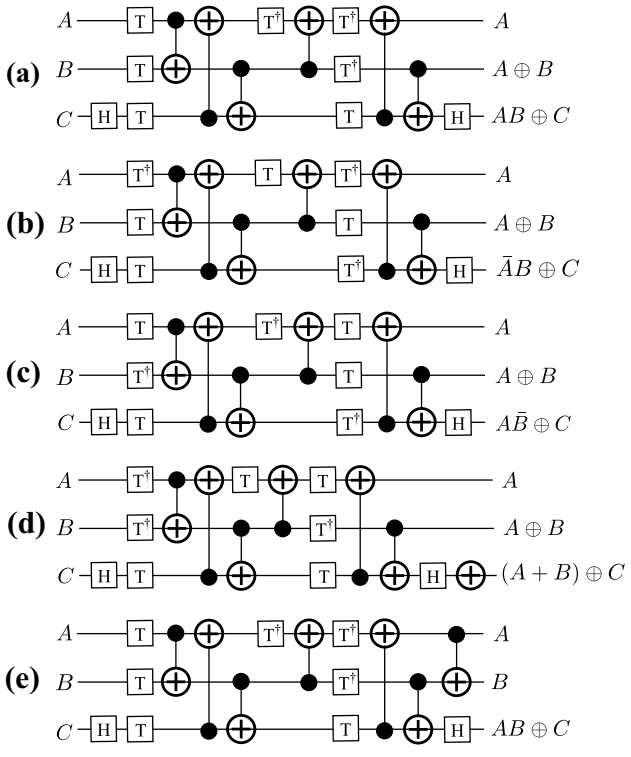

Fig. 3 Clifford+T quantum mappings of different MPP gates and the Toffoli gate: (**a**) Peres gate, (**b**) Peres gate with the first control negated, (**c**) inverse Peres gate, (**d**) or-Peres gate, (**e**) Toffoli gate

By considering NCV quantum mapping for multiple-control Toffoli gates, Kole et al. [11] proposed NNC metrics for the reversible logic level. In this work, by considering the Clifford+T quantum mapping for NCT and MPP gates, the NNC metrics for NCT and MPP gates for the reversible logic level are defined.

Assuming a numerical encoding of the circuit lines from the topmost line to the undermost line. For the CNOT, Toffoli, or MPP gates presented in Fig. 1 or Fig. 2, and the Clifford+T quantum mappings of the Toffoli gate and different MPP gates presented in Fig. 3, suppose that the lines taking A, B, or C as the input are numerically encoded by L_A, L_B, and L_C, respectively. In the following, those lines are called line L_A, line L_B, and line L_C, respectively.

In this work, NCT and MPP gates are used to synthesize reversible circuits from the KFDD. That is, the reversible circuits are composed of NOT, CNOT, Toffoli, and MPP gates. To realize a reversible circuit using gates from the Clifford+T library, a NOT or CNOT gate in the reversible circuit is directly mapped to a NOT or CNOT gate, without the need to be decomposed. Whereas, the Toffoli and MPP gates in the reversible circuit need to be mapped or decomposed into the quantum gate cascades as shown in Fig. 3. If the Clifford+T quantum mappings presented in Fig. 3 are considered as templates, the mapping or decomposition of a Toffoli or an MPP gate is to substitute the reversible gate by the corresponding template. Single-qubit quantum gates have an NNC of 0. Thus, for making the resulting quantum circuits NN-compliant, only whether those CNOT gates are NN-compliant needs to be considered.

It is usually assumed that two SWAP-gates are required in order to decrease the distance between the control and the target line of a two-qubit gate by one (one SWAP-gate for moving the control and the target line together, another to restore the original order) [11, 19]. Considering the first CNOT gate presented in Fig. 3(a) as an example. When the CNOT gate is not NN-compliant (i.e., $|L_A - L_B| > 1$), firstly, $|L_A - L_B| - 1$ SWAP-gates are inserted in front of the CNOT gate for moving the control and the target line of the CNOT gate being next to each other, and then, $|L_A - L_B| - 1$ SWAP-gates are inserted behind the CNOT gate for restoring the original order of lines.

Example 1 Figure 4 presents a trivial circuit with 4 lines and a CNOT gate which takes line L_A as the control line and line L_B as the target line, as well as the transformed circuit which satisfies the NN-constraints. Assuming that $L_A = 1$ and $L_B = 4$.

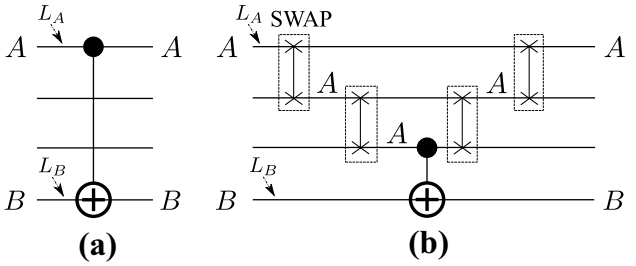

Fig. 4 SWAP insertion: (**a**) the original circuit, (**b**) the transformed circuit

Known that $|L_A - L_B| = 3$. Then for converting the CNOT gate presented in Fig. 4(a) to be NN-compliant, 2 SWAP-gates are successively inserted in front of the CNOT gate to make the control and the target line of the CNOT gate adjacent. After the CNOT operation has been performed, 2 SWAP-gates are successively inserted behind the CNOT gate to restore the original order of lines. Consequently, $2(|L_A - L_B| - 1) = 4$ SWAP-gates are inserted in the circuit, the transformed circuit presented in Fig. 4(b) is resulted.

As can be seen from Fig. 3(a), the Clifford+T quantum mapping of a Peres gate has 6 CNOT gates. To convert those CNOT gates to be NN-compliant, $4(|L_A - L_B| - 1)$, $4(|L_A - L_C| - 1)$, and $4(|L_B - L_C| - 1)$ SWAP-gates are required for the two CNOT gates applied between line L_A and line L_B, the two CNOT gates applied between line L_C and line L_A, and the two CNOT gates applied between line L_B and line L_C, respectively. Consequently, to convert the Clifford+T quantum mapping of a Peres gate as shown in Fig. 3(a) to be NN-compliant, a total of $4(|L_A - L_B| - 1) + 4(|L_B - L_C| - 1) + 4(|L_A - L_C| - 1)$ SWAP-gates are needed. In a similar way, the number of SWAP-gates needed to make the Clifford+T quantum mappings of a Toffoli gate, a Peres gate with the first control negated, an inverse Peres gate, or an or-Peres gate as shown in Fig. 3 to be NN-compliant can be derived.

In conclusion, by the CNOT gate presented in Fig. 1(b) as well as the Clifford+T quantum mappings presented in Fig. 3, the NNC of the CNOT, Toffoli, and MPP gates can be evaluated using the equations as follows.

$$\begin{cases} NNC(CNOT) & = 2(|L_A - L_C| - 1) \\ NNC(TOF) & = 4(|L_A - L_C| - 1) \\ & + 4(|L_B - L_C| - 1) \\ & + 6(|L_A - L_B| - 1) \\ NNC(MPP) & = 4(|L_A - L_C| - 1) \\ & + 4(|L_B - L_C| - 1) \\ & + 4(|L_A - L_B| - 1) \end{cases} \tag{1}$$

By using the above equations, the NNC of a circuit can be evaluated at the reversible logic level. The T-count, #QG, or NNC of a reversible circuit is the cumulative sum of the T-count, #QG, or NNC of the reversible gates in the circuit, respectively. Similarly, the T-depth of a reversible circuit is estimated as the cumulative sum of the T-depth of the reversible gates in the circuit.

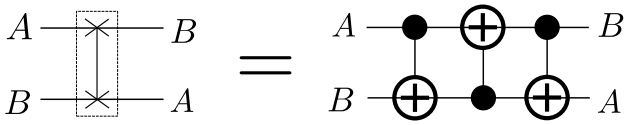

Fig. 5 Quantum realization of the SWAP-gate

The insertion of SWAP-gates affects the operational reliability of quantum circuits [9]. Thus, it is important to reduce the NNC which is a marker to the number of SWAP-gates needed to make a quantum circuit NN-compliant. By using the NNC metrics in Eq. (1), the NNC can be reduced (i.e., the NN-constraints can be handled) at the reversible logic level while synthesizing a reversible circuit. Therefore, it is possible to combine reversible logic synthesis, gate decomposition, and qubit mapping in one synthesis flow.

At the quantum circuit level, a SWAP-gate is usually realized using 3 CNOT gates [12] as shown in Fig. 5. Therefore, known the NNC of a circuit, the number of quantum gates needed to convert the quantum circuit to be NN-compliant is 3 times of the NNC.

3 KFDD and Dependency Matrices

3.1 KFDD

Let $f : B^n \rightarrow B$ be a totally defined Boolean function and $X = \{x_1, x_2, \cdots, x_n\}$ be the corresponding set of primary variables. The function f can be decomposed over the primary variable set X using the following well-known decomposition types [20].

$$\begin{cases} f = \bar{x}_i f_{\bar{x}_i} \oplus x_i f_{x_i} & \textit{Shannon} \\ f = f_{\bar{x}_i} \oplus x_i (f_{\bar{x}_i} \oplus f_{x_i}) & \textit{Positive Davio} \\ f = f_{x_i} \oplus \bar{x}_i (f_{\bar{x}_i} \oplus f_{x_i}) & \textit{Negative Davio} \end{cases} \tag{2}$$

where $f_{\bar{x}_i}$ and f_{x_i} denote the cofactors of f with respect to $x_i = 0$ and $x_i = 1$ respectively, $\oplus$ is the exclusive-OR operation.

Definition 1 A KFDD for the function f is a rooted DAG $G = (V, E)$ with node set V and edge set E.

A KFDD for the function f is built by carrying out Shannon decomposition, positive or negative Davio

decompositions as shown in Eq. (2) over X, with one type of decomposition per primary variable. An inner node in the KFDD is labeled with one of the primary variables and has exactly two successors, the low and the high successors. The terminal nodes of a KFDD indicate constant 0 or 1 and have no successors. The primary variable that labels an inner node $v_i \in V$ is denoted as $var(v_i)$ ($var(v_i) \in X$). The level at which node v_i is positioned in the KFDD is denoted as $lev(v_i)$ ($lev(v_i) \geq 1$). The root of the KFDD is positioned at the top level, the terminal 0 or 1 nodes are positioned at the $(n+1)$ -th level. If node $v_j \in V$ is one of the descendants of node v_i (i.e., v_i is a paraent node of v_j), $lev(v_j) > lev(v_i)$. Since variable $x_j \in X$ labels all of the nodes positioned at one level in the KFDD, the level at which x_j is positioned in the KFDD is denoted as $lev(x_j)$.

In this work, the conventions for a KFDD presented in [6] are complied with. Only the edge from node v_i to its high successor (i.e., this edge takes node v_i and its high successor as the tail and the head, respectively) can be a complemented edge, and only the terminal 1 node is used in a KFDD. The terminal 0 node is represented by a terminal 1 node pointed to by a complemented edge.

Suppose the low and high successors of node v_i are nodes v_j and v_k, respectively. Then the following definitions are presented.

Definition 2 The function represented by node v_i, which is denoted by f_{v_i}, can be derived by the following equations depending on the decomposition type of the variable $x_l = var(v_i)$.

$$\begin{cases} f_{v_i} = \bar{x}_l f_{v_j} \oplus x_l f_{v_k} & Shannon \\ f_{v_i} = f_{v_j} \oplus x_l f_{v_k} & Positive\ Davio \\ f_{v_i} = f_{v_j} \oplus \bar{x}_l f_{v_k} & Negative\ Davio \end{cases}$$

where f_{v_j} and f_{v_k} are the functions represented by nodes v_j and v_k, respectively. If the edge from node v_i to node v_k is a complemented edge, the function f_{v_k} in above equations is complemented.

Definition 3 For an inner node v_i, if the function f_{v_i} can be expressed as $f_{v_i} = var(v_i) \oplus g$ or $f_{v_i} = \overline{var(v_i)} \oplus g$, where the function g is independent of variable $var(v_i)$, then v_i is called a linear node.

Fig. 6 presents sub-graphs for linear nodes in the KFDD. Note that in Fig. 6, a dashed (solid) edge having node v_i as a tail means an edge from node v_i to the low (high) successor of v_i. A white dot on a solid edge means that the edge is a complemented edge.

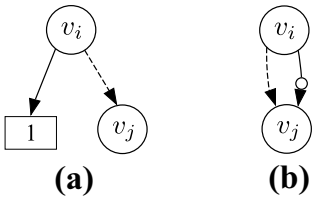

Fig. 6 Sub-graphs for linear nodes. The decomposition type of variable $var(v_i)$ is a: (**a**) Davio decomposition, (**b**) Shannon decomposition

As shown in Fig. 6, if v_i is a linear node, by Definition 2 and Definition 3, it is known that its high successor (node v_k) must be the terminal 1 node and the edge from node v_i to the terminal 1 node must not be a complemented edge when the decomposition type of variable $var(v_i)$ is a Davio decomposition, or there is $v_j = v_k$ and the edge from node v_i to node v_k must be a complemented edge when the decomposition type of variable $var(v_i)$ is a Shannon decomposition.

Variable nodes defined in the following are special kinds of linear nodes.

Definition 4 For a linear node v_i, if v_j is the terminal 1 node, v_i is called a variable node, due to the fact that $f_{v_i} = var(v_i)$ or $f_{v_i} = \overline{var(v_i)}$.

FDDs are special kinds of the KFDD [6]. An FDD for the function f is built by carrying out positive or negative Davio decompositions as shown in Eq. (2) over X, with one type of decomposition per primary variable.

3.2 Dependency Between Nodes and Dependency Matrices

Due to the sharing of nodes in FDDs or KFDDs, while mapping a node to a cascade of reversible gates (or, for simplicity, a reversible gate cascade or reversible cascade), an ancilla line is generally added to the circuit to store the result of the node function. In order to reduce the number of circuit lines required for the resulting circuits by reusing ancilla lines, Stojković et al. [23] proposed dependency between nodes in FDDs for ranking the ordering of nodes to be mapped and dependency matrices for expressing the dependency between nodes. In this work, dependency between nodes and dependency matrices are generalized to KFDDs.

For inner nodes v_i, v_j, and v_k in a KFDD, suppose that node v_k is a successor of both node v_i and node v_j. Then the definitions of dependency between nodes are presented in the following.

Definition 5 Nodes v_i and v_j are strongly dependent on node v_k.

Definition 6 When the decomposition type of variable $var(v_i)$ is a Davio decomposition and node v_k is the low successor of node v_i, or the decomposition type of variable $var(v_i)$ is a Shannon decomposition and v_i is a linear node. If the decomposition type of variable $var(v_j)$ is a Davio decomposition and node v_k is the high successor of v_j, or the decomposition type of variable $var(v_j)$ is a Shannon decomposition and v_j is not a linear node, then node v_i is weakly dependent on node v_j.

Figure 7 presents sub-graphs in which node v_i weakly depends on node v_j. According to Definitions 5 and 6, if node v_i strongly or weakly depends on node v_j, it is called that node v_i depends on node v_j. Otherwise, it is called that node v_i does not depend on node v_j, or node v_i is independent of node v_j.

Suppose there are w inner nodes in a KFDD which constitute a set $V = \{v_i | 1 \le i \le w\}$. Then for using a dependency matrix to express the dependency between those w nodes, V is mapped onto the set $J = \{j | 1 \le j \le w\}$. In other words, every node in V is encoded by one and only one integer in the set J and every integer in the set J corresponds to one and only one node in V. Suppose that node v_i is encoded by $j \in J$ which is also the row index or column index of the dependency matrix. Then in the dependency matrix, the j-th row and the j-th column are both correlated to node v_i. The j-th row indicates that node v_i strongly or weakly depends on which nodes. Whereas the j-th column indicates which nodes weakly or strongly depend on node v_i. In the following, in the context of dependency matrices, node j or $Node(j)$ is referred to as the node which is encoded by the integer $j \in J$. For a KFDD, suppose that w_l is the number of nodes positioned at level l in the KFDD and w is the number of inner nodes in the KFDD. Then based on Definitions 5 and 6, the dependency matrix [23] is generalized to express the dependency between nodes in the KFDD as follows.

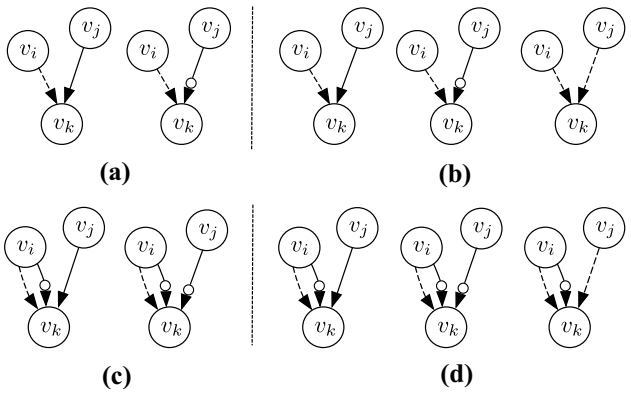

Fig. 7 Sub-graphs in which node v_i weakly depends on node v_j: (**a**) the decomposition types of $var(v_i)$ and $var(v_j)$ are both Davio decompositions, (**b**) the decomposition types of $var(v_i)$ and $var(v_j)$ are Davio and Shannon decompositions, respectively, and node v_j is not a linear node, (**c**) the decomposition types of $var(v_i)$ and $var(v_j)$ are Shannon and Davio decompositions, respectively, (**d**) the decomposition types of $var(v_i)$ and $var(v_j)$ are both Shannon decompositions and node v_j is not a linear node

Definition 7 The level dependency matrix (LDM) with respect to the l-th level in the KFDD is a $w_l \times w_l$ matrix, whose element d_{ij} is set to 1 when node i weakly depends on node j, or set to 0 when node i is independent of node j.

Definition 8 The diagram dependency matrix (DDM) for the KFDD is a $w \times w$ matrix, whose element d_{ij} is set to 2 when node i strongly depends on node j, set to 1 when node i weakly depends on node j, or set to 0 when node i is independent of node j.

In a dependency matrix, if all of the elements of the i-th row have an value of 0, the row is called a *zero* row. The i-th row being a *zero* row implies that the mapping conditions for node i are satisfied. That is, node i can be mapped to a cascade of reversible gates.

4 Synthesis of Reversible Circuits With Reduced NNC Using the KFDD

4.1 Mapping Nodes By Node Dependency

For inner nodes v_i and v_j in a KFDD, when node v_i strongly depends on node v_j, node v_i has to be mapped after node v_j [23]. Strong dependency between nodes assures the functional equivalence of the resulting reversible circuit to the original function by applying strong constraint on the ordering of nodes to be mapped. When node v_i weakly depends on node v_j, node v_i is preferred to be mapped after node v_j [23]. Weak dependency aims at reusing ancilla lines by applying weak constraint on the ordering of nodes to be mapped.

Example 2 presented in the following is used to demonstrate the mapping of nodes by node dependency. Note that, if a node has been mapped into a reversible cascade, it is called a mapped node. Otherwise, it is called a node not mapped.

Example 2 Figure 8 presents a sub-graph for nodes v_1 and v_2 which is extracted from a KFDD and reversible cascades mapped for nodes v_1 and v_2 by using different ordering of nodes to be mapped. Following the conventions presented in Sect. 2, in Fig. 8(b) and (c), the symbols located on the left of a line indicate the input of the line, whereas the symbols located on the right of a line indicate the output of the line.

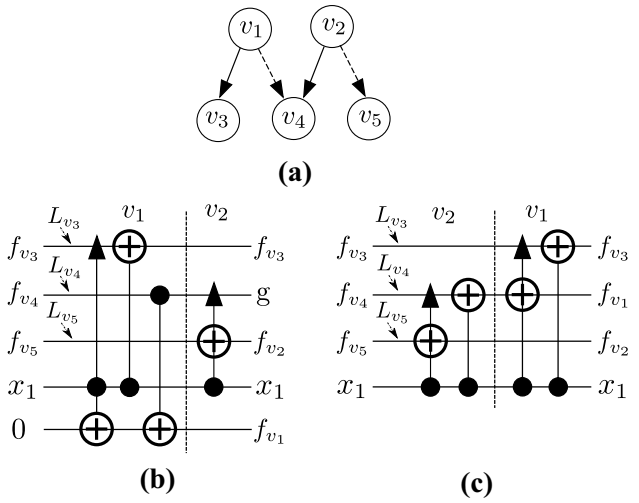

Fig. 8 Sub-graph for nodes v_1 and v_2 as well as the constructed reversible cascades by using different ordering of nodes v_1 and v_2 to be mapped: (**a**) the sub graph for nodes v_1 and v_2, (**b**) node v_1 is mapped before node v_2, (**c**) node v_2 is mapped before node v_1

In addition, the reversible cascade indicated by symbol 'v_i' ($1 \le i \le 2$) is the mapping result for node v_i. For the sake of the simplicity of description, suppose that L_{v_3}, L_{v_4}, and L_{v_5} indicate the numerical encoding of the lines taking f_{v_3}, f_{v_4}, and f_{v_5} as the input, respectively. Then the lines taking f_{v_3}, f_{v_4}, and f_{v_5} as the input are called line L_{v_3}, line L_{v_4}, and line L_{v_5}, respectively.

The sub graph presented in Fig. 8(a) consists of 5 nodes which are all inner nodes in the complete KFDD. Assuming that $var(v_1) = var(v_2) = x_1$, and the decomposition type of x_1 is a positive Davio decomposition. Moreover, in the complete KFDD, there is not any other node which takes node v_5 as a successor excepting node v_2, and there is not any other node which takes node v_4 as a successor excepting nodes v_1 and v_2.

Known by Definitions 5 and 6, node v_1 strongly depends on nodes v_3 and v_4, node v_2 strongly depends on nodes v_4 and v_5, and node v_1 weakly depends on node v_2. According to Definition 2, there are $f_{v_1} = f_{v_4} \oplus x_1 f_{v_3}$ and $f_{v_2} = f_{v_5} \oplus x_1 f_{v_4}$.

Node v_1 strongly depends on nodes v_3 and v_4. Thus, nodes v_3 and v_4 should be mapped before node v_1. Similarly, nodes v_4 and v_5 should be mapped before node v_2. Here, we focus on the effect of mapping nodes by weak dependency. That is, we focus on the ordering of nodes v_1 and v_2 to be mapped.

Assuming that nodes v_3, v_4, and v_5 have been mapped into reversible cascades (which are not presented in Fig. 8(b) and (c)), and lines L_{v_3}, L_{v_4}, and L_{v_5} have been used to store the results of the functions f_{v_3}, f_{v_4}, and f_{v_5}, respectively. Next, the impact of the ordering of nodes v_1 and v_2 to be mapped on the number of required lines is demonstrated.

If node v_1 is mapped before node v_2, by the expressions of the functions f_{v_1} and f_{v_2}, the reversible cascade as shown in Fig. 8(b) is constructed. When mapping node v_1, because node v_4 has another parent node not mapped (i.e., node v_2), line L_{v_4} can not be reused to store the result of f_{v_1}. An ancilla line with the initial value of 0 needs to be first added to the circuit, and then, the function $h_1 = x_1 f_{v_3} \oplus 0$ is synthesized to a Peres gate and a CNOT gate as shown in Fig. 8(b). The CNOT gate (the first CNOT gate presented in Fig. 8(b)) is used to recover the value on line L_{v_3}, becuase there may be other nodes not mapped in the complete KFDD which take v_3 as a successor in addition to node v_1. After that, the function $f_{v_1} = f_{v_4} \oplus h_1$ is synthesized to a CNOT gate (the second CNOT gate presented in Fig. 8(b)) and the added ancilla line is used to store the result of function f_{v_1} as shown in Fig. 8(b). Node v_4 has two parent nodes (i.e., the nodes v_1 and v_2) in the complete KFDD, but node v_1 is a mapped node when mapping node v_2. Hence, while mapping node v_2, there is not any other node not mapped which takes node v_4 as a successor excepting node v_2. Meanwhile, there is not any other node in the complete KFDD which takes node v_5 as a successor excepting node v_2. Consequently, the function $f_{v_2} = x_1 f_{v_4} \oplus f_{v_5}$ is synthesized to a Peres gate by reusing line L_{v_5} to store the result of function f_{v_2}. Note that, the output of line L_{v_4} becomes a garbage output indicated by the symbol 'g'. The reversible cascade presented in Fig. 8(b) is composed of 2 Peres gates, 2 CNOT gates, and 5 lines.

However, since node v_1 weakly depends on node v_2, node v_1 is preferred to be mapped after node v_2. Therefore, by weak dependency, node v_2 is first mapped, and then node v_1. While mapping node v_2, because there is not any other node in the complete KFDD which takes node v_5 as a successor excepting node v_2, the function $f_{v_2} = x_1 f_{v_4} \oplus f_{v_5}$ is synthesized to a Peres gate and a CNOT gate by reusing line L_{v_5} to store the result of function f_{v_2}. Node v_4 has two parent nodes (i.e., the nodes v_1 and v_2) in the complete KFDD, but node v_2 is a mapped node when mapping node v_1. Hence, while mapping node v_1, there is not any other node not mapped in the complete KFDD which takes v_4 as a successor excepting node v_1. As a result, $f_{v_1} = x_1 f_{v_3} \oplus f_{v_4}$ is synthesized to a Peres gate and a CNOT gate by reusing line L_{v_4} to store the result of function f_{v_1}. The reversible cascade as shown in Fig. 8(c) is constructed. This reversible cascade is composed of 2 Peres gates, 2 CNOT gates, and 4 lines.

As can be seen from above, mapping nodes by weak dependency, excepting that line L_{v_5} is reused to store the result of function f_{v_2} while mapping node v_2, line L_{v_4} is also reused to store the result of function f_{v_1} while mapping node v_1, no extra ancilla lines need to be added to the circuit. As a result, the number of required lines is reduced.

4.2 Mapping Nodes By Applying Local Transformations

While mapping a node, Bu and Wang [3] had detailed that which type of local transformations can be applied on the node function and the conditions to be satisfied for applying the local transformations, as well as the reversible cascades constructed for the node by applying different local transformations on the node function in their paper. More details about local transformations are referred to Ref. [3]. However, to keep the paper self-contained, Example 3 presented in the following is used to demonstrate the construction of a reversible cascade by applying local transformations on the node function while mapping a node.

Example 3 Figure 9 presents the sub-graph for node v_1 and the reversible cascades mapped for v_1. Suppose that $var(v_1) = x_1$ and the decomposition type of x_1 is a Shannon decomposition. In addition, nodes v_2 and v_3 are both inner nodes in the complete KFDD, and have been mapped into reversible cascades (which are not presented in Fig. 9) by using lines taking f_{v_2} and f_{v_3} as the input to store the results of function f_{v_2} and function f_{v_3}, respectively, as shown in Fig. 9(b)-(f).

Known from Definition 2, $f_{v_1} = \bar{x}_1 f_{v_2} \oplus x_1 f_{v_3}$, which is the original form of the expression of function f_{v_1}. When mapping node v_1 by this expression of function f_{v_1}, an ancilla line with the initial value of 0 is first appended to the circuit. And then, the functions $h_1 = x_1 f_{v_3} \oplus 0$ and $f_{v_1} = \bar{x}_1 f_{v_2} \oplus h_1$ are successively synthesized to reversible gates as shown in Fig. 9(b). The function h_1 is realized using a Toffoli gate. Whereas, the function $f_{v_1} = \bar{x}_1 f_{v_2} \oplus h_1$ is realized using a Peres gate with the first control negated and a CNOT gate. Note that, the CNOT gate is used to recover the value on the line taking f_{v_2} as the input. The #lines, T-depth, T-count, and #QG of the reversible cascade presented in Fig. 9(b) are 4, 6, 14, and 32, respectively.

However, by applying local transformations on function f_{v_1}, the expression $f_{v_1} = x_1(f_{v_2} \oplus f_{v_3}) \oplus f_{v_2}$ can be derived. Node v_1 can be mapped into different reversible cascades by further applying local transformations on function f_{v_1} as the four cases described in the following.

(i) Case I. When nodes v_2 and v_3 both have other parent nodes not mapped in the complete KFDD in addition to node v_1, by successively synthesizing the functions $h_2 = f_{v_2} \oplus f_{v_3}$, $h_3 = x_1 h_2 \oplus 0$, $f_{v_1} = f_{v_2} \oplus h_3$, and $f_{v_3} = f_{v_2} \oplus h_2$, the reversible cascade as shown in Fig. 9(c) is constructed. An ancilla line with the initial value of 0 is added to the circuit when synthesizing the function h_3 and is used to store the result of function f_{v_1} when synthesizing $f_{v_1} = f_{v_2} \oplus h_3$. The #lines, T-depth, T-count, and #QG of the reversible cascade presented in Fig. 9(c) are 4, 3, 7, and 19, respectively.

(ii) Case II. When node v_2 has other parent nodes not mapped in the complete KFDD in addition to node v_1, but node v_3 does not have any other parent node not mapped excepting node v_1, node v_1 can be mapped in a similar way as the above case. However, while mapping node v_1 in this case, node v_3 does not have any other parent node not mapped excepting node v_1. Hence, a Peres gate instead of a Toffoli gate is used to realize the function h_3, and the last function $f_{v_3} = f_{v_2} \oplus h_2$, which is used to recover the value on the line taking f_{v_3} as the input, does not need to be realized. As a result, the output of the line taking f_{v_3} as the input becomes a garbage output indicated by the symbol 'g'. The constructed reversible cascade is presented in Fig. 9(d). The #lines, T-depth, T-count, and #QG of this reversible cascade are 4, 3, 7, and 17, respectively.

(iii) Case III. When node v_3 has other parent nodes not mapped in the complete KFDD in addition to node v_1, but node v_2 does not have any other parent node not mapped excepting node v_1, by successively synthesizing the functions $h_4 = f_{v_2} \oplus 0$, $h_2 = f_{v_3} \oplus f_{v_2}$, and $f_{v_1} = x_1 h_2 \oplus h_4$, the reversible cascade as shown in Fig. 9(e) is constructed. An ancilla line with the initial value of 0 is appended to the circuit when synthesizing the function h_4 and is used to store the result of function f_{v_1} when synthesizing $f_{v_1} = x_1 h_2 \oplus h_4$. Moreover, the output of the line taking f_{v_2} as the input becomes a garbage output indicated by the symbol 'g'. The #lines, T-depth, T-count, and #QG of the reversible cascade presented in Fig. 9(e) are 4, 3, 7, and 17, respectively.

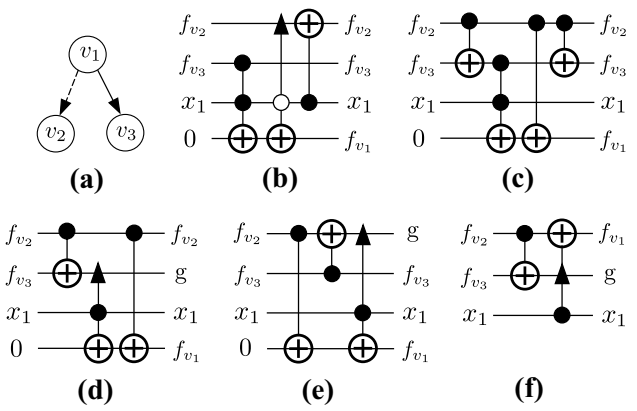

Fig. 9 Sub-graph for node v_1 and reversible cascades mapped for v_1: (**a**) the sub graph for node v_1, (**b**) reversible cascade constructed by using the original form of the expression of function f_{v_1}, (**c**) Case I, (**d**) Case II, (**e**) Case III, (**f**) Case IV

(iv) Case IV. When neither of nodes v_2 and v_3 have any other parent node not mapped in the complete KFDD excepting node v_1, by successively synthesizing the functions $h_2 = f_{v_2} \oplus f_{v_3}$ and $f_{v_1} = x_1 h_2 \oplus f_{v_2}$, the reversible cascade as shown in Fig. 9(f) is constructed. Note that, when synthesizing the function $f_{v_1} = x_1 h_2 \oplus f_{v_2}$, the line taking f_{v_2} as the input is reused to store the result of function f_{v_1}. Hence, no ancilla line is added for mapping node v_1 in this case. Moreover, the output of the line taking f_{v_3} as the input becomes a garbage output indicated by the symbol 'g'. The #lines, T-depth, T-count, and #QG of the reversible cascade presented in Fig. 9(f) are 3, 3, 7, and 16, respectively.

Known from Example 3 in the above and the descriptions of local transformations presented in Ref. [3], when synthesizing reversible circuits using the KFDD by mapping a node into a reversible cascade, applying local transformations on the node function helps reduce the quantum cost of the resulting circuit, even helps reduce the number of required lines by reusing ancilla lines. Nevertheless, due to the sharing of a node in the KFDD (i.e., a node may have more than one parent node in the KFDD), there are different types of local transformations which can be applied on the node function. In fact, while mapping a node, the type of local transformations applied on the node function depends on the ordering of the node to be mapped [3].

4.3 Reversible Circuit Synthesis By Combining the Dependency Matrix and Local Transformations

Performing local transformations on the node function when mapping a node into a reversible cascade helps reduce the quantum cost, even the number of lines of reversible circuits synthesized using the KFDD [3]. On the other hand, the ordering of nodes in a KFDD to be mapped influences the number of lines, T-count, T-depth, and NNC of the circuit synthesized using the KFDD. Consequently, combining the dependency matrix and local transformations for synthesizing reversible circuits using the KFDD helps improve the quality of the resulting circuits.

For a node v_i, there are different types of local transformations which can be applied on function f_{v_i} [3]. Nevertheless, the type of local transformations applied on function f_{v_i} can be determined only after the ordering of node v_i to be mapped has been determined [3]. Consequently, in this work, the dependency matrix and local transformations are combined in a two-step way. Firstly, the ordering of nodes to be mapped is ranked by using dependency matrices. And then, local transformations are applied on the node function while mapping a node into a reversible cascade [3].

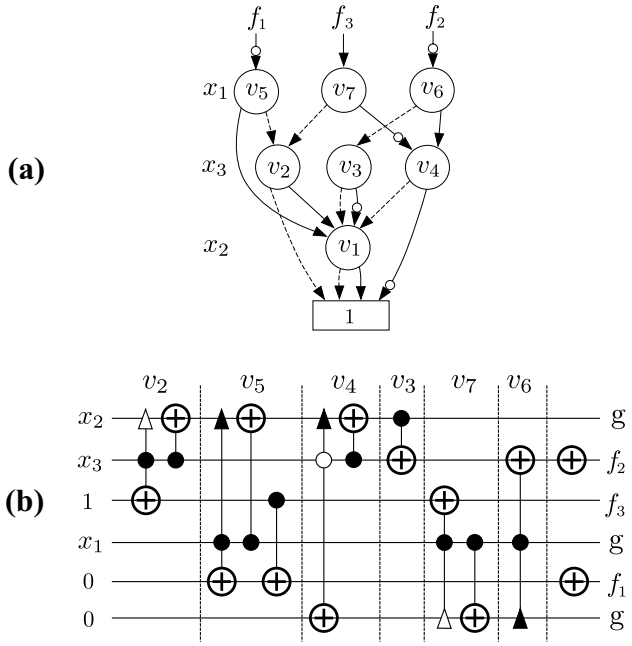

Fig. 10 The KFDD and reversible circuit for an exemplar function: (**a**) the KFDD, (**b**) the reversible circuit

However, dependency matrices only apply constraint on the ordering of nodes to be mapped which have dependencies between each other. The ordering of nodes to be mapped which do not depend on each other also influences the quality of the resulting circuit.

Example 4 Figure 10 presents a KFDD for an exemplar function with 3 inputs and 3 outputs as well as the circuit synthesized by using the DDM for that KFDD. Whereas, Fig. 11 presents two other reversible circuits synthesized from the KFDD presented in Fig. 10(a).

In the KFDD presented in Fig. 10(a), the variable labeling those nodes positioned at one level is placed on the left. The decomposition types of variables x_1, x_3, and x_2 are positive Davio decomposition, Shannon decomposition, and negative Davio decomposition, respectively.

According to the KFDD presented in Fig. 10(a) as well as the definitions of dependency between nodes and diagram dependency matrix, the DDM for the KFDD is created as follows.

$$DDM = \begin{array}{c} \\ v_1 \\ v_2 \\ v_5 \\ v_3 \\ v_4 \\ v_7 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} v_1 & v_2 & v_5 & v_3 & v_4 & v_7 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 \end{bmatrix} \end{array}.$$

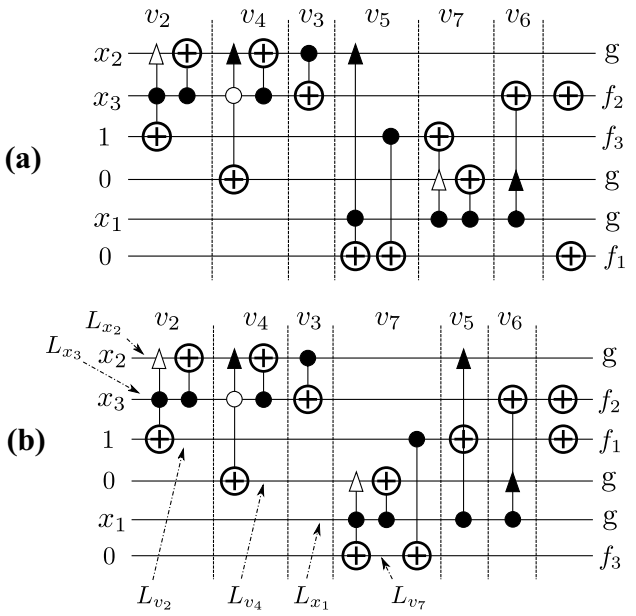

Fig. 11 Other reversible circuits synthesized for the exemplar function by using different ordering of nodes to be mapped: (**a**) $(v_1, v_2, v_4, v_3, v_5, v_7, v_6)$, (**b**) $(v_1, v_2, v_4, v_3, v_7, v_5, v_6)$

By the above DDM, according to '*Algorithm* 3' in Ref. [23], the ordering of nodes to be mapped is determined as (v_1 , v_2 , v_5 , v_4 , v_3 , v_7 , v_6), the reversible circuit presented in Fig. 10(b) is generated by applying local transformations on the node function while mapping a node into a reversible gate cascade [3]. The number of lines, T-count, T-depth, and NNC of that circuit is 6, 35, 15, and 92, respectively. Note that the symbols '1' or '0' located on the left of a circuit line imply that the initial value of the circuit line is 1 or 0. The symbol 'g' located on the right of a circuit line indicates that the output of the circuit line is a garbage output [2]. The reversible gate cascade indicated by symbol 'v_i' is the mapping result for node v_i.

On the other hand, according to the definitions of weak dependency between nodes and level dependency matrix, the LDMs for the KFDD presented in Fig. 10(a) are created as follows.

$$\begin{cases} LDM(lev(x_2)) = \begin{matrix} \\ v_1 \end{matrix} \begin{matrix} v_1 \\ \end{matrix} \begin{bmatrix} 0 \end{bmatrix} \\ LDM(lev(x_3)) = \begin{matrix} \\ v_2 \\ v_3 \\ v_4 \end{matrix} \begin{matrix} v_2 v_3 v_4 \\ \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ LDM(lev(x_1)) = \begin{matrix} \\ v_5 \\ v_7 \\ v_6 \end{matrix} \begin{matrix} v_5 v_7 v_6 \\ \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{cases}$$

By the above LDMs, according to '*Algorithm* 2' in Ref. [23], the ordering of nodes to be mapped is determined as $(v_1, v_2, v_4, v_3, v_5, v_7, v_6)$, the reversible circuit presented in Fig. 11(a) is resulted by applying local transformations on the node function while mapping a node into a reversible gate cascade [3]. The number of lines, T-count, T-depth, and NNC of the circuit is 6, 35, 15, and 64, respectively. Compared to the circuit presented in Fig. 10(b), the NNC of the circuit presented in Fig. 11(a) is reduced by 28.

Nodes v_5, v_6, and v_7 positioned at level $lev(x_1)$ in the KFDD presented in Fig. 10(a) do not depend on each other. When mapping nodes in a BFS manner, in other words, level by level from the bottom to the top level, the ordering of the nodes v_5, v_6, and v_7 to be mapped does not impact the functionality of the resulting circuit, but may influence the quality of the resulting circuit. If those three nodes are mapped in order of (v_7, v_5, v_6), a reversible circuit with the NNC of 56 as shown in Fig. 11(b) can be obtained. Compared to the circuit presented in Fig. 11(a), the NNC is further reduced by 8.

Note that, the symbols L_{x_2}, L_{x_3}, L_{v_2}, L_{v_4}, L_{x_1}, and L_{v_7} presented in Fig. 11(b) indicate the numerical encoding of the circuit lines from the topmost line to the undermost line (e.g., L_{x_2} indicates the numerical encoding of the line taking x_2 as the input). They will be used in Example 5 presented in Sect. 4.4.2 to describe the construction of the circuit presented in Fig. 11(b).

4.4 Synthesis of Reversible Circuits With Reduced NNC

From Example 4 presented in Sect. 4.3, it can be seen that the ordering of nodes in a KFDD to be mapped influences the NNC of the resulting circuit. Since NN-constraints require the control and the target line of a reversible gate to be adjacent, it is necessary to reduce the distance between the control and the target line of a reversible gate for reducing the NNC of reversible circuits.

Ranking the ordering of nodes to be mapped for reducing the NNC is a combinational optimization problem. Many discrete optimization algorithms (e.g., integer programming, genetic algorithm, etc.) can be used to solve this problem. However, in this work, a method by using strategies guided by NNC metrics is proposed for solving this problem. In the following, the strategies for ranking the ordering of nodes to be mapped are first elaborated. And then, the reversible circuit synthesis method is presented.

4.4.1 Strategies for Ranking the Ordering of Nodes to Be Mapped

For an inner node v_i, suppose that $var(v_i) = x_l$, and the low and high successors of node v_i are inner nodes v_j

and v_k, respectively. Although local transformations will be applied on function f_{v_i} while mapping node v_i into a reversible cascade. However, as mentioned in Sect. 4.2, due to the sharing of node v_i in the KFDD, the type of local transformations applied on function f_{v_i} can not be determined before the ordering of node v_i to be mapped has been determined. Consequently, in order to be able to provide directive rules for ranking the ordering of nodes to be mapped, the reversible cascades mapped for node v_i by using the original form of the expression of function f_{v_i} are discussed in the following.

Figure 12 presents the reversible cascades mapped for node v_i by using the original form of the expression of function f_{v_i} when the decomposition type of variable x_l is a Shannon or positive Davio decomposition. When the decomposition type of variable x_l is a negative Davio decomposition, the reversible cascade mapped for node v_i is similar to the reversible cascade presented in Fig. 12(b).

For the sake of description, assuming that symbols L_{v_i}, L_{v_j}, L_{v_k}, and L_{x_l} indicate the numerical encoding of the lines presented in Fig. 12 which take 0, f_{v_j}, f_{v_k}, and x_l as the input, respectively. In the following, those lines are called line L_{v_i}, line L_{v_j}, line L_{v_k}, and line L_{x_l}, respectively.

Lines L_{v_j}, L_{v_k}, and L_{v_i} are used to store the results of functions f_{v_j}, f_{v_k}, and f_{v_i}, and paired with nodes v_j, v_k, and v_i, respectively. Line L_{x_l} is used to trace the value of variable x_l before it is reused to store the result of a node function. Since nodes v_j and v_k are mapped before node v_i, and line L_{x_l} is added to the circuit while the first node positioned at level $lev(x_l)$ in the KFDD is mapped. There are $L_{v_i} > L_{v_j}$, $L_{v_i} > L_{v_k}$, $L_{x_l} > L_{v_j}$, $L_{x_l} > L_{v_k}$, and $L_{v_i} > L_{x_l}$.

Let $NNC(v_i)$ indicate the NNC of the reversible cascade mapped for node v_i. For simplicity, $NNC(v_i)$ is also referred to as the NNC for node v_i. By Eq. (1) and Fig. 12(a), the following approximate equations can be derived if the decomposition type of variable x_l is a Shannon decomposition.

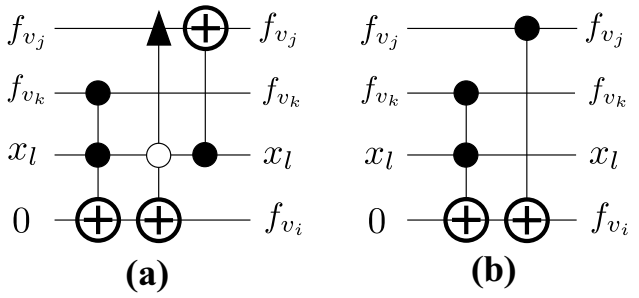

Fig. 12 Reversible gate cascades mapped for node v_i labeled with variable x_l. The decomposition type of variable x_l is a: (**a**) Shannon decomposition, (**b**) positive Davio decomposition

$$\begin{aligned} NNC(v_i) =& NNC(TOF) + NNC(MPP) \\ &+ NNC(CNOT) \\ =& 6(L_{x_l} - L_{v_k} - 1) + 4(L_{v_i} - L_{x_l} - 1) \\ &+ 4(L_{v_i} - L_{v_k} - 1) + 6(L_{x_l} - L_{v_j} - 1) \\ &+ 4(L_{v_i} - L_{x_l} - 1) + 4(L_{v_i} - L_{v_j} - 1) \\ &\propto 4L_{x_l} + 16L_{v_i} - 10(L_{v_j} + L_{v_k}) \\ &\propto 2L_{x_l} + 8L_{v_i} - 5(L_{v_j} + L_{v_k}) \\ =& 2(L_{x_l} - L_{v_i}) + 5(L_{v_i} - L_{v_j}) \\ &+ 5(L_{v_i} - L_{v_k}) \end{aligned} \tag{3}$$

When the decomposition type of variable x_l is a Davio decomposition, there is $f_{v_i} = f_{v_j} \oplus \tilde{x}_l f_{v_k}$, where $\tilde{x}_l \in \{x_l, \bar{x}_l\}$. Then by Eq. (1) and Fig. 12(b), the following approximate equations can be derived.

$$\begin{aligned} NNC(v_i) =& NNC(TOF) + NNC(CNOT) \\ =& 6(L_{x_l} - L_{v_k} - 1) + 4(L_{v_i} - L_{x_l} - 1) \\ &+ 4(L_{v_i} - L_{v_k} - 1) + 2(L_{v_i} - L_{v_j} - 1) \\ &\propto 2L_{x_l} + 10L_{v_i} - 2(L_{v_j} + 5L_{v_k}) \\ &\propto L_{x_l} + 5L_{v_i} - (L_{v_j} + 5L_{v_k}) \\ =& L_{x_l} + 5(L_{v_i} - L_{v_k}) - L_{v_j} \end{aligned} \tag{4}$$

Guided by the NNC metrics as shown in Eqs. (3) and (4), the strategies for ranking the ordering of nodes to be mapped are described in the following.

Above all, as can be seen from Fig. 12 and Eqs. (3) and (4), for reducing the NNC of the reversible cascade mapped for node v_i, line L_{v_i} should be near to lines L_{v_j} and L_{v_k}. Because $lev(v_j) = lev(v_k)$ and $lev(v_i) = lev(v_j) - 1$ in many cases, mapping nodes level by level in a BFS manner helps reduce the NNC compared to mapping nodes in a DFS manner. Consequently, it is better to use LDM rather than DDM to rank the ordering of nodes to be mapped.

Secondly, the weak dependency between nodes positioned at level $lev(x_l)$ in the KFDD is common when the decomposition type of variable x_l is a Davio decomposition. The ordering of those nodes to be mapped is determined by using LDM [23].

However, when the decomposition type of variable x_l is a Shannon decomposition, only a linear node is weakly dependent on other nodes positioned at level $lev(x_l)$ which are not linear nodes. In other words, most of those nodes positioned at level $lev(x_l)$ do not depend on each other according to the definitions of dependency presented in Sect. 3.2, the ordering of those nodes to be mapped can not be determined by using LDM. Accordingly, for reducing the NNC of the resulting circuit, when the decomposition type of variable x_l is a Shannon decomposition, the

ordering of nodes positioned at level $lev(x_l)$ to be mapped should be cautiously considered. For ranking the ordering of those nodes to be mapped, a look-ahead strategy is adopted. That is, the impact of the ordering of nodes positioned at level $lev(x_l)$ to be mapped on the NNC for nodes positioned at level $lev(x_l) - 1$ is considered. Suppose that variable x_t is positioned at level $lev(x_l) - 1$ in the KFDD. Then the look-ahead strategy is as follows.

(i) When the decomposition type of variable x_t is a Davio decomposition, known from Eq. (4), the weight of L_{v_k} with respect to the NNC for nodes positioned at level $lev(x_t)$ in the KFDD is larger than that of L_{v_j}. That is, the larger the value of $L_{v_i} - L_{v_k}$ is, the NNC of the reversible cascades mapped for nodes positioned at level $lev(x_t)$ in the KFDD is also larger. For reducing the NNC, at first, every node v_i positioned at level $lev(x_l)$ is assigned a weight which is indicated by wt_{v_i} and computed as the number of nodes that take v_i as the high successor. And then, nodes positioned at level $lev(x_l)$ are sorted by the value of wt_{v_i} in ascending order. Doing that helps reduce the value of the component $5(L_{v_i} - L_{v_k})$ in Eq. (4), and thus the NNC.

(ii) When the decomposition type of variable x_t is a Shannon decomposition, known from Eq. (3), the weight of L_{v_k} with respect to the NNC for nodes positioned at level $lev(x_t)$ in the KFDD is the same as that of L_{v_j}. As a result, the ordering of nodes positioned at level $lev(x_l)$ is left unchanged.

Once more, while mapping nodes positioned at level $lev(x_l)$ by using $LDM(lev(x_l))$, there may be more than one row in $LDM(lev(x_l))$ being *zero* row. In other words, the mapping conditions for more than one node are satisfied. The ordering of those nodes to be mapped also influences the NNC of the resulting circuit. In that case, one of those nodes is selected to be first mapped by using the best mapping node selecting strategy described as follows. Suppose the nodes whose mapping conditions are satisfied constitute a set denoted by S.

(i) When the decomposition type of variable x_l is a Shannon decomposition, from Eq. (3), it is known that the weight of L_{v_k} with respect to the NNC is the same as that of L_{v_j}. To reduce the NNC, for a node $v_i \in S$ with the non-terminal low successor v_j and the non-terminal high successor v_k, the number of nodes that take node v_j or node v_k as a successor and are positioned at level $lev(x_l)$ is first counted and indicated by ws_{v_i}. Then the node for which the value of ws_{v_i} is the largest is selected from the set S as the node to be first mapped. The reason for doing that is to reduce the component $5(L_{v_i} - L_{v_j})$ and the component $5(L_{v_i} - L_{v_k})$ in Eq. (3).

(ii) When the decomposition type of variable x_l is a Davio decomposition, by Eq. (4), it is known that the weight of L_{v_k} with respect to the NNC is larger than that of L_{v_j}. Suppose the index of the first *zero* row in $LDM(lev(x_l))$ is r, the low successor of node r is node v_j, and nodes in the set S which take node v_j as the low successor constitute a set indicated by N. If there is not any other node not mapped which takes node v_j as the low successor excepting nodes in N, then the node $v_i \in N$ with the high successor v_k for which the paired line L_{v_k} has the largest numerical value is greedily selected from the set N as the node to be first mapped. Otherwise, node r is selected as the node to be first mapped.

Lastly, for nodes positioned at level $lev(x_l)$ which take node v_j as the low successor when the decomposition type of variable x_l is a Davio decomposition, or for linear nodes positioned at level $lev(x_l)$ which take node v_j as a successor when the decomposition type of variable x_l is a Shannon decomposition, the last node to be mapped can reuse line L_{v_j} to store the result of the function represented by this node [3, 23]. For reducing the NNC as well as the number of required lines, if there is a variable node positioned at level $lev(x_l)$, the variable node is selected as the last node to be mapped for using line L_{x_l} to store the result of the function represented by it. Otherwise, the last linear node in the set of nodes that are positioned at level $lev(x_l)$ is selected as the last node to be mapped.

Note that, always selecting a linear node (e.g., linear node v_i which takes node v_j as the low successor) as the last node to be mapped may help reduce the the number of CNOT gates in some cases [3]. However, doing that will increase the NNC. Because node v_i can be mapped only after all the other nodes positioned at level $lev(x_l)$ have been mapped, the distance between line L_{v_j} and line L_{x_l} will be increased.

4.4.2 The Proposed Synthesis Method

Following the strategies described in Sect. 4.4.1, the KFDD based method of synthesizing reversible circuits with reduced NNC is presented as *Algorithm* 1 in the following.

Algorithm 1 KFDD based method of synthesizing reversible circuits with reduced NNC

1: Read the PLA network;
2: Generate the KFDD G and reduce the size of G by using sifting techniques [6];
3: **for** each level l in G **do** ▷ /*Suppose nodes positioned at level l constitute the set $V = \{v_i | 1 \leq i \leq w_l\}$, and the variable labeling nodes positioned at level $l-1$ is x_t.*/
4: Add a line numerically encoded by $L_{var(v_i)}$ to the circuit to trace the value of variable $var(v_i)$;
5: Select the last node to be mapped (suppose it is node u) from the set V which can use line $L_{var(v_i)}$ to store the result of function f_u;
6: Let $V = V \backslash \{u\}$;
7: **if** $DT(var(v_i)) = SD$ and $DT(x_t) = DD$ **then**
8: Sort nodes in V by the value of wt_{v_i} in ascending order;
9: **end if**
10: Create $LDM(l)$ according to V and G;
11: **while** HasValidRow($LDM(l)$) **do**
12: R=GetAllZeroRows($LDM(l)$);
13: **if** $R = \varnothing$ **then**
14: j=GetMaxColumn($LDM(l)$);
15: **else**
16: Select node j from R by using the best mapping node selecting strategy;
17: **end if**
18: Map $Node(j)$ into a reversible cascade;
19: SetZerosColumn($LDM(l)$, j);
20: $LDM(l)[j, j] = -1$;
21: **end while**
22: Map node u into a reversible cascade by using line $L_{var(v_i)}$ to store the result of function f_u;
23: **end for**
24: **for** each root node v of G **do**
25: **if** v has an incoming complemented edge **then**
26: Add a NOT gate for node v;
27: **end if**
28: **end for**

In Step 7 of *Algorithm* 1, '*DT*' implies decomposition type, '*SD*' and '*DD*' indicate Shannon and Davio decomposition, respectively. In Step 8, the implication of wt_{vi} has been described in Sect. 4.4.1. In Step 11, in a level dependency matrix $LDM(l)$, if there is a row which does not have an element with the value of -1, 'HasValidRow($LDM(l)$)' returns *true*, otherwise it returns *false*. In Step 12, 'GetAllZeroRows' returns the indices of the *zero* rows in $LDM(l)$ which constitute a set R. In Step 14, 'GetMaxColumn' returns the index of the column in $LDM(l)$ which does not have an element with the value of -1 and has a maximal number of non-zero elements [23]. In Step 18, the mapping technique by performing local transformations on node functions and by using NCT and MPP gates proposed in Ref. [3] is used. In Step 19, the elements of the j-th column in $LDM(l)$ take the value 0 [23]. In Step 20, the value of the element d_{jj} in $LDM(l)$ is set to -1 [23]. In Step 22, the strategy proposed in Ref. [3] is used.

Subsequently, the KFDD presented in Fig. 10(a) is considered as a running example for demonstrating *Algorithm* 1. Remember that, the symbols L_{x_2}, L_{x_3}, L_{v_2}, L_{v_4}, L_{x_1}, and L_{v_7} presented in Fig. 11(b) indicate the numerical encoding of the circuit lines from the topmost line to the undermost line.

Example 5 The nodes of the KFDD presented in Fig. 10(a) are partitioned into 3 levels. As can be seen from Fig. 10(a), $lev(x_2) = 3$, $lev(x_3) = 2$, and $lev(x_1) = 1$. From level 3 to level 1, the nodes will be mapped into reversible cascades for generating a reversible circuit.

1) Map nodes positioned at level 3. The decomposition type of x_2 is a negative Davio decomposition. Node v_1 is positioned at level 3. That is, $V = \{v_1\}$. A line encoded by $L_{x_2} = 1$ (the topmost horizontal line in the circuit presented in 11(b) which takes x_2 as the input) is added to the circuit in Step 4. Because node v_1 is a variable node, it is selected as the last node to be mapped in Step 5. That is, $u = v_1$. Because $V = V \backslash \{u\} = \varnothing$, the algorithm goes to Step 22. Since $f_{v_1} = x_2$, there is no need to add any gate in this round.
2) Map nodes positioned at level 2. The decomposition type of x_3 is a Shannon decomposition. Nodes v_2, v_3, and v_4 are positioned at level 2. That is, $V = \{v_2, v_3, v_4\}$. A line encoded by $L_{x_3} = 2$ (the second horizontal line in the circuit presented in Fig. 11(b) which takes x_3 as the input) is added to the circuit in Step 4. Since $f_{v_3} = x_3 \oplus f_{v_1}$, node v_3 is a linear node. Because node v_3 is the only linear or variable node positioned at level 2, it is selected as the last node to be mapped in Step 5. That is, $u = v_3$. After that, $V = \{v_2, v_4\}$. Because $DT(x_3) = SD$ and $DT(x_1) = DD$, nodes in V are sorted by the value of wt_{v_i} in ascending order in Step 8. Since $wt_{v_2} = 0$ and $wt_{v_4} = 2$, $V = \{v_2, v_4\}$ is resulted. In Step 10, the LDM for level 2 is created as follows

$$LDM(2) = \begin{array}{c} \\ v_2 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cc} v_2 & v_4 \end{array} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}.$$

In Step 12, there are $R = \{1, 2\}$, $Node(1) = v_2$, and $Node(2) = v_4$. Node v_2 has the non-terminal high successor v_1. Whereas node v_4 has the non-terminal low

successor v_1. Because there are two nodes (i.e., nodes v_2 and v_4) in the set V which take node v_1 as a successor, $ws_{v_2} = 2$ and $ws_{v_4} = 2$. Consequently, v_2 is selected as $Node(j)$ in Step 16 by using the best mapping node selecting strategy. In Step 18, known that $f_{v_2} = 1 \oplus x_3\bar{f}_{v_1}$, an ancilla line encoded by $L_{v_2} = 3$ (the third horizontal line in the circuit presented in Fig. 11(b)) which has the initial value of 1 is first added to the circuit, and then node v_2 is mapped into the reversible cascade indicated by the symbol 'v_2' as shown in Fig. 11(b). The reversible cascade is composed of an inverse Peres gate and a CNOT gate. The inverse Peres gate realizes function f_{v_2}. The CNOT gate is used to recover the value on line L_{x_2}, because node v_1 has two parent nodes not mapped (i.e., nodes v_3 and v_4). Thereafter, $LDM(2)$ becomes

$$LDM(2) = \begin{array}{c} \\ v_2 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cc} v_2 & v_4 \end{array} \\ \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} \end{array}.$$

The algorithm goes to Step 11. Because $R = \{2\}$ and $f_{v_4} = \bar{x}_3 f_{v_1}$, an ancilla line encoded by $L_{v_4} = 4$ (the fourth horizontal line in the circuit presented in Fig. 11(b)) which has the initial value of 0 is first added to the circuit, and then node v_4 is mapped into the reversible cascade indicated by the symbol 'v_4' as shown in Fig. 11(b). The reversible cascade is composed of a Peres gate with the first control negated and a CNOT gate. The Peres gate with the first control negated realizes function f_{v_4}. The CNOT gate is used to recover the value on line L_{x_2}, because node v_1 has one parent node not mapped (i.e., node v_3). In Step 22, known that $f_{v_3} = x_3 \oplus f_{v_1}$, node v_3 is mapped into a CNOT gate using line L_{x_3} as the target line which is indicated by the symbol 'v_3' as shown in Fig. 11(b).

3) Map nodes positioned at level 1. The decomposition type of x_1 is a positive Davio decomposition. Node v_5, v_7, and v_6 are positioned at level 1. That is, $V = \{v_5, v_7, v_6\}$. A line encoded by $L_{x_1} = 5$ (the fifth horizontal line in the circuit presented in Fig. 11(b) which takes x_1 as the input) is added to the circuit in Step 4. Known from Fig. 10(a), there is not a linear or variable node in V. Furthermore, the decomposition type of x_1 is a positive Davio decomposition. Subsequently, the algorithm goes to Step 10, the LDM for level 1 is created as follows

$$LDM(1) = \begin{array}{c} \\ v_5 \\ v_7 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccc} v_5 & v_7 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}.$$

In Step 12, there are $R = \{1, 2, 3\}$, $Node(1) = v_5$, $Node(2) = v_7$, and $Node(3) = v_6$. The index of the first *zero* row in $LDM(1)$ is 1. As can be seen from Fig. 10(a), nodes v_5 and v_7 share the low successor v_2. That is, $N = \{v_5, v_7\}$. The high successors of nodes v_5 and v_7 are nodes v_1 and v_4, respectively. In addition, there is not any other node not mapped which takes v_2 as the low successor excepting nodes in N, and there is $L_{v_4} > L_{v_1}$ where $L_{v_1} = L_{x_2}$. Thus, by the best mapping node selecting strategy, node v_7 is greedily selected as the node to be first mapped in Step 16. In Step 18, known that $f_{v_7} = f_{v_2} \oplus x_1\bar{f}_{v_4}$, an ancilla line encoded by $L_{v_7} = 6$ (the sixth horizontal line in the circuit presented in Fig. 11(b)) which has the initial value of 0 is first added to the circuit, and then node v_7 is mapped into the reversible cascade indicated by the symbol 'v_7' as shown in Fig. 11(b). The reversible cascade is composed of an inverse Peres gate and two CNOT gates. The inverse Peres gate realizes the function $h_1 = x_1\bar{f}_{v_4} \oplus 0$. The first CNOT gate is used to recover the value on line L_{v_4}. The second CNOT gate realizes the function $f_{v_7} = f_{v_2} \oplus h_1$. Thereafter, $LDM(1)$ becomes

$$LDM(1) = \begin{array}{c} \\ v_5 \\ v_7 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccc} v_5 & v_7 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}.$$

Subsequently, according to the functions $f_{v_5} = x_1 f_{v_1} \oplus f_{v_2}$ and $f_{v_6} = x_1 f_{v_4} \oplus f_{v_3}$, nodes v_5 and v_6 are both mapped into a Peres gate indicated by the symbol 'v_5' and the symbol 'v_6' as shown in Fig. 11(b), respectively. Because neither of nodes v_2 and v_3 have parent nodes not mapped, line L_{v_2} and line L_{x_3} are reused to store the results of functions f_{v_5} and f_{v_6}, respectively. In addition, since neither of nodes v_1 and v_4 have parent nodes not mapped, the values on line L_{x_2} and line L_{v_4} do not need to be recovered, and the outputs of line L_{x_2} and line L_{v_4} become garbage outputs indicated by the symbol 'g'.

4) Add a NOT gate for every root node of G which has an incoming complemented edge. Since all the nodes in the KFDD presented in Fig. 10(a) have been mapped into reversible cascades, the algorithm goes to Step 24. The KFDD presented in Fig. 10(a) has 3 root nodes which are nodes v_5, v_6, and v_7. As can be seen from Fig. 10(a), nodes v_5 and v_6 both have incoming complemented edges. Consequently, two NOT gates are added on lines L_{v_2} and L_{x_3}, respectively. Thereafter, the circuit is generated completely, the algorithm terminates.

According to the circuit presented in Fig. 11(b), by Eq. (1), the NNC of the reversible gate cascade mapped for every node can be computed as follows.

$NNC(v_1) = 0, NNC(v_2) = 4, NNC(v_4) = 12, NNC(v_3) = 0$, $NNC(v_7) = 8$, $NNC(v_5) = 20$, and $NNC(v_6) = 12$.

The NNC of the circuit presented in Fig. 11(b) is 56 which is the cumulative sum of the NNC of the reversible gate cascades in the circuit. In addition, the circuit is composed

of 5 MPP gates, 5 CNOT gates, and 2 NOT gates. Consequently, the T-count of the circuit is $5 \times 7 = 35$, which is the cumulative sum of the T-count of the 5 MPP gates. Whereas the T-depth of the circuit is estimated as $5 \times 3 = 15$, which is the cumulative sum of the T-depth of the 5 MPP gates. Since the MPP gates all consist of 15 quantum gates excepting the or-Peres gate, the number of quantum gates of the circuit presented in Fig. 11(b) before SWAP insertion is $\#QG = 5 \times 15 + 5 + 2 = 82$. Let #QG_A indicate the number of quantum gates of a circuit after SWAP insertion. Then there is $\#QG\_A = \#QG + 3 \times NNC = 82 + 3 \times 56 = 250$ for the circuit presented in Fig. 11(b).

The costs of the circuits presented Fig. 10(b) and Fig. 11 are summarized in Table 1. As can be seen from Table 1, the circuit as shown in Fig. 11(b) which is achieved with *Algorithm* 1 has lower NNC and lower #QG_A compared to two other circuits as shown in Fig. 10(b) and Fig. 11(a).

5 Experimental Evaluations

Algorithm 1 has been implemented in C++ on the top of Revkit [30]. With regard to the KFDD data structure, the *PUMA* package [6] has been used. The experimental evaluations have been carried out on an Intel Core i7-10700 Processor with 32 GB of main memory running Ubuntu 16.04 64bit OS.

5.1 Evaluating the Effect of the Proposed Method On NNC

For reducing the NNC, the proposed synthesis method uses strategies governed by NNC metrics to rank the ordering of nodes in the KFDD to be mapped. In order to evaluate the effect of the proposed method on the NNC of the synthesized reversible circuits, two other algorithms named by *FDD+DDM* and *KFDD+DDM* have been designed. *FDD+DDM* synthesizes a reversible circuit from an FDD and uses the DDM to rank the ordering of nodes in the FDD to be mapped as described in Ref. [23]. Similarly, *KFDD+DDM* synthesizes a reversible circuit from a KFDD and also uses the DDM to rank the ordering of nodes in the KFDD to be mapped as described in Ref. [23]. While mapping a node into a reversible cascade, algorithms *FDD+DDM* and *KFDD+DDM* both apply local transformations on node functions [3] and both use the NCT and MPP gates to generate reversible cascades. Similarly to *Algorithm* 1, the KFDDs used by *KFDD+DDM* are generated by using the *PUMA* package and sifting techniques [6]. As FDDs are special kinds of the KFDD, the FDDs used by *FDD+DDM* are also generated by using the *PUMA* package and sifting techniques [6].

Algorithms *FDD+DDM*, *KFDD+DDM*, and *Algorithm* 1 have been used to synthesize reversible circuits for 31 functions available in the RevLib benchmark suite [25] which were also used by Stojković et al. [23] to evaluate their FDD-based synthesis method or used by Abdalhaq et al. [1] to evaluate their BDD-based synthesis method. In the following, Table 2 presents the results with respect to #lines, #QG, NNC, #QG_A, and #nodes. Table 3 presents the percentage reduction (improvement) in those results achieved by comparing *KFDD+DDM* to *FDD+DDM* and by comparing *Algorithm* 1 to *KFDD+DDM*. In Table 2, the column indicated by '#in/#out' gives the number of inputs and outputs of the functions. Note that #nodes indicates the number of nodes in the FDD or KFDD, #QG indicates the number of Clifford+T quantum gates before SWAP insertion, and #QG_A indicates the number of Clifford+T quantum gates after SWAP insertion. Because a SWAP-gate is commonly realized using 3 CNOT gates, there is $\#QG\_A = \#QG + 3 \times NNC$. In addition, Fig. 13 illustrates the improvement results of those functions presented in Table 2 for which *KFDD+DDM* achieves results different from *FDD+DDM*. Figure 14 illustrates the improvement results of those functions presented in Table 2 for which *Algorithm* 1 achieves results different from *KFDD+DDM*.

As can be seen from Tables 2 and 3, *KFDD+DDM* can achieve #lines, #QG, NNC, and #QG_A not inferior to *FDD+DDM* for all of the functions excepting function *mini- alu*_84 and function *mod5adder*_66. Compared to *FDD+DDM*, *KFDD+DDM* increases the NNC, #QG, and #QG_A while not increasing the number of lines for function *mini- alu*_84. Whereas for function *mod5adder*_66, *KFDD+DDM* only slightly increases the NNC and #QG_A. Note that, compared to *FDD+DDM*, there are 7 cases where *KFDD+DDM* reduces #lines, #QG, NNC, and #QG_A. As can be obviously observed from Table 3 and Fig. 13, this is because the KFDD is more compact than the FDD. In other words, for a given function, the KFDD has the number of nodes (#nodes) less than the FDD. For function *mini- alu*_84, although the number of nodes of the FDD is the same as that of the KFDD, the KFDD is not

Table 1 The costs of the circuits presented in Fig. 10(b) and Fig. 11

Circuit	#lines	T-depth	T-count	#QG	NNC	#QG_A
Figure 10(b)	6	15	35	83	92	359
Figure 11(a)	6	15	35	82	64	274
Figure 11(b)	6	15	35	82	56	250

Table 2 The results wrt. #lines, #QG, NNC, #QG_A, and #nodes

Benchmark specification		*FDD+DDM*					*KFDD+DDM*					*Algorithm* 1			
Function	#in/#out	#lines	#QG	NNC	#QG_A	#nodes	#lines	#QG	NNC	#QG_A	#nodes	#lines	#QG	NNC	#QG_A
4mod5_8	4/1	5	34	16	82	4	5	34	16	82	4	5	34	16	82
9symml_91	9/1	12	325	652	2,281	26	12	325	652	2,281	26	12	325	626	2,203
alu_9	5/1	8	83	76	311	7	8	83	76	311	7	8	83	76	311
apex2_101	39/3	356	8,468	602,230	1,815,158	538	283	6,006	316,280	954,846	368	288	5,888	118,876	362,516
apex5_104	117/88	358	7,095	354,550	1,070,745	444	335	6,839	350,578	1,058,573	426	336	6,849	293,814	888,291
bw_116	5/28	67	1,430	19,846	60,968	91	67	1,430	19,846	60,968	91	67	1,432	15,826	48,910
cordic_138	23/2	41	645	2,110	6,975	43	41	645	2,110	6,975	43	42	645	1,844	6,177
cycle10_2_61	12/12	23	197	148	641	24	23	197	148	641	24	21	195	136	603
decod24_10	2/4	6	68	64	260	5	6	68	64	260	5	6	68	64	260
e64_149	65/65	192	2,033	2,118	8,387	128	192	2,033	2,118	8,387	128	192	2,033	1,030	5,123
ex5p_154	8/63	197	4,283	214,180	646,823	267	191	3,907	185,914	561,649	243	194	3,934	91,664	278,926
ham15_30	15/15	42	483	4,380	13,623	67	42	483	4,380	13,623	67	42	483	2,408	7,707
ham7_29	7/7	17	196	480	1,636	23	17	196	480	1,636	23	17	196	402	1,402
hwb5_13	5/5	17	496	1,676	5,524	35	17	496	1,676	5,524	35	17	495	1,850	6,045
hwb6_14	6/6	43	1,188	10,772	33,504	81	41	1,093	9,258	28,867	72	43	1,094	9,788	30,458
hwb7_15	7/7	51	1,797	20,304	62,709	118	51	1,797	20,304	62,709	118	52	1,796	21,648	66,740
hwb8_64	8/8	69	2,901	47,168	144,405	189	69	2,901	47,168	144,405	189	66	2,899	51,770	158,209
hwb9_65	9/9	220	7,743	412,036	1,243,851	494	171	4,959	168,784	511,311	305	172	4,943	137,336	416,951
mini–alu_84	4/2	8	129	136	537	10	8	130	166	628	10	8	130	174	652
mod5adder_66	6/6	20	399	2,058	6,573	33	20	399	2,060	6,579	33	17	397	1,458	4,771
pdc_191	16/40	625	17,759	3,136,236	9,426,467	1,110	419	9,910	1,118,880	3,366,550	593	450	9,862	388,650	1,175,812
plus127mod8192_78	13/13	24	196	44	328	24	24	196	44	328	24	24	196	44	328
plus63mod4096_79	12/12	22	179	40	299	22	22	179	40	299	22	22	179	40	299
plus63mod8192_80	13/13	24	197	44	329	24	24	197	44	329	24	24	197	44	329
rd53_68	5/3	8	133	96	421	13	8	133	96	421	13	8	133	96	421
rd73_69	7/3	10	231	316	1,179	21	10	231	316	1,179	21	10	231	316	1,179
rd84_70	8/4	15	342	612	2,178	29	15	342	612	2,178	29	15	342	628	2,226
spla_202	16/46	551	14,903	2,347,384	7,057,055	922	428	9,579	1,088,888	3,276,243	574	441	9,561	390,440	1,180,881
sym6_63	6/1	10	196	262	982	14	10	196	262	982	14	10	196	290	1,066
sym9_71	9/1	12	325	652	2,281	26	12	325	652	2,281	26	12	325	626	2,203
xor5_195	5/1	5	6	0	6	5	5	6	0	6	5	5	6	0	6

Table 3 Improvement (%) results of *KFDD+DDM* and *Algorithm* 1

Function	*KFDD+DDM* vs. *FDD+DDM*					*Algorithm* 1 vs. *KFDD+DDM*			
	#lines	#QG	NNC	#QG_A	#nodes	#lines	#QG	NNC	#QG_A
4mod5_8	0	0	0	0	0	0	0	0	0
9symml_91	0	0	0	0	0	0	0	3.99	3.42
alu_9	0	0	0	0	0	0	0	0	0
apex2_101	20.51	29.07	47.48	47.40	31.60	–1.77	1.96	62.41	62.03
apex5_104	6.42	3.61	1.12	1.14	4.05	–0.30	–0.15	16.19	16.09
bw_116	0	0	0	0	0	0	–0.14	20.26	19.78
cordic_138	0	0	0	0	0	–2.44	0	12.61	11.44
cycle10_2_61	0	0	0	0	0	8.70	1.02	8.11	5.93
decod24_10	0	0	0	0	0	0	0	0	0
e64_149	0	0	0	0	0	0	0	51.37	38.92
ex5p_154	3.05	8.78	13.20	13.17	8.99	–1.57	–0.69	50.70	50.34
ham15_30	0	0	0	0	0	0	0	45.02	43.43
ham7_29	0	0	0	0	0	0	0	16.25	14.30
hwb5_13	0	0	0	0	0	0	0.20	–10.38	–9.43
hwb6_14	4.65	8.00	14.05	13.84	11.11	–4.88	–0.09	–5.72	–5.51
hwb7_15	0	0	0	0	0	–1.96	0.06	–6.62	–6.43
hwb8_64	0	0	0	0	0	4.35	0.07	–9.76	–9.56
hwb9_65	22.27	35.96	59.04	58.89	38.26	–0.58	0.32	18.63	18.45
mini–alu_84	0	–0.78	–22.06	–16.95	0	0	0	–4.82	–3.82
mod5adder_66	0	0	–0.10	–0.09	0	15.00	0.50	29.22	27.48
pdc_191	32.96	44.2	64.32	64.29	46.58	–7.40	0.48	65.26	65.07
plus127mod8192_78	0	0	0	0	0	0	0	0	0
plus63mod4096_79	0	0	0	0	0	0	0	0	0
plus63mod8192_80	0	0	0	0	0	0	0	0	0
rd53_68	0	0	0	0	0	0	0	0	0
rd73_69	0	0	0	0	0	0	0	0	0
rd84_70	0	0	0	0	0	0	0	–2.61	–2.20
spla_202	22.32	35.72	53.61	53.57	37.74	–3.04	0.19	64.14	63.96
sym6_63	0	0	0	0	0	0	0	–10.69	–8.55
sym9_71	0	0	0	0	0	0	0	3.99	3.42
xor5_195	0	0	0	0	0	0	0	0	0
Average	3.62	5.31	7.44	7.59	5.75	0.13	0.12	13.47	12.86

same as the FDD. For that function, synthesizing a reversible circuit using the KFDD instead of the FDD significantly increases the NNC and #QG_A. For the functions presented in Tables 2 and 3, compared to *FDD+DDM*, the average reductions achieved by *KFDD+DDM* in #lines, #QG, NNC, and #QG_A are 3.62%, 5.31%, 7.44%, and 7.59%, respectively. As can be seen, while synthesizing reversible circuits by mapping a node into a reversible cascade, using the KFDD instead of the FDD helps reduce the NNC, also the number of lines, the number of Clifford+T gates before and after SWAP insertion.

Algorithm 1 and *KFDD+DDM* both synthesize a reversible circuit from a KFDD and the KFDDs used by them are both generated by using the *PUMA* package and sifting techniques [6]. That is, for a given function, *Algorithm* 1 and *KFDD+DDM* synthesize reversible circuits from the same KFDD. Subsequently, *Algorithm* 1 is compared to *KFDD+DDM*.

As can be seen from Tables 2 and 3, out of the 31 functions, there are 15 cases where *Algorithm* 1 outperforms *KFDD+DDM* on NNC as well as #QG_A, and 9 cases where *Algorithm* 1 achieves the NNC and #QG_A both same as *KFDD+DDM*. In the other 7 cases, compared to algorithm *KFDD+DDM*, although *Algorithm* 1 increases both the NNC and #QG_A, the NNC and #QG_A are increased by no more than 11% and 10%, respectively. Whereas, compared to *KFDD+DDM*, *Algorithm* 1 reduces the NNC and #QG_A by up to 65.26% and 65.07% (function *pdc*_191), respectively. Moreover, there are 12 cases where *Algorithm* 1 reduce both the NNC and #QG_A by no less than 11%. On the other hand, *Algorithm* 1 achieves the same #QG as *KFDD+DDM* in 18 cases. In the other 13 cases,

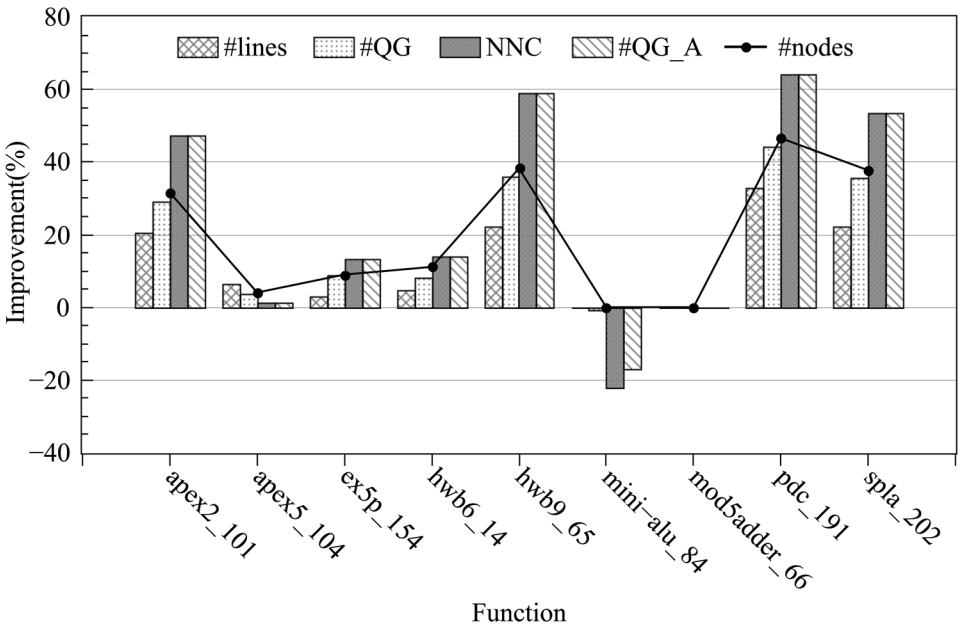

Fig. 13 Improvement results of *KFDD+DDM* wrt. *FDD+DDM*

Algorithm 1 slightly increases (by no more than 0.69%) or decreases (by no more than 1.96%) the #QG compared to *KFDD+DDM*. Consequently, it can be concluded that, compared to *KFDD+DDM*, the reduction in the number of Clifford+T gates after SWAP insertion which is achieved by *Algorithm* 1 is mainly contributed to the reduction in NNC. Compared to *KFDD+DDM*, the average reductions achieved by *Algorithm* 1 in NNC and #QG_A are 13.47% and 12.86%, respectively.

In addition, as can be seen from Table 3 and Fig. 14, compared to *KFDD+DDM*, there are 9 cases where *Algorithm* 1 increases the number of lines (#lines). Increasing the number of lines may help reduce the NNC and #QG_A. In the 15 cases where *Algorithm* 1 reduces both the NNC and #QG_A, there are 7 cases where *Algorithm* 1 increases the number of lines. However, it is worth noting that there are 6 cases where *Algorithm* 1 achieves the same #lines as *KFDD+DDM*, and 2 cases where *Algorithm* 1 even reduces the number of lines. For the functions presented in Table 2, compared to *KFDD+DDM*, the average reduction achieved by *Algorithm* 1 in the number of lines is 0.13%.

Table 4 presents the T-count and the T-depth of the reversible circuits obtained with algorithms *FDD+DDM*, *KFDD+DDM*, and *Algorithm* 1. In Table 4, the columns indicated by 'Imp_1' show the percentage reduction in T-depth or T-count achieved by comparing *KFDD+DDM* to *FDD+DDM*. Whereas the columns

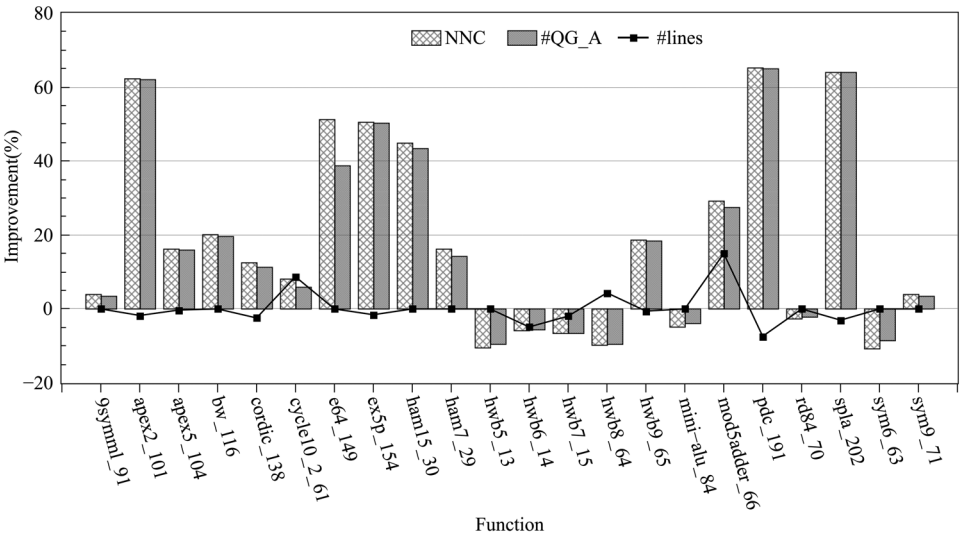

Fig. 14 Improvement results of *Algorithm* 1 wrt. *KFDD+DDM*

Table 4 The results wrt. T-depth and T-count

Function	*FDD+DDM*		*KFDD+DDM*				*Algorithm* 1			
	T-depth	T-count	T-depth	T-count	Imp_1(%)		T-depth	T-count	Imp_2(%)	
					T-depth	T-count			T-depth	T-count
4mod5_8	6	14	6	14	0	0	6	14	0	0
9symml_91	60	140	60	140	0	0	60	140	0	0
alu_9	15	35	15	35	0	0	15	35	0	0
apex2_101	1,611	3,759	1,134	2,646	29.61	29.61	1,110	2,590	2.12	2.12
apex5_104	1,314	3,066	1,260	2,940	4.11	4.11	1,263	2,947	–0.24	–0.24
bw_116	258	602	258	602	0	0	258	602	0	0
cordic_138	120	280	120	280	0	0	120	280	0	0
cycle10_2_61	33	77	33	77	0	0	33	77	0	0
decod24_10	12	28	12	28	0	0	12	28	0	0
e64_149	381	889	381	889	0	0	381	889	0	0
ex5p_154	792	1,848	720	1,680	9.09	9.09	726	1,694	–0.83	–0.83
ham15_30	81	189	81	189	0	0	81	189	0	0
ham7_29	33	77	33	77	0	0	33	77	0	0
hwb5_13	90	210	90	210	0	0	90	210	0	0
hwb6_14	222	518	201	469	9.46	9.46	201	469	0	0
hwb7_15	333	777	333	777	0	0	333	777	0	0
hwb8_64	537	1,253	537	1,253	0	0	537	1,253	0	0
hwb9_65	1,452	3,388	909	2,121	37.40	37.40	906	2,114	0.33	0.33
mini–alu_84	24	56	24	56	0	0	24	56	0	0
mod5adder_66	72	168	72	168	0	0	72	168	0	0
pdc_191	3,324	7,756	1,830	4,270	44.95	44.95	1,821	4,249	0.49	0.49
plus127mod8192_78	33	77	33	77	0	0	33	77	0	0
plus63mod4096_79	30	70	30	70	0	0	30	70	0	0
plus63mod8192_80	33	77	33	77	0	0	33	77	0	0
rd53_68	24	56	24	56	0	0	24	56	0	0
rd73_69	42	98	42	98	0	0	42	98	0	0
rd84_70	63	147	63	147	0	0	63	147	0	0
spla_202	2,760	6,440	1,767	4,123	35.98	35.98	1,764	4,116	0.17	0.17
sym6_63	36	84	36	84	0	0	36	84	0	0
sym9_71	60	140	60	140	0	0	60	140	0	0
xor5_195	0	0	0	0	0	0	0	0	0	0
Average	446.81	1042.55	328.94	767.52	5.50	5.50	327.97	765.26	0.07	0.07

indicated by 'Imp_2' give the percentage reduction in T-depth or T-count achieved by comparing *Algorithm* 1 to *KFDD+DDM*. It can be observed from Table 4 that, compared to *FDD+DDM*, there are 7 cases where algorithm *KFDD+DDM* reduces both the T-depth and the T-count. For the other 24 cases, *KFDD+DDM* achieves the T-depth and the T-count both same as *FDD+DDM*. Compared to *FDD+DDM*, the average reductions achieved by *KFDD+DDM* in T-depth and T-count are both 5.50%. As can be seen, while synthesizing reversible circuits by mapping a node into a reversible cascade, using KFDDs is also better than using FDDs in terms of the T-depth and T-count.

On the other hand, since *KFDD+DDM* and *Algorithm* 1 both synthesize a reversible circuit from a KFDD, the resulting T-depth or T-count of the two algorithms are quite close to each other.

It can be concluded from above analyses that, the strategies presented for ranking the ordering of nodes to be mapped for reducing NNC which are used by the proposed synthesis method is effective. While synthesizing reversible circuits using the KFDD, the proposed method helps reduce the NNC as well as the number of Clifford+T gates after SWAP insertion and has a slight impact on the resulting #lines, T-depth, T-count, and the number of Clifford+T gates before SWAP insertion.

5.2 Comparison to Prior Synthesis Methods Based On FDD or BDD

Stojković et al. [23] used FDDs to synthesize reversible circuits. Whereas, Abdalhaq et al. [1] used BDDs to synthesize reversible circuits. They did not consider the reduction of the NNC in their works. However, for the completeness of this work and due to the fact that BDDs and FDDs are both special kinds of the KFDD, we compare the proposed synthesis method to their methods in this section.

Stojković et al. [23] and Abdalhaq et al. [1] both used the NCV-cost to measure the quantum cost of reversible circuits. In other words, they considered the NCV quantum realizations for reversible circuits. In the following, *Algorithm* 1 is compared to their methods by using the NCV-cost metric. The results are listed in Tables 5 and 6.

In Tables 5 or 6, *OFPFDD+LDM* indicates the algorithm in Ref. [23] which synthesizes circuits from the optimal fixed-polarity FDDs by using LDMs. Whereas *OFPFDD+DDM* indicates the algorithm in Ref. [23] which synthesizes circuits from the optimal fixed-polarity FDDs by using DDMs. *BDD+GA* indicates the algorithm in Ref. [1] which synthesizes circuits from BDDs by using a genetic algorithm to search the optimal BDDs. The columns indicated by 'runtime' list the runtime in CPU seconds of those algorithms. Table 6 shows the percentage reduction (improvement) in #lines and NCV-cost achieved by comparing *Algorithm* 1 to *OFPFDD+DDM* or *BDD+GA*.

Table 5 Comparison with the results of FDD based and BDD based methods

Function	*OFPFDD+LDM* [23]		*OFPFDD+DDM* [23]		*BDD+GA* [1]			*Algorithm* 1		
	#lines	NCV-cost	#lines	NCV-cost	#lines	NCV-cost	runtime	#lines	NCV-cost	runtime
4mod5_8	5	18	5	18	7	24	< 0.01	5	12	< 0.01
9symml_91	–	–	–	–	27	206	0.02	12	104	< 0.01
alu_9	8	29	8	28	7	29	< 0.01	8	26	< 0.01
apex2_101	–	–	–	–	282	2747.78	6.53	288	1,735	0.81
apex5_104	–	–	–	–	1,015	9839.09	5.31	336	2,124	0.29
bw_116	72	619	74	619	86	931.34	0.01	67	458	< 0.01
cordic_138	–	–	–	–	49	311.44	0.33	42	196	< 0.01
cycle10_2_61	97	552	97	552	–	–	–	21	74	0.01
decod24_10	7	23	6	23	6	27	< 0.01	6	23	< 0.01
e64_149	–	–	–	–	192	886	2.21	192	636	0.03
ex5p_154	231	1,808	225	1,803	206	1,970	0.03	194	1,186	< 0.01
ham15_30	–	–	–	–	42	270	0.44	42	176	0.26
ham7_29	16	85	15	85	21	141	0.01	17	71	< 0.01
hwb5_13	18	199	18	196	27	268	0.01	17	160	< 0.01
hwb6_14	35	382	34	378	44	503	0.01	43	354	< 0.01
hwb7_15	56	677	57	678	76	910	0.02	52	564	< 0.01
hwb8_64	78	1,109	85	1,087	114	1,552	0.04	66	912	< 0.01
hwb9_65	–	–	–	–	169	2,239	0.07	172	1,617	< 0.01
mini-alu_84	8	43	8	43	10	60	< 0.01	8	41	< 0.01
mod5adder_66	21	151	20	150	29	301.52	0.01	17	130	< 0.01
pdc_191	–	–	–	–	619	6598.02	0.71	450	2,977	0.05
plus127mod8192_78	24	73	24	73	25	98	0.09	24	75	0.04
plus63mod4096_79	22	66	22	66	23	89	0.04	22	69	0.02
plus63mod8192_80	–	–	–	–	25	97	0.08	24	76	0.03
rd53_68	8	44	8	44	13	98	< 0.01	8	45	< 0.01
rd73_69	10	76	10	76	25	217	0.01	10	77	< 0.01
rd84_70	15	112	15	112	34	304	0.02	15	111	< 0.01
spla_202	–	–	–	–	482	5960.72	0.50	441	2,873	0.04
sym6_63	10	69	10	69	14	93	0.01	10	62	< 0.01
sym9_71	12	106	12	106	27	206	0.02	12	104	< 0.01
xor5_195	–	–	–	–	6	8	< 0.01	5	6	< 0.01

–: the data is not available

Table 6 Improvement (%) results of *Algorithm* 1 wrt. *OFPFDD+DDM* and *BDD+GA*

Function	*Algorithm* 1 vs. *OFPFDD+DDM*		*Algorithm* 1 vs. *BDD+GA*	
	#lines	NCV-cost	#lines	NCV-cost
4mod5_8	0	33.33	28.57	50.00
9symml_91	–	–	55.56	49.51
alu_9	0	7.14	–14.29	10.34
apex2_101	–	–	–2.13	36.86
apex5_104	–	–	66.90	78.41
bw_116	9.46	26.01	22.09	50.82
cordic_138	–	–	14.29	37.07
cycle10_2_61	78.35	86.59	–	–
decod24_10	0	0	0	14.81
e64_149	–	–	0	28.22
ex5p_154	13.78	34.22	5.83	39.80
ham15_30	–	–	0	34.81
ham7_29	–13.33	16.47	19.05	49.65
hwb5_13	5.56	18.37	37.04	40.30
hwb6_14	–26.47	6.35	2.27	29.62
hwb7_15	8.77	16.81	31.58	38.02
hwb8_64	22.35	16.10	42.11	41.24
hwb9_65	–	–	–1.78	27.78
mini–alu_84	0	4.65	20.00	31.67
mod5adder_66	15.00	13.33	41.38	56.89
pdc_191	–	–	27.30	54.88
plus127mod8192_78	0	–2.74	4.00	23.47
plus63mod4096_79	0	–4.55	4.35	22.47
plus63mod8192_80	–	–	4.00	21.65
rd53_68	0	–2.27	38.46	54.08
rd73_69	0	–1.32	60.00	64.52
rd84_70	0	0.89	55.88	63.49
spla_202	–	–	8.51	51.80
sym6_63	0	10.14	28.57	33.33
sym9_71	0	1.89	55.56	49.51
xor5_195	–	–	16.67	25.00
Average	5.67	14.07	22.39	40.33

As can be observed from Table 5, the results of *OFPFDD+LDM* and *OFPFDD+DDM* are quite close to each other. This is because the two algorithms both synthesize reversible circuits from the optimal fixed-polarity FDDs. Consequently, *Algorithm* 1 is only compared to *OFPFDD+DDM* and *BDD+GA* in the following.

As can be seen from Tables 5 and 6, there are 14 cases where *Algorithm* 1 achieves results not inferior to *OFPFDD+DDM* in terms of #lines and NCV-cost. Compared to *OFPFDD+DDM* [23], there are only 2 cases (functions *ham*7_29 and *hwb*6_14) where *Algorithm* 1 only increases the number of lines, and 4 cases (functions *plus*127*mod*8192_78, *plus*63*mod*4096_79, *rd*53_68, and *rd*73_69) where *Algorithm* 1 only increases the NCV-cost. Furthermore, *Algorithm* 1 reduces the number of lines and the NCV-cost by up to 78.35% and 86.59% (function *cycle*10_2_61), respectively. The average reductions achieved by *Algorithm* 1 in the number of lines and NCV-cost are 5.67% and 14.07%, respectively.

On the other hand, compared to *BDD+GA* [1], there are only 3 cases (functions *alu*_9, *apex*2_101, and *hwb*9_65) where *Algorithm* 1 only increases the number of lines. In the other 27 cases, *Algorithm* 1 outperforms *BDD+GA* in terms of #lines and NCV-cost. Furthermore, *Algorithm* 1 reduces the number of lines and the NCV-cost by up to 66.90% and 78.41% (function *apex*5_104), respectively. The average reductions achieved by *Algorithm* 1 in the number of lines and NCV-cost are 22.39% and 40.33%, respectively.

With regard to the runtime of the algorithms, Stojković et al. [23] did not report the runtime of their algorithms. It can be observed from Table 5 that, the time efficiency of *Algorithm* 1 is better than *BDD+GA*.

As can be seen from the above analyses, while using NCV library to realize reversible circuits, using KFDDs to generate reversible circuits is also better than using FDDs or BDDs.

6 Conclusion

While synthesizing reversible circuits using the KFDD, although how to reduce the quantum cost and the number of qubits has been extensively researched, the restricted interactions between qubits are rarely considered. In this work, focusing on the NN-constraints, an attempt to combine reversible logic synthesis, gate decomposition, and qubit mapping in one synthesis flow is conducted. Based on the Clifford+T gate library, by defining NNC metrics for the NCT and MPP gates for the reversible logic level, we address the reduction of the NNC of reversible circuits synthesized from the KFDD. The ordering of nodes to be mapped influences the quality of reversible circuits synthesized from the KFDD. Thus, for reducing the NNC of the resulting reversible circuits, the ordering of nodes to be mapped is ranked by applying strategies guided by NNC metrics. For further improving the quality of the resulting circuits, local transformations are applied on node functions while mapping a node to a cascade of reversible gates.

In the Clifford+T quantum mappings of NCT and MPP gates, two SWAP-gates are applied in order to decrease the distance between the control and the target line of each CNOT gate by one. One SWAP-gate is used for moving the control and the target line together, another is used to restore the original order of lines [11, 19]. As a result, the NNC metrics defined for the NCT and MPP gates and the NNC evaluation of the reversible

circuits are pessimistic. On the other hand, it is usually considered that the NN-constraints imposed by lattice models are less restricted than the coupling constraints imposed by IBM quantum architectures. Combining reversible logic synthesis, gate decomposition, and qubit mapping in a more general synthesis flow for handling the coupling constraints at the reversible logic level by defining more exact NNC metrics and designing more exact strategies is future work.

Funding Information This work was supported by the National Natural Science Foundation of China (No.61961023), the Jiangxi Provincial Natural Science Foundation (No.20202BABL202007), the Guangxi Natural Science Foundation (No.2021GXNSFAA220046), and the Doctoral Foundation of Guangxi University of Science and Technology (No.21Z04).

Data Availability Statement All data generated or analyzed during this study are within the paper.

Conflicts of Interests The authors declare that they have no conflicts of interest to this work.

Competing Interests The authors declare that they have no competing interests.

References

1. Abdalhaq BK, Awad A, Hawash A (2020) Reversible logic synthesis using binary decision diagrams with exploiting efficient reordering operators. IEEE Access 8:156001–156016
2. Abdessaied N, Drechsler R (2016) Reversible and Quantum Circuits: Optimization and Complexity Analysis. Springer International Publishing AG
3. Bu D, Wang P (2019) An improved KFDD based reversible circuit synthesis method. Integr VLSI J 69:251–265
4. Deb A, Dueck GW, Wille R (2021) Exploring the potential benefits of alternative quantum computing architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 40(9):1825–1835
5. Ding J, Yamashita S (2020) Exact synthesis of nearest neighbor compliant quantum circuits in 2-D architecture and its application to large-scale circuits. IEEE Trans Comput Aided Des Integr Circuits Syst 39(5):1045–1058
6. Drechsler R, Becker B (1998) Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions. IEEE Trans Comput Aided Des Integr Circuits Syst 17(10):965–973
7. Fazel K, Thornton MA, Rice JE (2007) ESOP-based toffoli gate cascade generation. In: Proc. 2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. pp 206–209
8. Gupta P, Agrawal A, Jha NK (2006) An algorithm for synthesis of reversible logic circuits. IEEE Trans Comput Aided Des Integr Circuits Syst 25(11):2317–2330
9. Kole A, Hillmich S, Datta K, Wille R, Sengupta I (2020) Improved mapping of quantum circuits to IBM QX architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 39(10):2375–2383
10. Kole A, Datta K, Sengupta I (2016) A heuristic for linear nearest neighbor realization of quantum circuits by swap gate insertion using N-gate lookahead. IEEE J Emerg Sel Top Circuits Syst 6(1):62–72
11. Kole A, Datta K, Sengupta I, Wille R (2015) Towards a cost metric for nearest neighbor constraints in reversible circuits. In: Proc. International Conference on Reversible Computation. pp 273–278
12. Li S, Zhou X, Feng Y (2021) Qubit mapping based on subgraph isomorphism and filtered depth-limited search. IEEE Trans Comput 70(11):1777–1788
13. Lin CC, Jha NK (2014) RMDDS: Reed-Muller decision diagram synthesis of reversible logic circuits. ACM J Emerg Technol Comput Syst 10(2): Article 14, 25 pages
14. Meter RV, Oskin M (2006) Architectural implications of quantum computing technologies. ACM J Emerging Technologies Comp Syst 2(1):31–63
15. Miller DM, Maslov D, Dueck GW (2003) A transformation based algorithm for reversible logic synthesis. In: Proc. the 40th Annual Design Automation Conference. pp 318–323
16. Nielsen MA, Chuang IL (2010) Quantum Computation and Quantum Information: 10th, Anniversary. Cambridge University Press, New York, USA
17. Niemann P, Bandyopadhyay C, Drechsler R (2021) Combining SWAPs and remote Toffoli gates in the mapping to IBM QX architectures. In: Proc. 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp 200–205
18. Niemann P, Gupta A, Drechsler R (2019) T-depth optimization for fault-tolerant quantum circuits. In: Proc. 2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL). pp 108–113
19. Saeedi M, Wille R, Drechsler R (2011) Synthesis of quantum circuits for linear nearest neighbor architectures. Quantum Inf Process 10(3):355–377
20. Sasao T (1999) Switching Theory for Logic Synthesis. Springer, Boston, MA
21. Soeken M, Roetteler M, Wiebe N, Micheli GD (2019) LUT-based hierarchical reversible logic synthesis. IEEE Trans Comput Aided Des Integr Circuits Syst 38(9):1675–1688
22. Soeken M, Wille R, Drechsler R (2010) Hierarchical synthesis of reversible circuits using positive and negative davio decomposition. In: Proc. 2010 5th International Design and Test Workshop (IDT). pp 143–148
23. Stojković S, Stanković R, Moraga C, Stanković M (2020) Reversible circuits synthesis from functional decision diagrams by using node dependency matrices. J Circuits Syst Comput 29(5): Article 2050079, 32 pages
24. Wille R, Drechsler R (2009) BDD-based synthesis of reversible logic for large functions. In: Proc. the 46th Annual Design Automation Conference. pp 270–275
25. Wille R, Große D, Teuber L, Dueck GW, Drechsler R (2008) RevLib: an online resource for reversible functions and reversible circuits. In: Proc. 38th International Symposium on Multiple Valued Logic. pp 220–225
26. Wille R, Lye A, Drechsler R (2014) Considering nearest neighbor constraints of quantum circuits at the reversible circuit level. Quantum Inf Process 13(2):185–199
27. Wille R, Lye A, Drechsler R (2014) Exact reordering of circuit lines for nearest neighbor quantum architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 33(12):1818–1831
28. Wille R, Drechsler R (2010) Towards a Design Flow for Reversible Logic. Springer, Dordrecht
29. Zulehner A, Wille R (2018) One-pass design of reversible circuits: combining embedding and synthesis for reversible logic. IEEE Trans Comput Aided Des Integr Circuits Syst 37(5):996–1008
30. Soeken M, Frehse S, Wille R, Drechsler R (2011) RevKit: an open source toolkit for the design of reversible circuits. In: Proc. International Conference on Reversible Computation. pp 64–76

Dengli Bu is currently a full professor in the School of Electrical, Electronic and Computer Science at Guangxi University of Science and Technology. He received the bachelor's and the master's degrees from the Department of Electronic Science and Technology of Nankai University, Tianjin, China, in 1996 and 1999, respectively. He received the Ph.D. degree from the Department of Computer Science and Technology of Tongji University, Shanghai, China, in 2015. From 2002 to 2020, he served as an associate professor at the School of Electronics and Information Engineering of Jinggangshan University. He joined School of Electrical, Electronic and Computer Science of Guangxi University of Science and Technology in 2021. His research interests include synthesis of reversible circuits and quantum circuits.

Junjie Yan received the Ph.D degree from Chongqing University of Post and Telecommunication in 2021. He joind School of Electrical, Electronic and Computer Science of Guangxi University of Science and Technology in 2021.

Pengjie Tang received the Ph.D. degree in the Department of computer science and technology of Tongji University, Shanghai, China, in 2019. He joined School of Electronics and Information Engineering of Jinggangshan University in 2012.

Haohao Yuan received the master's degree from North University of China in 2007. She is currently an associate professor in the School of Electrical, Electronic and Computer Science of Guangxi University of Science and Technology. Her research interests include circuit design and internet-of-things (IoT).