

Exact Global Reordering for Nearest Neighbor Quantum Circuits Using A*

Alwin Zulehner, Stefan Gasser, and Robert Wille

Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
{alwin.zulehner, stefan.gasser, robert.wille}@jku.at

Abstract. Since for certain realizations of quantum circuits only adjacent qubits may interact, qubits have to be frequently swapped – leading to a significant overhead. Therefore, optimizations such as exact global reordering have been proposed, where qubits are reordered such that the overall number of swaps is minimal. However, to guarantee minimality all $n!$ possible permutations of qubits have to be considered in the worst case – which becomes intractable for larger circuits. In this work, we tackle the complexity of exact global reordering using an A* search algorithm. The sophisticated heuristics for the search algorithm proposed in this paper allow for solving the problem in a much more scalable fashion. In fact, experimental evaluations show that the proposed approach is capable of determining the best order of the qubits for circuits with up to 25 qubits, whereas the recent state-of-the-art already reaches its limits with circuits composed of 10 qubits.

1 Introduction

Quantum computations employ an emerging technology where operations are performed on quantum bits (qubits) rather than conventional bits that can only represent two basis states. Exploiting quantum physical effects of qubits like superposition and entanglement allow to reduce the computational complexity of certain tasks significantly compared to conventional logic (cf. [1]). Well known examples are Shor’s algorithm (cf. [2]) for integer factorization or Grover’s algorithm for database search (cf. [3]). Such quantum computations are usually described using so-called quantum circuits, where qubits are represented as circuit lines. Operations on a subset of these qubits are described by quantum gates.

However, for many physical realizations, quantum circuits have to employ constraints on the interaction distance of qubits. More precisely, quantum gates can only be applied to adjacent qubits. To fulfill this requirement, SWAP operations (gates) that swap the values of two adjacent qubits are added to the quantum circuit – leading to a significant overhead. This overhead can be reduced by permuting the order of the qubits (circuit lines).

A broad variety of different approaches has been presented for this purpose – including solutions relying on templates [4], local and global reordering strategies [4], dedicated data-structures [5–8], or look-ahead schemes [9]. Also exact approaches, i.e. solutions guaranteeing the minimal number of SWAP gate

insertions, have been proposed [10, 11]. The work published in [11] provides a comprehensive overview of the state-of-the-art. All these approaches particularly focus on how to properly reorder the qubits in the circuit so that the respective interaction distance (and, hence, the number of required SWAP gates) is reduced.

In this work, we focus on global reordering. Here, heuristic as well as exact solutions have been proposed. Exact solutions are of particular interest as they guarantee the minimal number of SWAP insertions. Guaranteeing minimality, however, significantly increases the complexity of the considered problem. In the worst case, all $n!$ possible permutations of qubits have to be considered – an exponential complexity. We tackle this exponential complexity by using the A* search algorithm, i.e. a state-space search algorithm that traverses – guided by dedicated heuristics – only parts of the exponential search space until an optimal solution is determined. Experimental evaluations show that the proposed approach is able to determine the optimal order of the qubits (circuit lines) for quantum circuits composed of up to $n = 25$ qubits, whereas state-of-the-art solutions for exact global reordering are currently limited to $n = 10$ qubits.

This paper is structured as follows. In Section 2, we review nearest neighbor compliant quantum circuits. Based on that, we discuss the effect of globally permuting the order of the circuit lines in Section 3. In Section 4, we propose two approaches to determine the optimal order of the qubits using the A* search algorithm and discuss their differences. Finally, the proposed approaches are experimentally evaluated in Section 5 while Section 6 concludes the paper.









2 Nearest Neighbor Compliant Quantum Circuits

In contrast to conventional computation, *quantum computation* [1] operates on qubits instead of bits. A *qubit* is a two-state quantum system, with basis states $|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ (representing Boolean values 0 and 1, respectively). Furthermore, a qubit can be in a superposition of these basis states, i.e. $|x\rangle = \alpha|0\rangle + \beta|1\rangle$, where the complex amplitudes α and β satisfy $|\alpha|^2 + |\beta|^2 = 1$. Note that the state of a qubit cannot directly be observed, because measurement collapses the qubit into one of the two basis states $|0\rangle$ or $|1\rangle$. More precisely, the qubit collapses to basis state $|0\rangle$ with probability $|\alpha|^2$ and to basis state $|1\rangle$ with probability $|\beta|^2$.

This simply extends to quantum systems composed of n qubits. Such a system is in a superposition of its 2^n basis states. Operations on such systems are performed through multiplication of appropriate $2^n \times 2^n$ unitary matrices.

A usual representation for quantum computations are *quantum circuits*. Here, the respective qubits are denoted by solid *circuit lines*. Operations are represented by *quantum gates*. These operations may operate on a subset of the circuit lines only. Table 1 lists common 1-qubit quantum gates together with the corresponding unitary matrices describing their operation. In order to perform operations on more than one qubit, *controlled quantum gates* are applied. These gates are composed of a *target line* $|t\rangle$ and a control line $|c\rangle$ and realize the unitary operation represented by the matrix

Table 1. Quantum gates

Hadamard-Gate  $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	Pauli-Y-Gate  $\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-X-Gate  $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	Pauli-Z-Gate  $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
V-Gate  $\frac{1+i}{2} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$	S-Gate  $\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{2}} \end{pmatrix}$
W-Gate  $\frac{1}{2} \begin{pmatrix} 1 + \sqrt{i} & 1 - \sqrt{i} \\ 1 - \sqrt{i} & 1 + \sqrt{i} \end{pmatrix}$	T-Gate  $\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$

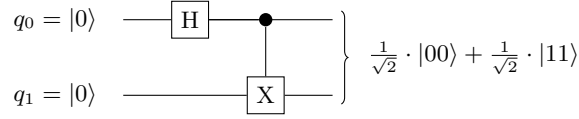


Fig. 1. Quantum circuit

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U & \\ 0 & 0 & & \end{pmatrix},$$

where U denotes the operation applied to the target line. In the remainder of this work, we use the following formal notation:

Definition 1. A quantum circuit is denoted by the cascade $G = g_1 g_2 \dots g_{|G|}$ of gates (in figures drawn from left to right), where $|G|$ denotes the total number of gates. The number of qubits and, thus, the number of circuit lines is denoted by n . The costs of a quantum circuit (also denoted as quantum cost) are defined by the number $|G|$ of gates.

Example 1. Fig. 1 shows a quantum circuit composed of $n = 2$ circuit lines and $|G| = 2$ gates. This circuit gets $|00\rangle$ as input and transforms the state of the underlying quantum system to $\frac{1}{\sqrt{2}} \cdot |00\rangle + \frac{1}{\sqrt{2}} \cdot |11\rangle$.

In the recent years, researchers proposed several physical realizations for quantum circuits. This led to a better understanding of their physical limitations

and constraints, e.g. with respect to the interaction distance, decoherence time, or scaling (see e.g. [12–14]). Besides that, so-called *nearest neighbor constraints* have to be satisfied for many quantum circuit architectures. This particularly holds for technologies based on proposals for ion traps [15–17], nitrogen-vacancy centers in diamonds [18, 19], quantum dots emitting linear cluster states linked by linear optics [20], laser manipulated quantum dots in a cavity [21], and superconducting qubits [22, 23]. Here, nearest neighbor constraints limit the interaction distance between gate qubits and require that computations are performed between adjacent, i.e. nearest neighbor, qubits only.

In order to formalize this restriction for electronic design automation, a corresponding metric representing the costs of a quantum circuit to become nearest neighbor compliant has been introduced in [4]. There, the authors defined the *Nearest Neighbor Cost* as follows:

Definition 2. *Assume a 2-qubit quantum gate $g(c, t)$ with a control at the line c and a target at line t , where c and t are numerical indices holding $0 \leq c, t < n$. Then, the Nearest Neighbor Cost (NNC) for g is calculated using the distance between the target and the control line. More precisely,*

$$NNC(g) = |c - t| - 1.$$

As a result, a single control gate g is termed nearest neighbor compliant if $NNC(g) = 0$. 1-qubit gates are assumed to have NNC of 0. The resulting NNC for a quantum circuit is defined by the sum of the NNC of its gates, i.e.

$$NNC(G) = \sum_{g \in G} NNC(g).$$

A quantum circuit G is termed nearest neighbor compliant if $NNC(G) = 0$, i.e. if all quantum gates are 1-qubit gates or adjacent 2-qubit gates.

Example 2. Consider the circuit G depicted in Fig. 2(a). Gates are denoted by $G = g_1 \dots g_7$ from the left to the right. As can be seen, gates g_2, g_4, g_5 , as well as g_6 are non-adjacent and have nearest neighbor costs of $NNC(g_2) = 2$, $NNC(g_4) = 1$, $NNC(g_5) = 1$, as well as $NNC(g_6) = 2$, respectively. Hence, the entire circuit has nearest neighbor costs of $NNC(G) = 6$.

A naive way to make an arbitrarily given quantum circuit nearest neighbor compliant is to modify it by additional SWAP gates.

Definition 3. *A SWAP gate is a quantum gate $g(q_i, q_j)$ including two qubits q_i, q_j and maps $(q_0, \dots, q_i, q_j, \dots, q_{n-1})$ to $(q_0, \dots, q_j, q_i, \dots, q_{n-1})$. That is, a SWAP gate realizes the exchange of two quantum values (in figures drawn using two connected \times symbols).*

These SWAP gates allow for making all control lines and target lines adjacent and, by this, help to satisfy the nearest neighbor constraint. More precisely, a cascade of adjacent SWAP gates can be inserted in front of each gate g with non-adjacent circuit lines in order to shift the control line of g towards the target line, or vice versa, until they are adjacent. Afterwards, SWAP gates are inserted to restore the original order of circuit lines.

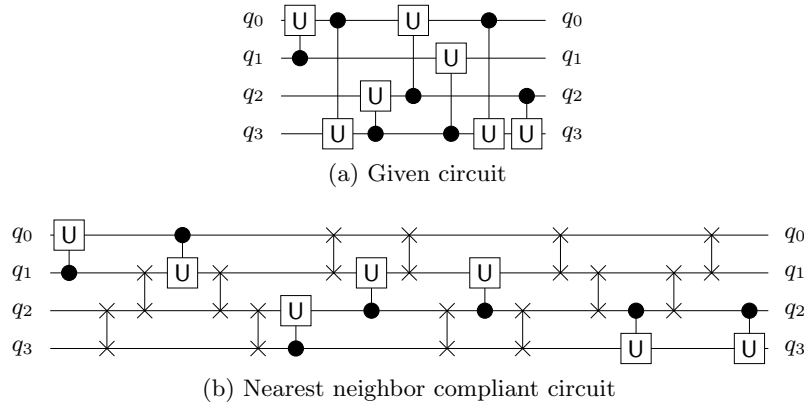


Fig. 2. Establishing nearest neighbor compliance

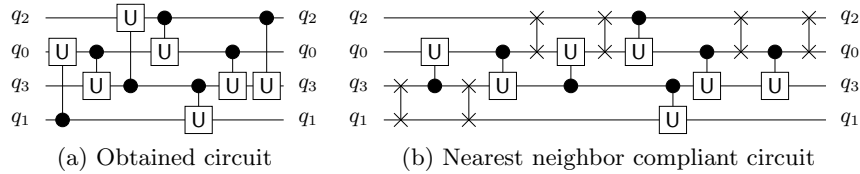


Fig. 3. Global Reordering (applied to the circuit from Fig. 2(a))

Example 3. Consider again the circuit depicted in Fig. 2(a). In order to make this circuit nearest neighbor compliant, SWAP gates in front and after all these gates are inserted as shown in Fig. 2(b).

3 Global Reordering for Nearest Neighbor Quantum Circuits

Global reordering became a suitable solution to reduce the cost of nearest neighbor compliant quantum circuits. Before adding SWAP gates to the circuit as reviewed in the previous section, the position of the qubits (circuit lines) is changed in order to reduce the number of required SWAP gates. An example illustrates the idea.

Example 4. Consider again the quantum circuit depicted in Fig. 2(a) and its nearest neighbor compliant version shown in Fig. 2(b). Permuting the order of qubits from $q_0q_1q_2q_3$ to $q_2q_0q_3q_1$ results in the circuit depicted in Fig. 3(a) – the nearest neighbor cost are reduced from 6 to 3. Hence, only 6 (instead of 12) SWAP gates are required to make the circuit nearest neighbor compliant (cf. Fig. 3(b)).

As demonstrated by the example above, the positions of the qubits have a significant impact on the nearest neighbor cost of the resulting circuit (and, thus,

on the number of required SWAP gates). To simplify the determination of the resulting nearest neighbor cost for a specific order of the qubits, the concept of an *adjacency matrix* of a quantum circuit can be used. The entries of this matrix indicate how often two qubits have to be adjacent, i.e. how many gates exist in the quantum circuit that operate exactly on these qubits. More formally:

Definition 4. Consider a quantum circuit composed of n qubits. Then, the *adjacency matrix* of this circuit is an $n \times n$ dimensional matrix M where the entries $m_{i,j}$, $0 \leq i, j < n$ provide the number of gates with the target and the controlling qubit at the i^{th} and j^{th} position, or vice versa (i.e. the number of gates with $g(i, j)$ or $g(j, i)$).

Since we do not distinguish between target and controlling qubit of a gate, the adjacency matrix is symmetric with respect to the main diagonal. Furthermore, the main diagonal of the matrix is skipped as well, because a single qubit cannot be the target and the controlling qubit of the same gate.

Example 5. Consider again the circuit depicted in Fig. 2(a). The according adjacency matrix is given in Table 2. To improve readability, we set all entries $m_{i,j}$ for which $j \leq i$ to *don't care* (denoted by $-$), because they contain redundant information. For example, the entry $m_{0,3}$ has value 2 because the circuit contains exactly two gates (the second and the sixth) for which qubits q_0 and q_3 have to be adjacent.

Table 2. Adjacency matrix for the circuit in Fig. 2(a)

	q_0	q_1	q_2	q_3
q_0	-	1	1	2
q_1	-	-	0	1
q_2	-	-	-	2
q_3	-	-	-	-

The adjacency matrix M of a quantum circuit can be used to determine the nearest neighbor cost (and, hence, the number of required SWAP gates) of a quantum circuit. More precisely:

$$NNC(M) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} m_{i,j} \cdot (j - i - 1)$$

Example 5 (continued). The nearest neighbor cost of the circuit shown in Fig. 2(a) is:

$$NNC(M) = m_{0,1} \cdot 0 + m_{0,2} \cdot 1 + m_{0,3} \cdot 2 + m_{1,2} \cdot 0 + m_{1,3} \cdot 1 + m_{2,3} \cdot 0 = 6.$$

Consequently, 12 SWAP gates are required to make the circuit nearest neighbor compliant.

Permuting the order of the qubits does not require to update the adjacency matrix. Instead, the resulting nearest neighbor cost when applying a permutation π to the order of the qubits is determined by

$$NNC(M, \pi) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} m_{i,j} \cdot (|\pi(j) - \pi(i)| - 1).$$

Example 5 (continued). If we apply the permutation $\pi = (1, 3, 0, 2)$ to the qubits of the circuit shown in Fig. 2(a), the new resulting order is $q_2q_0q_3q_1$ (cf. Fig. 3(a)). This changes the nearest neighbor cost to:

$$NNC(M, \pi) = m_{0,1} \cdot 1 + m_{0,2} \cdot 0 + m_{0,3} \cdot 0 + m_{1,2} \cdot 2 + m_{1,3} \cdot 0 + m_{2,3} \cdot 1 = 3.$$

Using the *adjacency matrix* of a quantum circuit allows for efficiently computing the nearest neighbor cost of the circuit with a permuted order of qubits. However, determining the best possible permutation, which requires the least number of SWAP gates is a computationally complex task. In the worst case, all $n!$ possible permutations have to be considered – an exponential complexity. Previous attempts tried to tackle this complexity by exploiting reasoning engines such as satisfiability solvers (see [10,11]). However, their applicability is still limited to rather small quantum circuits, i.e. circuits with not more than $n = 10$ qubits.

4 Global Reordering Using A*

In this section, we propose an alternative solution for global reordering in order to generate cost-efficient nearest neighbor compliant quantum circuits. To this end, we employ the power of the A* search algorithm. In the following, we review the basics of the A* algorithm and how global reordering can be translated into a search problem first. Based on that, we discuss two strategies for how to traverse the search space for the considered problem using A* search.

4.1 A* Algorithm

The A* algorithm is a state-space search algorithm. To this end, (sub-)solutions of the considered problem are represented by state nodes. Nodes that represent a solution are called *goal nodes* (multiple goal nodes may exist). The main idea is to determine the cheapest path (i.e. the path with the lowest cost) from the root node to a goal node. Since the search space is typically exponential, sophisticated mechanisms are employed in order to keep considering as few paths as possible.

All state-space search algorithms are similar in the way that they start with a root node (representing an initial partial solution) which is iteratively expanded towards the goal node (i.e. the desired complete solution). How to choose the node that shall be expanded next depends on the actual search algorithm. For A* search, we determine the cost of each leaf-node of the search state. Then, the node with the lowest cost is chosen to be expanded next. To this end, we

determine the cost $f(x) = g(x) + h(x)$ of a node x . The first part ($g(x)$) describes the cost of the current sub-solution (i.e. the cost of the path from the root to x). The second part describes the remaining cost (i.e. the cost from x to a goal node), which is estimated by a heuristic function $h(x)$. Since the node with the lowest cost is expanded, some parts of the search space (those that lead to expensive solutions) are never expanded.

Example 6. Consider the tree shown in Fig. 4. This tree represents the part of the search space that has already been explored for a certain search problem. The nodes that are candidates to be expanded in the next iteration of the A* algorithm are highlighted in blue. For all these nodes, we determine the cost $f(x) = g(x) + h(x)$. This sum is composed by the cost of the path from the root to the node x (i.e. the sum of the cost annotated at the respective edges) and the estimated cost of the path from node x to a goal node (provided in red). Consider the node labeled E . This node has cost $f(E) = (40 + 60) + 200 = 300$. The other candidates labeled B , C , and F have cost $f(B) = 580$, $f(C) = 360$, and $f(F) = 320$, respectively. Since the node labeled E has the fewest expected cost, it is expanded next.

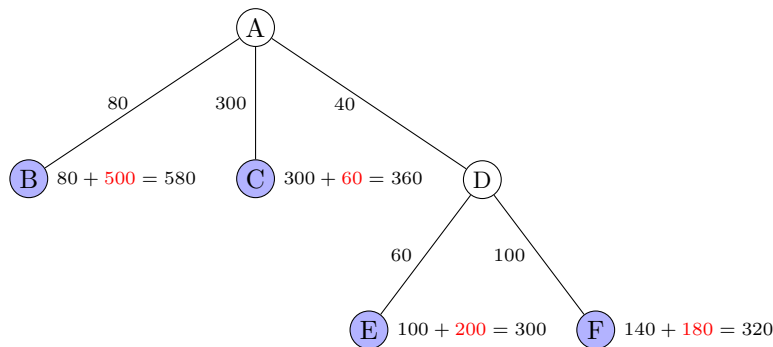


Fig. 4. A* search algorithm

Obviously, the heuristic cost should be as accurate as possible, to expand as few nodes as possible. If $h(x)$ always provides the correct minimal remaining cost, only the nodes along the cheapest path from the root node to a goal node would be expanded. But since the minimal costs are usually not known (otherwise, the search problem would be trivial to solve), estimations are employed. However, to ensure an optimal solution, $h(x)$ has to be *admissible*, i.e. $h(x)$ must not overestimate the cost of the cheapest path from x to a goal node. This ensures that no goal node is expanded (which terminates the search algorithm) until all nodes that have the potential to lead to a cheaper solution are expanded.

Example 6 (continued). Consider again the node labeled E . If $h(x)$ is admissible, the true cost of each path from this node to a goal node is greater than or equal to 200.

The general concept of the A* search algorithm as described above can easily be applied for exact global reordering of quantum circuits. In this case, the goal is to determine the permutation (the order) of the qubits, for which the fewest number of SWAPs gates are required in order to make the currently considered quantum circuit nearest neighbor compliant. Therefore, the nodes of the search space describe a (partial) permutation of the qubits. More precisely, a node with depth i (i.e. a node on with distance i to the root node) represents a partial permutation of i qubits. For simplicity, we label the nodes with the resulting order of the qubits instead of the partial permutation and neglect those qubits for which the permutation is not yet defined.

Example 7. Consider a quantum circuit composed of $n = 4$ qubits $q_0, q_1, q_2,$ and q_3 as well as a partial permutation $\pi = (0, \diamond, \diamond, 1)$. This partial permutation maps qubit q_0 to the first position and qubit q_3 to the second position. The mapping for the other qubits is not defined and, hence, denoted by \diamond (also called *hole*). The resulting order of the qubits is then $q_0q_3\diamond\diamond$. For simpler graphical visualization, we label the node that represents π with q_0q_3 – neglecting the qubits for which the position is not yet fixed.

A function $g(x)$ is needed to determine the cost of the path from the root to node x . Note that an edge in the tree describes a qubit that is added to the partial permutation. Consequently, the cost of the path from the root to node x can also be determined by the partial permutation that is represented by x . To this end, we determine the resulting nearest neighbor cost of the circuit. Since the permutation is only partially defined, we consider only those gates for which the position of the target and the controlling qubit is already fixed (i.e. these qubits have to occur in the partial permutation).

Example 8. Consider a node labeled $q_1q_3q_0$. The cost $g(x)$ of this node is determined by the nearest neighbor cost of all gates $g(c, t)$, for which $c, t \in \{0, 1, 3\}$. As discussed above, this cost can be determined from the adjacency matrix by $g(x) = m_{0,1} \cdot 1 + m_{0,3} \cdot 0 + m_{1,3} \cdot 0$.

Besides the representation of the (sub-) solutions and a cost function $g(x)$, we need two more things for exact global reordering for nearest neighbor quantum circuits using A*:

- An expansion strategy for the nodes, i.e. a strategy how another qubit shall be added to the partial permutation and
- an admissible heuristic function $h(x)$ to estimate the resulting cost from node x to a goal node that suits to the expansion strategy.

In the following sections, we propose two such expansion strategies and discuss their according heuristic function $h(x)$.

4.2 Straightforward Strategy

In this section, we discuss a straightforward expansion strategy for the nodes encountered during the A* algorithm and a corresponding admissible heuristic $h(x)$. To this end, we consider a quantum circuit composed of n qubits.

Consider a tree node with depth i . This node represents a partial permutation composed of i qubits. Hence, the position of i qubits is already fixed. To generate a permutation of $i + 1$ bits, we simply add one of the remaining qubits to the right of the already placed ones. Since $n - i$ such qubits exist, the expansion of the node yields $n - i$ successors. An example illustrates the idea.

Example 9. Consider a quantum circuit composed of $n = 4$ qubits q_0 , q_1 , q_2 , and q_3 , and assume that the node highlighted blue in Fig. 5 has to be expanded next. This node represents a partial permutation q_1 . Since there are three qubits that are not contained in the partial permutation (q_0 , q_2 , and q_3), three successor nodes are generated. These nodes represent the partial permutations $q_1 q_0$, $q_1 q_2$, and $q_1 q_3$, respectively. The resulting nodes are illustrated in Fig. 5.

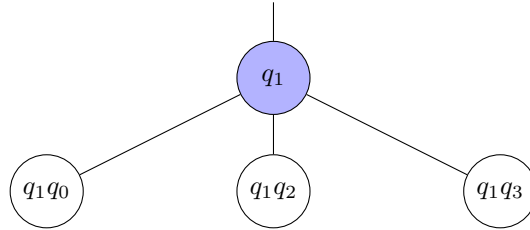


Fig. 5. Straightforward expansion strategy

Based on this expansion strategy, we have to estimate the cost $h(x)$ of the path from node x to a goal node. Recall that a goal node represents a permutation of the qubits. Consequently, we estimate how much the nearest neighbor cost increase when appending the remaining qubits to the current order. To ensure admissibility of this heuristic, we consider each qubit individually. Appending a qubit q_j to the right of the current order changes the nearest neighbor cost by Δ_{q_j} . This increase is determined by the nearest neighbor cost of all gates for which q_j is the controlling or the target qubit. Furthermore, the other qubit involved in the gate has to be part of the current order. All these values Δ_{q_j} are then summed up to approximate the overall increase of the nearest neighbor cost $h(x)$. Obviously this leads to an under-approximation of the real cost, since not all remaining qubits can be appended at the same location and the nearest neighbor costs between the remaining qubits are not considered.

Example 9 (continued). Consider the node labeled $q_1 q_0$ in Fig. 5. Appending qubit q_2 to the right of the current order would increase the resulting cost by $\Delta_{q_2} = m_{1,2} \cdot 1 + m_{0,2} \cdot 0$. Analogously, appending qubit q_3 to the right would increase the resulting cost by $\Delta_{q_3} = m_{1,3} \cdot 1 + m_{0,3} \cdot 0$. Consequently, the overall cost increase is estimated by the sum $h(x) = \Delta_{q_2} + \Delta_{q_3} = m_{1,2} + m_{1,3}$.

4.3 Elaborated Strategy

While the solution introduced above employs a rather straightforward scheme, we additionally propose a more sophisticated approach for expansion and estimation – described in this section. Here, we allow qubits to be inserted not only to the right of the already placed ones, but at all possible positions within the partial permutation. To this end, we restrict that, within an expansion, only one qubit is considered (while in the straightforward scheme introduced above all remaining qubits are considered; albeit with a fixed position). More precisely, out of the remaining qubits we choose the one which occurs most often as target or controlling qubit (accelerating the search by focusing on qubits with many interactions within the circuit). An example illustrates the idea.

Example 10. Consider a quantum circuit composed of $n = 4$ qubits q_0 , q_1 , q_2 , and q_3 , and assume that the node highlighted blue in Fig. 6 has to be expanded next. This node represents a partial permutation $q_1 q_0$. Assume that qubit q_2 is considered next as this is the one of the remaining qubits which interacts most often in the considered circuit, i.e. occurs most often as target or controlling qubit. Since there are three possibilities where to insert qubit q_2 , three successor nodes are generated. These nodes represent the partial permutations $q_2 q_1 q_0$, $q_1 q_2 q_0$, and $q_1 q_0 q_2$, respectively. The resulting nodes are illustrated in Fig. 6.

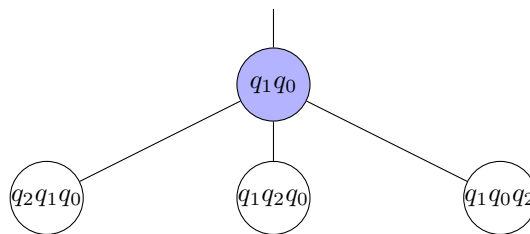


Fig. 6. Elaborated expansion strategy

Since we have a different expansion strategy, another heuristic to approximate the remaining cost is required. In contrast to above, the position at which the remaining qubits are inserted is not fixed anymore. Therefore, we have to determine $\Delta_{q_j}^k$ for each position k at which a qubit q_j can be inserted. Then, Δ_{q_j} is the minimum of all these values (since the heuristic has to be admissible). Finally, to estimate the overall cost increase when all remaining qubits q_j are inserted, we sum up all these values Δ_{q_j} to obtain $h(x)$.

Example 10 (continued). Consider again the node highlighted in blue in Fig. 6. The heuristic cost $h(x)$ of this node is determined as follows. For all remaining qubits (i.e. q_2 and q_3) we estimate the cost increase when adding the respective qubit to the permutation. Each remaining qubit can be inserted at three positions. Inserting the qubit q_2 at position zero (at the left of q_1) increases the

nearest neighbor cost by $\Delta_{q_2}^0 = m_{0,2} \cdot 1$. Analogously, inserting the qubit at positions one and two yields $\Delta_{q_2}^1 = m_{0,1} \cdot 1$ and $\Delta_{q_2}^2 = m_{1,2} \cdot 1$, respectively. Then, the minimum $\Delta_{q_2} = \min(\Delta_{q_2}^0, \Delta_{q_2}^1, \Delta_{q_2}^2) = \min(m_{0,2}, m_{0,1}, m_{1,2})$ of the three possibilities is determined. Analogously, $\Delta_{q_3} = \min(m_{0,3}, m_{0,1}, m_{1,3})$ is determined. Finally, the sum of the minima is determined, i.e. $h(x) = \Delta_{q_2} + \Delta_{q_3}$.

4.4 Discussion

In this section, we compare the two expansion strategies proposed above. To this end, we analyze how many successor nodes are generated when expanding a node with depth i . For the straightforward strategy, such a node has $n - i$ successors, because each of the $n - i$ remaining literals can be appended to the right of the current order. In contrast, a node with depth i generates $i + 1$ successors when expanded using the elaborated strategy, because the qubit that is inserted is fixed and there are $i + 1$ possibilities where this qubit might be inserted.

Consider the case that the estimated cost of a node with depth i is larger than the minimum that can be achieved. This means that this node (and, therefore, also its child nodes) will never be expanded. In case we use the straightforward expansion strategy, we therefore prune $(n - i) \cdot (n - i - 1) \cdot \dots \cdot 1 = (n - i)!$ possible solutions of the search tree. In contrast, if we apply the elaborated expansion strategy, we prune $(i + 1) \cdot (i + 2) \cdot \dots \cdot n = n!/i!$ solutions. Consequently, eliminating a node with depth i prunes significantly more possible solutions if the elaborated expansion strategy is used. However, the heuristics to estimate the resulting cost is computationally more expensive for the elaborated expansion strategy. Since we have to determine the best position for each of the remaining qubits, $O(n^3)$ lookups in the adjacency matrix are required. In contrast, using the straightforward expansion strategy requires $O(n^2)$ such lookups.

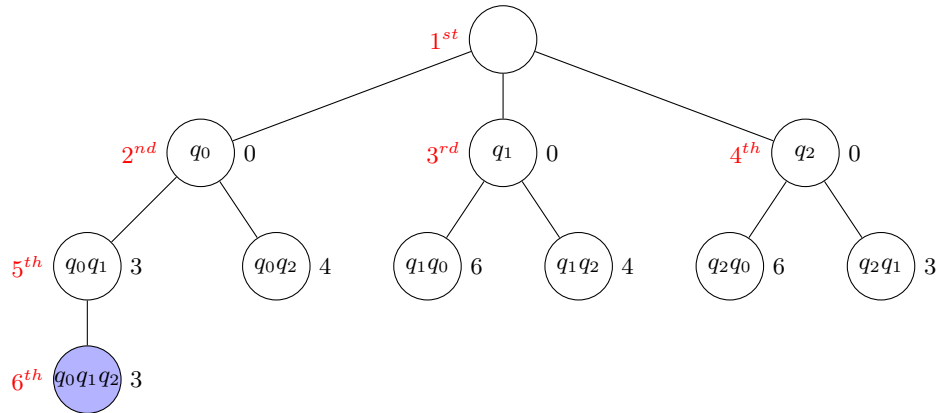


Fig. 7. Search tree of the straightforward expansion strategy

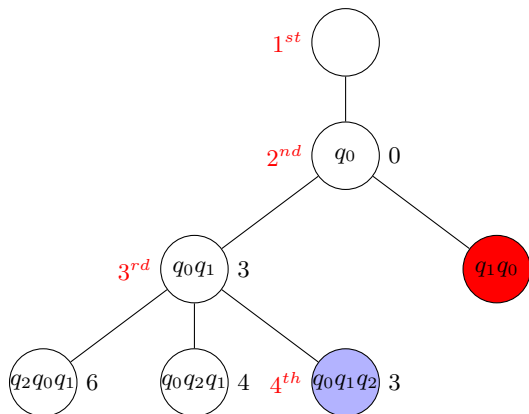


Fig. 8. Search tree of the elaborated expansion strategy

Example 11. Consider a quantum circuit with three circuit lines (denoted by q_0 , q_1 , and q_2), where the corresponding adjacency matrix has the entries $m_{0,1} = 4$, $m_{0,2} = 3$, and $m_{1,2} = 6$. Fig. 7 and Fig. 8 show the resulting trees generated by the A* search algorithm using the naive expansion strategy and the elaborated expansion strategy, respectively. The black numbers attached to the nodes represent the corresponding cost $f(x)$. The red numbers indicate the iteration in which the nodes were expanded. Furthermore, the expanded goal node (i.e. the one that yields an optimal solution) is highlighted in blue. The node highlighted red in Fig. 8 can immediately be rejected by the search algorithm, because it is symmetric to the other node on this level. For the straightforward strategy, a total of six nodes had to be expanded until the optimal solution was determined. In contrast, using the elaborated strategy allows to determine the same solution by expanding only four nodes.

5 Experimental Evaluation

We experimentally evaluated the proposed approach and compared the obtained results to the current state-of-the-art. To this end, we implemented the A* algorithm as well as the proposed expansion strategies described in the previous section in Java. The quantum circuits used as benchmarks were composed from the ones available in RevLib [24] and those previously used in [11]. The experiments for the proposed approaches were conducted on a Java virtual machine with 6 GB of memory running on a 1.7 GHz Intel i5 processor. The runtimes for the current state-of-the-art were taken from the corresponding paper (cf. [11]). However, since these experiments were conducted on a similar processor, the runtimes are comparable.

Table 3 summarizes the obtained results. In the first three columns, we list the name of the benchmark, the number of qubits n , as well as the minimal number of SWAP gates required to make the quantum circuit nearest neighbor

compliant. The fourth column lists the runtime of the current state-of-the-art approach [11]. The remaining columns list the runtime t , the number of created nodes, and the number of expanded nodes for the straightforward approach (proposed in Section 4.2) as well as for the elaborated approach (proposed in Section 4.3).

Table 3. Experimental Evaluation

Benchmark	n	SWAPs	s-o-t-a [11]	Straightforward (Sect. 4.2)		Elaborated (Sect. 4.3)			
			t	t	created	expanded	t	created	expanded
decod24-v3_46	4	4	0.10	0.00	23	9	0.00	10	4
hwb4_52	4	18	0.10	0.00	33	14	0.00	10	4
rd32-v0_67	4	4	1.10	0.00	23	9	0.00	10	4
4gt11_84	5	2	0.10	0.00	43	13	0.00	15	5
4gt13-v1_93	5	8	0.10	0.00	54	17	0.00	15	5
4mod5-v1_23	5	30	0.10	0.00	118	43	0.00	15	5
aj-e11_165	5	52	0.10	0.00	89	31	0.00	15	5
hwb5_55	5	120	0.10	0.00	112	40	0.00	19	6
QFT5	5	20	0.10	0.00	206	87	0.00	23	7
hwb6_58	6	290	0.10	0.01	743	271	0.00	79	18
mod8-10_177	6	156	0.10	0.01	419	128	0.00	44	11
ham7_104	7	140	1.90	0.01	1 407	391	0.02	151	30
rd53_135	7	136	1.80	0.01	1 539	412	0.00	41	10
QFT8	8	112	20.00	0.32	69 281	28 962	0.05	2 966	439
urf2_152	8	71 280	22.00	0.04	19 170	5 604	0.00	845	136
QFT9	9	168	236.50	3.30	623 530	260 651	0.30	23 127	2 959
urf1_149	9	179 832	241.30	0.21	84 588	21 384	0.05	5 896	786
urf5_158	9	176 284	247.00	0.44	137 694	38 582	0.03	3 648	522
QFT10	10	240	2936.80	45.73	6 235 301	2 606 502	1.13	204 568	23 119
rd73_140	10	150	1579.40	0.08	49 171	9 064	0.02	3 834	535
Shor3	10	4 802	1846.20	0.19	103 385	21 548	0.00	1 496	215
sym9_148	10	10 984	2415.12	0.34	138 043	28 921	0.00	474	77
sys6-v0_144	10	114	1586.40	0.07	40 315	7 340	0.02	1 990	290
urf3_155	10	453 368	3023.60	1.55	445 123	102 912	0.03	7 526	975
cycle10_2_110	12	4 104	TO	25.34	5 649 298	1 045 869	0.30	34 018	4 011
Shor4	12	13 588	TO	7.84	3 095 113	618 714	0.13	12 496	1 493
plus63mod4096_163	13	113 104	TO	481.68	39 226 031	6 834 319	0.44	36 283	4 246
0410184_169	14	48	TO	0.02	15 935	1 598	0.00	613	88
plus127mod8192_162	14	279 520	TO	TO	–	–	2.03	162 532	17 469
plus63mod8192_164	14	149 708	TO	TO	–	–	1.75	138 771	15 455
Shor5	14	34 680	TO	TO	–	–	1.77	172 154	16 703
ham15_108	15	1 340	TO	TO	–	–	1.11	78 458	7 910
rd84_142	15	284	TO	552.84	47 396 532	6 167 835	1.09	75 770	8 577
urf6_160	15	241 208	TO	TO	–	–	92.89	5 593 552	517 840
cnt3-5_180	16	340	TO	937.23	156 966 895	18 710 815	8.23	496 214	50 119
Shor6	16	76 318	TO	TO	–	–	47.39	3 310 774	263 822
add8_172	25	90	TO	499.31	48 043 975	3 007 062	60.13	990 050	70 269

A comparison to the current state-of-the-art shows that the proposed approaches allow for determining the optimal order of the qubits in a runtime which is magnitudes faster than the current state-of-the-art. For example, one of the largest benchmarks (*urf3_155*) requires more than 3000 CPU seconds using the state-of-the-art, while the approaches proposed here can solve this instance

in few seconds (straightforward approach) or even a fraction of a second (elaborated approach) only. Moreover, also the scalability is significantly better: While, thus far, minimal results for global reordering were available for quantum circuits composed of at most $n = 10$ qubits, the solutions proposed in this work are capable of generating results for circuits with up to 25 qubits.

Besides that, the results also confirm the discussion from Section 4.4 on the differences between the two A* schemes. The straightforward approach runs into a time out of half an hour for some circuits with 14, 15, or 16 qubits. In contrast, the more elaborated expansion strategy can also determine a solution for these benchmarks in less than 100 seconds. A further analysis explains this: Using the elaborated strategy, fewer nodes are generated and also significantly fewer of them are further expanded. This is because eliminating a node close to the root node of the tree prunes a larger part of the search space. Even though this requires a computationally more complex heuristic function to estimate the cost of a node, it eventually pays off and yields significant speedups compared to the straightforward strategy.

6 Conclusions

In this work we have considered the problem of exact global reordering to minimize the number of SWAP gates required to make a quantum circuit nearest neighbor compliant. Using the A* algorithm to determine the optimal permutation of the order of the qubits allows for significant improvements compared to the state-of-the-art. While current approaches are able to determine a solution for circuits with up to 10 qubits, the approach proposed in this paper is able to determine an exact solution for circuits composed of up to 25 qubits.

Acknowledgements

This work has partially been supported by the European Union through the COST Action IC1405.

References

1. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
2. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing* **26**(5) (1997) 1484–1509
3. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: *Symposium on the Theory of Computing*. (1996) 212–219
4. Saeedi, M., Wille, R., Drechsler, R.: Synthesis of quantum circuits for linear nearest neighbor architectures. *Quant. Info. Proc.* **10**(3) (2011) 355–377
5. Khan, M.H.: Cost reduction in nearest neighbour based synthesis of quantum Boolean circuits. *Engineering Letters* **16**(1) (2008) 1–5

6. Hirata, Y., Nakanishi, M., Yamashita, S., Nakashima, Y.: An efficient method to convert arbitrary quantum circuits to ones on a linear nearest neighbor architecture. In: Conf. on Quantum, Nano and Micro Technologies. (2009) 26–33
7. Shafaei, A., Saeedi, M., Pedram, M.: Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In: Design Automation Conf. (2013) 41–46
8. Wille, R., Quetschlich, N., Inoue, Y., Yasuda, N., Minato, S.: Using π dds for nearest neighbor optimization of quantum circuits. In: Reversible Computation - 8th Int'l Conf. (2016) 181–196
9. Wille, R., Keszocze, O., Walter, M., Rohrs, P., Chattopadhyay, A., Drechsler, R.: Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits. In: ASP Design Automation Conf. (2016) 292–297
10. Wille, R., Lye, A., Drechsler, R.: Optimal SWAP gate insertion for nearest neighbor quantum circuits. In: ASP Design Automation Conf. (2014) 489–494
11. Wille, R., Lye, A., Drechsler, R.: Exact reordering of circuit lines for nearest neighbor quantum architectures. IEEE Trans. on CAD **33**(12) (2014) 1818–1831
12. Fowler, A.G., Devitt, S.J., Hollenberg, L.C.L.: Implementation of Shor's algorithm on a linear nearest neighbour qubit array. Quant. Info. and Comput. **4** (2004) 237–245
13. Meter, R.V., Oskin, M.: Architectural implications of quantum computing technologies. J. Emerg. Technol. Comput. Syst. **2**(1) (2006) 31–63
14. Ross, M., Oskin, M.: Quantum computing. Comm. of the ACM **51**(7) (2008) 12–13
15. Amini, J.M., Uys, H., Wesenberg, J.H., Seidelin, S., Britton, J., Bollinger, J.J., Leibfried, D., Ospelkaus, C., VanDevender, A.P., Wineland, D.J.: Toward scalable ion traps for quantum information processing. New Journal of Physics **12**(3) (2010) 033031
16. Kumph, M., Brownnutt, M., Blatt, R.: Two-dimensional arrays of radio-frequency ion traps with addressable interactions. New Journal of Physics **13**(7) (2011) 073043
17. Nickerson, N.H., Li, Y., Benjamin, S.C.: Topological quantum computing with a very noisy network and local error rates approaching one percent. Nat Commun **4** (2013) 1756
18. Devitt, S.J., Fowler, A.G., Stephens, A.M., Greentree, A.D., Hollenberg, L.C.L., Munro, W.J., Nemoto, K.: Architectural design for a topological cluster state quantum computer. New Journal of Physics **11**(8) (2009) 083032
19. Yao, N.Y., Gong, Z.X., Laumann, C.R., Bennett, S.D., Duan, L.M., Lukin, M.D., Jiang, L., Gorshkov, A.V.: Quantum logic between remote quantum registers. Phys. Rev. A **87** (2013) 022306
20. Herrera-Martí, D.A., Fowler, A.G., Jennings, D., Rudolph, T.: Photonic implementation for the topological cluster-state quantum computer. Phys. Rev. A **82** (2010) 032332
21. Jones, N.C., Van Meter, R., Fowler, A.G., McMahon, P.L., Kim, J., Ladd, T.D., Yamamoto, Y.: Layered architecture for quantum computing. Phys. Rev. X **2** (2012) 031007
22. Ohliger, M., Eisert, J.: Efficient measurement-based quantum computing with continuous-variable systems. Phys. Rev. A **85** (2012) 062318
23. DiVincenzo, D.P., Solgun, F.: Multi-qubit parity measurement in circuit quantum electrodynamics. New Journal of Physics **15**(7) (2013) 075001
24. Wille, R., Große, D., Teuber, L., Dueck, G.W., Drechsler, R.: RevLib: an online resource for reversible functions and reversible circuits. In: Int'l Symp. on Multi-Valued Logic. (2008) 220–225 RevLib is available at <http://www.revlib.org>.