Improving Time-Indexed IP Models for Swap Minimization in Quantum Circuits

Paul Xie

August 28, 2021

Abstract

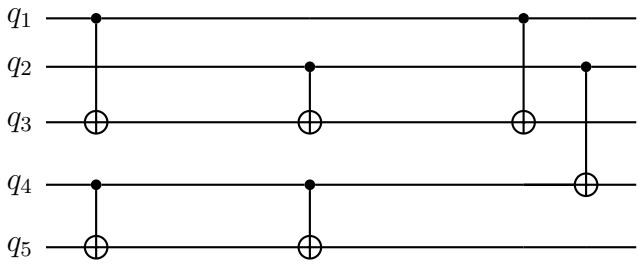
Due to physical limitations in quantum technologies, quantum circuits must adhere to the constraints posed by limited interactions between qubits. Hence, the Nearest Neighbor Compliance (NNC) problem has been proposed in recent years, which requires interacting qubits to neighbour each other on physical architectures. Though additional quantum gates, *swap* gates, can be inserted to existing quantum circuits to make them NNC, they bring a cost in terms of the reliability of the whole system. Therefore, we study the NNC problem where the number of swaps is to be minimized by solving Integer Linear Programming (ILP) models. In this report, we first provide a review of the approaches for solving swap minimization problems found in the literature and point out their respective limitations. We then propose alternative ILP approaches to help improve accuracy and we study the properties of the connectivity graphs to make sure that global optimal solutions can be found. Further, we run some benchmark instances and compare the results obtained by using different methods. Lastly, we identify the open problem that need to be solved in future research.

1 Introduction

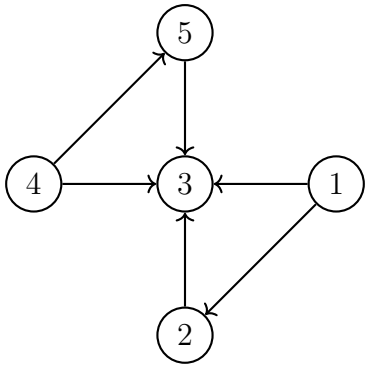
Quantum computers present many potential technological advantages [1] over classical computers due to their nature [2]. Prominent examples where quantum algorithms outperform classical algorithms include Shor's algorithm for factorization [3] and Grover's database search [4]. However, the realization of scalable quantum computers has been faced with many challenges, one of which is connectivity limitations. Because of this, nearest neighbour compliant (NNC) quantum circuits have been studied, which only allow two qubits to interact if they are positioned adjacent to each other in a quantum architecture.

An idealized quantum circuit is described by a sequence of gates that we need to implement (Figure 1a). A quantum architecture is described by the connectivity graph (Figure 1b). Connectivity graphs detail the physical layouts of nodes and edges. A connectivity graph can be denoted by $G = (V, A)$, where V is the set of all physical nodes and A is the set of all directed edges. A quantum circuit refers to numerous quantum gates which need to be implemented in a particular order (for instance Figure 1a). In general, a quantum circuit will include various gates, which might involve multiple qubits.

All complex quantum gates can be decomposed into sequences of CNOT gates and other 1-qubit gates [5]. The CNOT gates are 2-qubit gates that require certain qubit placement on the connectivity graph. We will only consider circuits that are composed of CNOT gates in this report since single qubit operations can be trivially performed on quantum chips.



(a) Quantum circuit



(b) Connectivity graph

Figure 1: Examples of (1a) idealized quantum circuit with 5 qubits and (1b) connectivity graph with 5 physical nodes.

Every CNOT gate has a control qubit and a target qubit. A CNOT gate will flip its target qubit's state if and only if the control qubit is in state $|1\rangle$. With this in mind, we can define the direction of a CNOT gate in the connectivity graph as from its control qubit to its target qubit. Since the edges between physical nodes have directions too, we say a gate is directly implemented if the direction of the gate aligns with the direction of the edge on which it is implemented. Gates can also be reversely implemented, which indicates the direction of the gate is opposite to the direction of the interacting edge. All gates have implementation costs, but gates that are reversely implemented will cost slightly more than those that are directly implemented.

In nearest neighbour compliant quantum circuits, two qubits are considered adjacent as long as there exists a directed edge between their respective physical locations on the connectivity graph. A complicated quantum circuit usually involves many single and 2-qubit gates, which make it impossible to place all qubits adjacent to one another simultaneously on 2D connectivity graphs. Thus, swap gates are used to interchange the position of two neighbouring qubits. Swap gates allow two qubits to be made adjacent when they need to interact with each other, and further apart when there is no interaction between them. However, these swap gates incur additional cost to the quantum circuit. So the number of swap gates used in a quantum circuit must be minimized.

The problem of swap gates minimization in NNC quantum circuits has been widely studied in recent years. Most researches focus on solving this problem with heuristic approaches [6][7], and some have tried using exact techniques to minimize the makespan [8][9] of each circuits. There has also been studies dedicated to 3D and multidimensional quantum architectures [10]. In this report, we focus on minimizing the overall gate implementation cost on 1D/2D architectures, and we look for the exact global optimal solution to each circuit using integer linear programming (ILP) formulations.

Our research is largely based on previously proposed time-index ILP formulation [11]. In the following sections, we will discuss how this model is incomplete and will fail to find the global optimal solution under some circumstances. We will then propose some additional methods that will elude such situations.

2 Problem Definition

In NNC quantum circuits, interacting qubits are required to neighbour each other. Given a quantum algorithm with only primitive gates, one should map the related quantum circuit into physical quantum architectures (e.g. quantum chips). The sequence in which all gates in the algorithm should be implemented is also provided in the quantum circuits. Multiple gates can be implemented simultaneously so long as there is no qubit involved in more than one operation. Like mentioned in Section 1, it is impossible to place all qubits adjacent to each other simultaneously, thus swap gates has to be implemented to make the circuit compliant. There are more than one ways to implement swap gates to make the original circuit compliant, some are more computationally costly than others. For instance, supposed we are given the architecture described by Figure 2, then the circuits described by Figure 3b and Figure 3c has identical effects on the original circuit (e.g. take same initial states and output same final states), but the former only used 2 swap gates while the latter implemented 5. Because of its smaller gate implementation cost, the solution described by Figure 3b is considered a better solution to the one shown in Figure 3c.

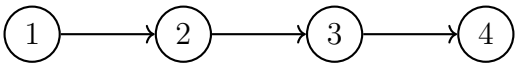


Figure 2: A linear connectivity graph with 4 nodes.

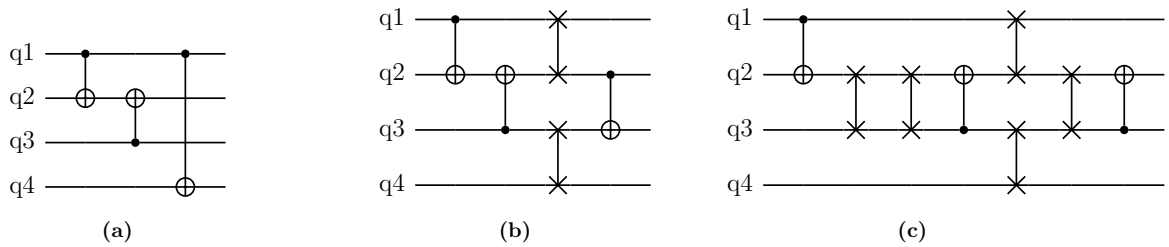


Figure 3: (3a): Idealized quantum circuit. (3b): Optimal solution to the original problem with only 2 swap gates. (3c): A more expensive solution with 5 swap gates.

As mentioned in Section 1, CNOT gates can be either directly or reversely implemented with slightly different cost. Therefore, the problem of minimizing the overall

computational cost arises. Given a quantum architecture and a quantum circuit, we need to determine the mapping of qubit positions onto physical architectures, as well as the sequence of swap gates required to modify the locations of each qubits.

At any given time instant, if there is at least one 2-qubit gate implemented, that time instant becomes the start of a new layer. Within each layer, there are additional time instant to allow the operation of swap gates. Therefore, the objective of this report can be defined as: **determine the optimal mapping of quantum circuits to physical architectures, and determine the optimal sequence of swap gates to make all the layers NNC.**

3 Related Works

In this section, we will review some useful findings and methods from the literature, which has helped our research. We will also point out how they are limited when solving larger scale quantum circuit compilation problems.

3.1 Time-Index ILP Formulation for MNS-NNC Problems

The time-indexed ILP model [11] is the foundation of our research. In this section, we will present the detailed mathematical formulation of the proposed ILP model.

Let $G = (V, A)$ denote a connectivity graph, where V and A represent the set of all physical nodes and directed edges on a quantum computer architecture respectively. Let Q denote the set of all qubits involved in the quantum circuit that we are trying to solve, and C denotes the set of all original 2-qubit quantum gates in the circuit. The set C can be partitioned into disjoint subsets $C_h \subseteq C$, where each of them associate to a layer h. C_h denote all 2-qubit quantum gates that need to be implemented at layer h, and that all gates in C_h should be implemented simultaneously at the first instance of layer h. The original gates can be implemented directly (i.e. in the same direction of the directed edges) or reversely (i.e. in the opposite direction of the directed edges), with respect to costs K_c^+ and K_c^- respectively. Let H denote the set of all layers, and for each layer $h \in H$, there is a maximum number of instances associated to the layer. Hence, the time horizon of a MNS-NNC problem can be denoted as $T = \{t_1^1, t_2^1, ..., t_m^1, t_1^2, ..., t_m^2, ...t_1^{|H|}, ..., t_m^{|H|}\}$, where instances between $[(h-1)m, hm]$ are related to layer h. In this report, we will indicate all instances of layer h as $T_h = \{t_1^h, ..., t_m^h\}$. At each time instance, we can implement in parallel either the gates composing the layer, or the required swap operations.

Given this setting,the problem determines for each time instant of a time horizon T, the optimal mapping of the quantum circuit onto the architecture G, such that all the gates of each layer can be implemented while minimizing the overall computational costs, which consists of direct/reverse implementation and swap cost. The variables used in this model is define as:

- $y_{ac}^t,\ c \in C,\ a \in A,\ t \in T$: binary variable, equal to 1 if gate c is directly implemented on arc (i, j) at time t, 0 otherwise.
- $y_{ac}^{*t},\ c \in C,\ a \in A,\ t \in T$: binary variable, equal to 1 if gate c is reversely implemented on arc (i, j) at time t, 0 otherwise.

- $x_{qi}^t,\ q \in Q,\ i \in V,\ t \in T$: binary variable, equal to 1 if qubit q is positioned at node i at time t, 0 otherwise.
- $z_{ij}^t,\ (i,j) \in A,\ t \in T$: binary variable, equal to 1 if a swap is implemented between a qubit positioned at node i and another qubit positioned at node j, or vice versa, at time t, 0 otherwise.

Thus, the objective function of the problem is:

$$min\ Z = \sum_{a\in A}\sum_{c\in C}\sum_{t\in T} K_c^+ y_{ac}^t + \sum_{a\in A}\sum_{c\in C}\sum_{t\in T} K_c^- y_{ac}^{*t} + \sum_{(i,j)\in A}\sum_{t\in T} S z_{ij}^t \tag{1}$$

Subject to the following constraints:

$$x_{pi}^{t_1^h} + x_{qj}^{t_1^h} \geq 2y_{ac}^{t_1^h}, \quad \forall c=(p,q) \in C_h,\ a=(i,j)\in A,\ h \in H \tag{2}$$

$$x_{pi}^{t_1^h} + x_{qj}^{t_1^h} \geq 2y_{ac}^{*t_1^h}, \quad \forall c=(p,q) \in C_h,\ a=(i,j)\in A,\ h \in H \tag{3}$$

$$\sum_{a\in A}(y_{ac}^{t_1^h} + y_{ac}^{*t_1^h}) = 1, \quad \forall c \in C_h,\ h \in H \tag{4}$$

$$\sum_{q\in Q} x_{qi}^t \leq 1, \quad \forall i \in V,\ t \in T \tag{5}$$

$$\sum_{i\in V} x_{qi}^t = 1, \quad \forall q \in Q,\ t \in T \tag{6}$$

$$x_{qj}^{t+1} + x_{qi}^t \leq 1 + z_{ij}^t, \quad \forall t \in \{1,2,...,|T|-1\},\ (i,j)\in A,\ q \in Q \tag{7}$$

$$x_{qi}^{t+1} + x_{qj}^t \leq 1 + z_{ij}^t, \quad \forall t \in \{1,2,...,|T|-1\},\ (i,j)\in A,\ q \in Q \tag{8}$$

$$x_{qi}^{t+1} + x_{qj}^t \leq 1, \quad \forall t \in \{1,...,|T|-1\},\ i,j \in V,\ (i,j)\wedge(j,i) \notin A,\ q \in Q \tag{9}$$

$$z_{ij}^{t^h} \leq \sum_{a\in\sigma(i)} z_a^{t^h-1} + \sum_{a'\in\sigma(j)} z_{a'}^{t^h-1}, \quad \forall t \in \{2,...,|T|\},\ t \neq t_1^h,\ h \in H,\ (i,j)\in A \tag{10}$$

Constraints 2 and 3 impose the condition that all involving qubits of gates in layer h shall be positioned at corresponding physical locations at the first instance of each layer. Constraints 4 make sure that gates in layer h should be implemented at the first instance of that layer, either directly or reversely. Constraints 5 and 6 impose that at every time instant, a qubit can only be positioned at one physical location, and every not may only contain no more than one qubit. Constraints 7 and 8 depict the physical scenario of swap operations, the qubit contained in node i can move to an adjacent node j if $z_{ij}^t = 1$. Constraints 9 make sure no qubit can move to any non-adjacent node between time instants. Finally, constraints 10 is the symmetry breaking constraints, where $\sigma(i)$ denotes all edges incident to node i.

3.1.1 Analysis and Limitations

As mentioned in previous sections, this formulation requires setting the appropriate values for m, the number of instants of a layer. If the value of m is too small, the solver may fail to find the optimal solution, or even a feasible solution (will be demonstrated in the next section). On the other hand, if the value of m is too large, constraints (5) - (10) will all be scaled up polynomially, so it takes significantly longer for the solver to find the

optimal solution. Therefore, it is vital to find the m value just "big enough". Finding the right m values for different graphs is not straight forward, which present the biggest challenge for this formulation.

3.2 Utilizing Kendall-Tau Distance to Model Swap Gates

The Kendall-Tau distance [12] is a metric of the number of pairwise disagreement between two ranged lists. Due to its nature, methods utilizing the Kendall-Tau distance to model swap gate operations on linear architectures [13] has been proposed. In this section, we present an ILP formulation with this alternative way of formulating swap gates.

In this formulation, it is unnecessary to calculate the number of layers needed between layers. We only need to look at the time instants at which one or more of the original 2-qubit gates are implemented. Let $t \in T$ denote all time instants with at least one 2-qubit interactions, let A, Q denote the set of all directed edges in the connectivity graph and the set of all qubits respectively. Let C represent the set of gates we need to implement in a given circuit, and the set of all gates we need to implement at time instant t is denoted by C_t.

Variables definition:

- $r_i^t \in \{1, 2, ..., n\}$: Location of qubit i at layer t;
- $z_{ij}^t \in \{0, 1\}$: $= 1$ if qubit i is positioned in front of qubit j in the physical layout ($r_i^t < r_j^t$) at time t;
- $x_{q,i}^t \in \{0, 1\}$: $= 1$ if qubit q is positioned at node i at time t;
- $y_{ac}^t \in \{0, 1\}$: $= 1$ if gate c is directly implemented over arc a at time t;
- $k_{ij}^t \in \{0, 1\}$: this variable is to make the IP linear. It put the constraint $k_{ij}^t \geq |z_{ij}^t - z_{ij}^{t+1}|$. This variable indicates whether there is a swap gate between two qubits i and j.

Objective:

$$min \sum_{\substack{i,j \in Q \\ i<j}} \sum_{t=1}^{|T|-1} S \cdot k_{ij}^t + \sum_{a \in A} \sum_{c \in C} \sum_{t=1}^{|T|} K_c^+ y_{ac}^t + \sum_{a \in A} \sum_{c \in C} \sum_{t=1}^{|T|} K_c^- y_{ac}^{*t} \tag{11}$$

Constraints:

$$x_{pi}^t + x_{qj}^t \geq 2y_{ac}^t, \quad \forall c = (p,q) \in C_t,\ a = (i,j) \in A,\ t \in T \tag{12}$$

$$\sum_{a \in A} y_{ac}^t + y_{ac}^{*t} = 1, \quad \forall c \in C_t,\ t \in T \tag{13}$$

$$r_i^t - r_j^t \leq M \cdot z_{ij}^t - 1, \quad \forall i,j \in Q,\ i \neq j,\ t \in T,\ M \text{ big enough} \tag{14}$$

$$r_q^t = \sum_{i \in V} i \cdot x_{qi}^t, \quad \forall q \in Q,\ t \in T \tag{15}$$

$$-k_{ij}^t \leq z_{ij}^t - z_{ij}^{t+1} \leq k_{ij}^t, \quad \forall i,j \in Q,\ i \neq j,\ t \in \{1,2,...,|T|-1\} \tag{16}$$

$$\sum_{q \in Q} x_{qi}^t \leq 1, \quad \forall i \in V,\ t \in T \tag{17}$$

$$\sum_{i \in V} x_{qi}^t = 1, \quad \forall q \in Q,\ t \in T \tag{18}$$

$$z_{ij}^t + z_{ji}^t = 1, \quad \forall i,j \in Q,\ i \neq j,\ t \in T \tag{19}$$

The set T in this formulation is essentially equivalent to the first instances of each layer in the time-index formulation [11]. On linear architectures, if qubit q_1 is positioned ahead of qubit q_2 at layer t (e.g. $z_{1,2}^t = 1$), and at layer $t+1$, q_1 is positioned behind q_2, then we know for sure that there is *exactly one* swap gate between these two qubits. Similarly, we can obtain the number of all swap gates between layer t and $t+1$ by

$$n_{swap}^t = \sum_{\substack{i,j \in Q \\ i<j}} |z_{ij}^t - z_{ij}^{t+1}| \tag{20}$$

Note that, equation (20) is exactly the definition of the Kendall-Tau distance between layers t and $t+1$. Multiplying n_{swap}^t with the cost of swap gates S gives the cost of swap implementations in layer t. However, having an absolute value in the objective function will make this formulation non-linear. Thus, we introduce an additional variable $k_{ij}^t \geq |z_{ij}^t - z_{ij}^{t+1}|$, $\forall i,j \in Q,\ i \neq j,\ t \in \{1,2,...,|T|-1\}$. Then we can simply replace the absolute value in the objective function with this k variable to ensure linearity of the formulation. Variables associated with original 2-qubit gates are defined exactly the same as the time-index formulation.

3.2.1 Analysis and Limitations

The advantage of this formulation is that it solves the problem really fast, while only able to solve problems on linear architectures is the most obvious limitation. For linear architectures, we can use this formulation to get the precise optimal solutions. For some complicated 2D architectures, we can make the graph linear by removing some edges [14], and the optimal solution found by solving this formulation is still a feasible solution in the original problem.

For non-linear architectures though, this method will no longer be applicable. For instance, if we have an architecture shown in 4, at layer 0, qubit i is placed at physical node i. The qubit ordering at this layer is $s_0 = \{q_1, q_2, q_3\}$. Suppose we simply swap qubits 1 and 3, then the qubit ordering at layer 1 becomes $s_1 = \{q_3, q_2, q_1\}$. Since qubit 1 and 3 were interconnected at layer 1, we know that there should only be 1 swap gate between layers 0 and 1. However, the calculation from equation (20) gives $n_{swap}^0 = 3$.

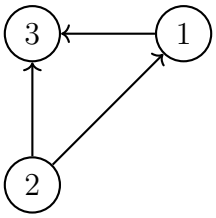


Figure 4: A triangular connectivity graph, on which the Kendall-Tau distance cannot be used to model swap gates.

3.3 Routing Numbers

Given the similarities to our objectives, the routing number of graphs [15] can be proven helpful to determine the values of m in quantum circuits. The routing number of a graph G, which is denoted by $rt(G)$, is defined as ***the smallest possible number of steps required to allow the graph to route from a starting configuration to any target configuration.*** If we define $rt(G, s_i, s_f)$ as the minimum possible number of steps required to achieve a specific target state s_f from an initial state s_i of graph G, then we have

$$\text{rt(G)}= \max_{s_i,s_f} \; rt(G, s_i, s_f) \tag{21}$$

The results of [15] can be summarized in Table 1.

Graph Type	**Upper Bound**	**Lower Bound**
Arbitrary fixed tree on n vertices	$3n-1$	Generic
Any tree on n vertices	$\lfloor \frac{3(n-1)}{2} \rfloor$	Generic
Complete graphs on $n \geq 3$ vertices	2	2
Complete bipartite graph $K_{n,n}$ with $n \geq 3$	4	4
General complete bipartite graphs $K_{m,n}$, with $m \leq n$	$2\lceil \frac{n}{m} \rceil + 2$	Generic
Cartesian products of graphs G and G', $rt(G) \leq rt(G')$	$2rt(G) + rt(G')$	Generic
$n - dim$ cubes	$2n-1$	Generic
$m \times n$ grid graphs, $m \leq n$	$2m+n$	Generic

Table 1: Upper and lower bounds of $rt(G)$ for different graph types.

Note: unless specified in Table 1, the lower bounds of $rt(G)$ is the intersection of the general lower bounds described by equations 22, 23 and 24. Also, there is a general upperbound described by equation 25.

$$\text{rt(G)} \geq diam(G), \text{where } Diam(G) \text{ is the diameter of } G. \tag{22}$$

$$\text{rt(G)} \geq \frac{2}{|C|} min(|A|, |B|), \text{where } A, B \subseteq V, \text{ separated by } C \subseteq V \text{ a cutset of vertices.} \tag{23}$$

$$\text{rt(G)} \geq \frac{2}{|C|} min(|A|, |B|) - 1, \text{where } A, B \subseteq V, \text{ separated by } C \subseteq E \text{ a cutset of edges.} \tag{24}$$

$$\text{rt(G)} \leq rt(H), \text{where } H \text{ is any spanning subgraph of } G. \tag{25}$$

3.3.1 Analysis and Limitations

The value of $rt(G)$ provides some useful insights to m^*, as it is a valid upperbound to m^*.

Claim: The routing number $rt(G)$ gives an upper bound on m^*.

Proof: Suppose we have graph $G = (V, E)$ in state s_1, where s_1 is an arbitrary state of graph G. Let S denote all possible states of graph G.

Assume that, we need to implement a certain combination of 2-qubit quantum gates $C_j \in C_G$ simultaneously in the next layer, and that $S_j^* = \{s_j : s_j \text{ makes } C_j \text{ optimally implemented}\}$ denote the set of all states that makes C_j optimal. Thus, we have the relation:

$$m^* = \max_{C_j \in C_G} \min_{s_i \in S_j^*} rt(G, s_1, s_i) \tag{26}$$

where C_G is the set of all possible combinations of 2-qubit quantum gates (e.g. $C = \{\{(q_1, q_2), (q_4, q_3)\}, \{(q_1, q_4), (q_2, q_3)\}, ...\}$, all possible gates combinations we can implement for a given graph). Since we know $S_j^* \subseteq S$, so we have equation 27:

$$\min_{s_i \in S} rt(G, s_1, s_i) \geq \min_{s_i \in S_j^*} rt(G, s_1, s_i), \quad \forall C_j \in C \tag{27}$$

Thus, we then have

$$\text{rt(G)} = \max_{s_i \in S} rt(G, s_1, s_i) \geq \min_{s_i \in S} rt(G, s_1, s_i) \geq \min_{s_i \in S_j^*} rt(G, s_1, s_i), \quad \forall C_j \in C \tag{28}$$

Since there is a "$\forall$" sign at the end of equation 28, the right-most part of equation 28 is equivalent to $\max_{C_j \in C_G} \min_{s_i \in S_j^*} rt(G, s_1, s_i)$, which means we have the following:

$$rt(G) \geq min(m) \tag{29}$$

It is interesting to explore when does equation 29 takes equal sign, but one thing we can say for sure is that, the upper bounds of $rt(G)$ will also be valid upper bounds for the minimum possible m value. Therefore, table 1 becomes handy when deciding the value of m in the time instance model.

However, the lowerbounds of $rt(G)$ is too tight for our problem in most cases and we will prove this by showing counterexamples in the next section. Intuitively, the value of $rt(G)$ covers all possible states, while m^* only covers situations in which corresponding gates can be optimally implemented.

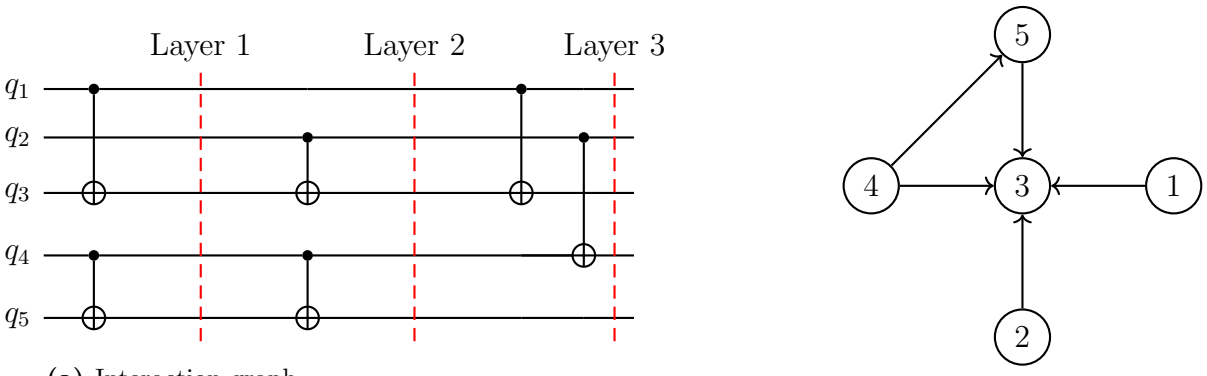
4 Formal Results

4.1 Counterexample of $m = Diam(G)$

The authors of [11] set $m = Diam(G)$, where $Diam(G)$ denotes the diameter of graph G. The diameter of a graph G is defined as the furthest distance (in arcs) between any two nodes in connectivity graph G. While this setting worked well for the benchmark instances they have shown in [11], it does not guarantee global optimal solution for some other architectures. For instance, we have connectivity graph $G = (V, A)$ shown in Figure 5b, and we need to implement the following simple circuit:

–Layer 1: implement gates (1,3) and (4,5) in parallel;
–Layer 2: implement gates (4,5) and (2,3) in parallel;
–Layer 3: implement gates (1,3) and (2,4) in parallel.

The first element in a parenthesis indicates the control qubit and the second element indicates the target qubit of each gate. The cost of reverse implementation is set to 12, while that of direct implementation is equal to 10. The cost of swap gate is set to 34. These costs are consistent with the values used in [11].

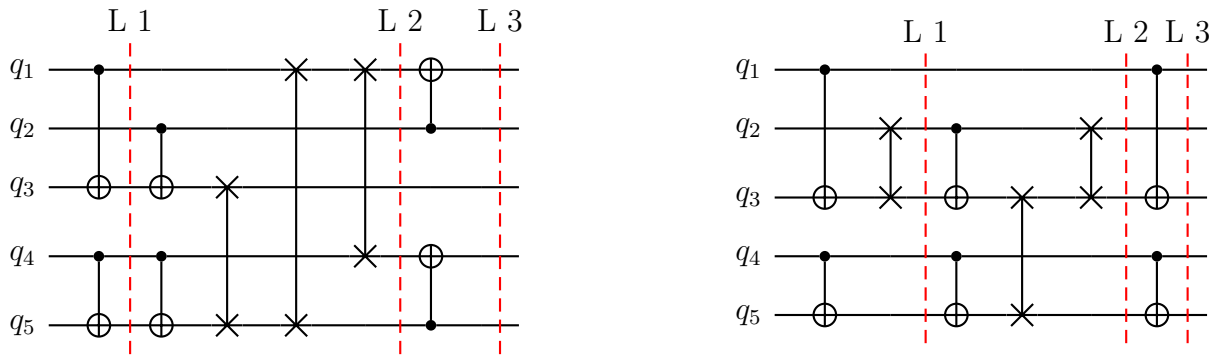


(a) Interaction graph.

(b) Connectivity graph.

Figure 5: In this architecture, setting $m = Diam(G) = 2$ does not guarantee global optimal solution to the given quantum circuit.

If we set $m = Diam(G) = 2$ in this architecture, the optimal solution $Z^*_{m=2} = 166$, with one swap gate implemented in layer 1 and two swap gates implemented in layer 2. But if we set m to any integer value larger than 2, we will have the global optimal solution $Z^*_{global} = 162$, with three swap gates implemented in layer 2. This is because when $m = 2$, the solver will not have enough steps to route the graph to the most desirable state. The global optimal solution can be visualized in Figure 6a, in which we can see that no two of the three swap gates implemented in layer 2 can be done in parallel, because $q3$ is involved in all three of the swaps. Figure 6b shows the optimal solution by setting $m = 2$. Thus, this solution requires $m \geq 3$ to be compliant.



(a) $m = 3$, cost = 162.

(b) $m = 2$, cost = 166.

Figure 6: Optimal solution to the problem described in Figure 5, with different m values.

4.2 Discussion on Feasibility and Optimality

Let's take a closer look at the objective function (equation (1)) of the NNC problem. We are minimizing the overall cost of swap, directly and reversely implemented gates, rather than just the cost of swap gates. Therefore, the solution with fewest number of swap gates is not necessarily the optimal solution, with Additional cost comes from the difference between direct/reverse implementation. For instance, if we have a complete graph $G = (V, A)$ whose vertices are all connected to each other, we do not need to implement any swap gate to make any gate sequence nearest neighbour compliant. But if, somehow, all the gates we want to implement are reversely connected, and we have a large number (> 50) of layers, we might want to take one extra step to swap all (or some) control/target qubits to make some gates directly implemented at the current and following layers, and it is possible that the resulting solution will cost less than the one without using any swap gate. Since one qubit can only be involved in one quantum gate at a certain instance, swapping all control/target qubits can be achieved within one step.

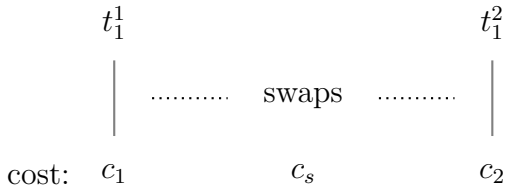
Claim: if $m = m_f^*$ guarantees a feasible solution for any given problem on graph G, then setting $m = m_f^* + 1$ will guarantee the optimal solution to be found by the solver.

Proof: Like discussed above, it is possible that a solution with more swap gates but fewer reverse implementations turns out to be optimal. So if there exist $m = m_f^*$ that guarantees a specific gate sequence feasible, we can simply swap all control/target qubits for the reversely implemented gates at the end of each layer, which only requires one additional step at each layer, to make all gates directly implemented. For instance, if at layer 1 the system is in state s_1 and s_2 is a state which makes the circuit compliant at layer 2. If we know it takes m_1 swaps to route the circuit from s_1 to s_2 (e.g. rt(G, s_1, s_2) $= m_1$), and that we want to know what m value will guarantee all gates to be directly implemented at layer 2, we can simply route the graph from s_1 to s_2 first, then swap all control/target qubits right before layer 2. Thus, $m = m_f^* + 1$ guarantees all gates can be implemented directly.

We can start the circuit by putting qubit i at node i. Let C_G denote all possible gate combinations that can possibly be implemented simultaneously on graph G. To determine the value of m_f^*, we need to find the minimum number of steps that can make every element in C_G feasible.

Note that: the starting configuration of the graph does not matter at all, since C_G covers all possible situations we might be facing in layer 2. Then at layer 2, we can simply

rename the qubit located at node i to q_1, and we will have the exact same problem. If we can find the m_f^* which guarantees all possible circuit feasible between layer 1 and layer 2, this m_f^* will also guarantee all possible circuits feasible between layer 2 and layer 3, and so on so forth.



Suppose we have a solution Z_1 to a given circuit between layer 1 and layer 2, in which all gates are directly implemented. The original gates implementation cost of this solution is denoted by c_1 and c_2, indicating the cost at layer 1 and layer 2 respectively, and the cost of swap gates is denoted by c_s. Now let's look at the global optimal solution, whose corresponding costs are denoted by c_1^*, c_2^* and c_s^* respectively. Since this is the global optimal solution, we have:

$$c_1^* + c_2^* + c_s^* \leq c_1 + c_2 + c_s \tag{30}$$

Also, we already know that the solution Z_1 has minimal implementation cost for the original 2-qubit gates due to the fact that all gates are directly implemented, we have:

$$c_1^* + c_2^* \geq c_1 + c_2 \tag{31}$$

Thus, we know that $c_s^* \leq c_2$, which indicates the global optimal solution implemented at most as many swap gates as Z_1. Therefore, if $m = m_f^* + 1$ guarantees all gates can be directly implemented between layer 1 and layer 2, it also guarantees the global optimal solution. □

With this relation, our problem becomes easier as we can simply look for the smallest possible m value which makes any gate sequence on a graph feasible, then we know what value to set in order for the solver to find the global optimal solution on a given graph.

For the same reason, if we can find the minimum number of steps m_1 such that all elements in C_G can not only be made feasible, but also all gates are directly implemented, then we know setting $m = m_1$ guarantees global optimal solution for any circuit acting on the given architecture.

The major challenge here is to find the m_f^* that guarantees feasibility to all elements in C_G on a given graph. If we are trying to solve a particular circuit on a given graph, and we find a feasible solution to this circuit, the m value acquired from this solution does not guarantee global optimal solution of the same problem, nor does it guarantee anything for other circuits applied on this graph.

4.3 Lower Bounds of $rt(G)$

We have discussed how the upperbounds of $rt(G)$ is a valid value to set for m, we will prove why the lower bounds of $rt(G)$ is too tight for m^*, which indicates setting $m = rt(G)$ will not always be the best option.

Claim: The lowerbounds of $rt(G)$ described by equations 22, 23 and 24 are too tight for m^*.

Proof: Suppose we have connectivity graph shown in Figure 2 and qubit q_i is located at node i. Since we have 4 qubits in total, we can implement a maximum of two 2-qubit gates simultaneously in the next layer. Let's first solve the combinatorial problem: if we have 4 nodes and we want to select 2 pairs of nodes without taking the direction into account, the number of distinct pair of selections is $C_4^2 C_2^2/2 = 3$. We can list all possible gate combinations:

1. implement qubits (1,4) and (2,3) simultaneously;
2. implement qubits (1,3) and (2,4) simultaneously;
3. implement qubits (1,2) and (3,4) simultaneously.

As has been discussed in previous sections, the directions in which the gates are implemented is not being considered at this step. We can just look for the feasible solutions where the involving qubits are neighbouring each other, and take that m value plus 1 will give us m^*. The first case can be achieved by swapping q_3 and q_4, then swapping q_2 and q_4; the second case can be done by swapping q_2 and q_3; the third case does not need any swap gate. Therefore, we know that $m = 3$ would allow the global optimal solution no matter what circuit we want to implement on architecture shown in Figure 2.

However, the lower bounds obtained by equations 22, 23 and 24 is 3, 2 and 4 respectively. Since the global lower bound of $rt(G)$ is the intersection of the three generic bounds, we obtain $LB_{global} = 4$ for this graph, which will exclude m^*. Therefore, setting $m = rt(G)$ will definitely guarantee global optimal solution to any problem, it will not be the best value we can find for m.

5 Proposed Approach

We first assign m values to outcomes from different approaches, and use those values to compute the optimal solution from the time-indexed model described in section 3.1. Then we compare the runtime for the solver to find optimal solutions in each setting. In addition, since the solution obtained from the Kendall-Tau based model described in section 3.2 is a feasible solution to the original problem, the number of swap gates used in this solution is smaller than that in the global optimal solution. We can then add some valid inequalities to the original time-indexed formulation.

In this report, we set the implementation cost of directly implemented gates to 10, while that of reversely implemented gates is set to either 11 or 12 [11]. The cost of swap gates is set to 34. These costs take into account the computational burden for the hardware to complete the circuit. First we run some benchmark instances from IBM Q platform [16] with 5 qubits, we then run some samples that are created by ourselves. As discussed in Section 3.2, the Kendall-Tau related model has the advantage of finding optimal solution faster than the time-indexed model, but is only applicable to linear architectures. As thus, we will make the connectivity graphs linear by removing a few edges [14] and compute the optimal solution using Kendall-Tau based model.

6 Experimental Results

In this section, we show some computational results by setting m to different values, and compare the computational results to other method.

6.1 Explanation of data

We decided to run two sets of experiments. In the first set, we will show some benchmark instances published in the IBM Q platform [16]. In the second set, we created some instances ourselves. The names of the instances contains information about their shape, direction of the edges (random or regular) and the number of qubits. For each instance, we will report the circuit information, best solution found by time-indexed method with $m = Diam(G)$ (column "TI-Diam(G)"), best solution found by time-indexed method with $m = UB_{rt(G)}$ (column "TI-rt(G)") and the best solution found by Kendall-Tau based method (column "KT"). For each solution, we report the best value z^* and time needed for the solver to find the solution. For solutions found by the time-indexed method, we also report the corresponding m value used to solve the problem.

6.2 Set-up

The mathematical models of both time-indexed method and Kendall-Tau based method were implemented in Python and solved with the commercial solver Gurobi 9.1 through the Python interface. All instances are solved to the best possible solution for each corresponding method. The evaluation was conducted using up to 16 threads of 2.6 GHz each, working with 16 GB of RAM. We manually set the computational time limit to 3 hours for all instances.

6.3 Benchmark Instances Experiment

Instance Name	**Circuit Info**			**TI-Diam(G)**			**TI-rt(G)**			**KT**	
	arcs	qubits	gates	z^*	time (s)	m	z^*	time (s)	m	z^*	time (s)
out-circle-rand-q5-2	5	5	30	353	9.6	2	353	51.5	7	387	1.15
out-circle-reg-q5-0	5	5	30	354	12.9	2	354	616.5	7	421	1.05
out-linear-reg-q5-1	4	5	30	386	32.1	4	386	267.8	7	386	1.1
out-ibmqx2-q5-0	6	5	30	419	83.5	2	419	923.6	4	421	1.0
out-ibmqx2-q5-8	6	5	30	422	26.7	2	422	787.1	4	456	1.1
out-ibmqx2-q5-9	6	5	30	420	33.6	2	420	495.9	4	454	1.6
out-ibmqx4-q5-3	6	5	30	353	26.7	2	353	35	4	354	0.7

Table 2: Benchmark instances with 5 qubits.

As we can see from Table 2, the runtime for the solver to find optimal solution increased significantly when setting m to the upper bound of $rt(G)$, and it is hard to predict how runtime will change according to m values. The instances "out-ibmqx2-q5-8" and "out-ibmqx2-q5-9" were using the exact same connectivity graph, the only difference between them is the sequence of gates that we need to implement. While the former has a shorter runtime when $m = 2$, it is the latter which has a shorter runtime when $m = 4$. According to our analysis in section 3.3.1, all instances listed in Table 2 can reach global optimality by setting $m = Diam(G)$, because the costs match the ones with $m = UB_{rt(G)}$.

Note that using the Kendall-Tau method solve the problems within a few seconds, the the best solutions found by the solver are not global optimum. Except the linear architecture, all solutions found using this method are more costly than the ones calculated by time-indexed method. In other word, if the graph can be made linear by removing some edges, the Kendall-Tau method is a fast way to get at least a feasible solution. The resulting graph should be a linear spanning subgraph of the original G.

6.4 New Instance Experiment

In addition to the above benchmark instances published on the IBM Q platform, we created a few new instances for the purpose of demonstration. These experiments are run to illustrate m-value impact on optimal solutions, as discussed analytically in section 4. The names of these new instances contain information about the shape of their respective connectivity graphs, and the number of qubits in the circuit. Same information will be reported as the previous section.

Instance Name	**Circuit Info**			**TI-Diam(G)**			**TI-rt(G)**			**KT**	
	arcs	qubits	gates	z^*	time (s)	m	z^*	time (s)	m	z^*	time (s)
triangle-q3	3	3	10	104	0.0	1	104	0.1	2	171	0.0
rectangle-q4	4	4	30	320	0.8	2	320	1.1	3	388	0.2
complete-bipartite-q5	6	5	30	353	9.1	2	353	40	6	354	0.8
fish-q5-1	5	5	10	246	0.2	2	244	1.0	4	-	-
fish-q5-2	5	5	10	inf	-	2	306	1.3	4	-	-

Table 3: Addtional instances.

In Table 3, the instance "triangle-q3" indicates the circuit involves 3 qubits and is conducted on a triangular graph (Figure 4), the naming pattern is consistent for all instances. The "fish-shaped" graph is shown in Figure 5b. We chose to run two different circuits on the fish-shaped graph, the first one indicates that setting $m = Diam(G)$ does not allow us to reach optimal solution, because it is larger than the one found by setting $m = UB_{rt(G)}$. For the instance "fish-q5-2", setting $m = Diam(G)$ prevents the solver from finding even a feasible solution, as shown in Table 3. The fish-shaped graph does not have a linear spanning subgraph, thus cannot be solved using the Kendall-Tau method.

7 Conclusions

For relatively small circuits (consist of less than 10 qubits and no more than 50 gates), the time-indexed formulation is able to solve them to optimality within the preset computational time limit. For the benchmark instances shown in our first set of experiment, this method is able to reach optimality by setting m, the number of instants of each layer, to the diameter of connectivity graph G. However, as demonstrated analytically and illustrated in our second set of experiments, setting m to the diameter of the connectivity graph does not always guarantee we can find the global optimal solution, and it could even block the solver from finding any feasible solution to the problem. If we are able to figure out the routing number $rt(G)$ of the connectivity graph of a given problem, then setting m to the upper bound of $rt(G)$ will surely guarantee global optimal solution, but will also likely to increase runtime significantly. As can be seen from Table 2 and Table 3, the runtime does not scale polynomially with m values. However, the time-indexed

method appears to be impractical when solving large-scaled problems, with the solver failing to solve the problem optimally within the 3hrs time limit.

In contrast, the Kendall-Tau based approach has proven to be fast when solving instance circuits to optimality, even with larger problems. It can be helpful when we aim to find a feasible solution within a short amount of time. The challenge, however, is that not all graphs can be made linear by removing edges, and the solution might be unreasonably large compared to the global optimal solution found by time-indexed method.

We believe that future research should aim to solve this problem from two distinct directions: using graph theory to prove the minimum m value that guarantees global optimality, or think of a better way to model swap gates without using time indices. Ultimately, the goal is to increase the accuracy of finding the exact best mapping of qubit locations with minimal sacrifice in solution time.

In summary, the time-indexed ILP formulation is an elegant and precise way to figure out the exact optimal solution of a given circuit, but it requires manual input of m, whose best value deserves further study. In order to solve larger scaled problems, alternative ways to model the swap gates are needed.

References

[1] Elizabeth Gibney. Hello quantum world! google publishes landmark quantum supremacy claim. *Nature*, 574:461–462, 2019.

[2] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[3] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[4] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[5] M. Mottonen and J. J. Vartiainen. Decompositions of general quantum gates. *Trends in Quantum Computing Research*, 2006.

[6] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 10(3):355–377, Oct 2010.

[7] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. A heuristic for linear nearest neighbor realization of quantum circuits by swap gate insertion. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(1):62–72, 2016.

[8] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. *CoRR*, abs/1809.02573, 2018.

[9] Naser Mohammadzadeh, Tayebeh Bahreini, and Hossein Badri. Optimal ilp-based approach for gate location assignment and scheduling in quantum circuits. *Modelling and Simulation in Engineering*, 2014:571374, Feb 2014.

[10] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies, 2017.

[11] Maurizio Boccia, Adriano Masone, Antonio Sforza, and Claudio Sterle. *Swap Minimization in Nearest Neighbour Quantum Circuits: An ILP Formulation*, pages 255–265. Springer International Publishing, Cham, 2019.

[12] Vincent Cicirello. Kendall tau sequence distance: Extending kendall tau from ranks to sequences. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 7(23):163925, May 2020.

[13] Jesse Mulderij, Karen I. Aardal, Irina Chiscop, and Frank Phillipson. A polynomial size model with implicit swap gate counting for exact qubit reordering, 2020.

[14] S. C. Carlson. Königsberg bridge problem. *Encyclopedia Britannica*, July 2010.

[15] N. Alon, F. R. K. Chung, and R. L. Graham. Routing permutations on graphs via matchings. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 583–591, New York, NY, USA, 1993. Association for Computing Machinery.

[16] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, and Luciano Bello. Qiskit: An Open-source Framework for Quantum Computing, January 2019.