A Greedy Search Algorithm for the Construction of Architecture-Aware Pauli-Exponential-Clifford Circuits

Silas Dilkes* and Yao Tang†
Quantinuum, 13-15 Hills Road, CB2 1NL, Cambridge, United Kingdom
(Dated: July 28, 2024)

Quantum computers are typically presented as having a small, fixed instruction set of single-qubit and two-qubit quantum gates that they are able to run. Additionally, many devices have extra constraints on which pairs of qubits two-qubit quantum gates can be run on. To run a quantum circuit on such a device, provided quantum circuits must guarantee that all two-qubit gates are between permitted pairs of qubits.

Coupling constraints are represented as connected, undirected graphs $G = (V, E)$ where V is a set of vertices representing qubits and E is a set of unweighted edges (v_0, v_1) for $v_0, v_1 \in V$, that define allowed two-qubit interactions. Two-qubit quantum gates between non-adjacent qubits can be realised by adding SWAP gates 4 that swap the states adjacent qubits. Compiling general circuits to fit these constraints while minimising the number of two-qubit gates is a well studied area[5].

It has been shown that converting a quantum circuit into a *Pauli-Exponential-Clifford* circuit (see section A of the supplementary material) and then re-synthesising it into a sequence of Clifford gates and single-qubit rotation gates via a greedy search algorithm [12, 14], has been shown to be an effective method for reducing the number of two-qubit gates in quantum circuits. The quantum software kit `TKET` [15] has an optimisation pass `GreedyPauliSimp` that optimises circuits in this manner.

We update the greedy search algorithm for synthesising Pauli-Exponential-Clifford circuits to only choose two-qubit Clifford gates that respect the coupling constraints of a device. We modify the cost function from counting the number of non-Identity terms in a Pauli exponential, to counting the number of terminal nodes and Steiner nodes in a Steiner tree. The use of Steiner trees in architecture-aware synthesis techniques is well established, including synthesis of phase polynomial circuits and CX circuits [8, 9].

This method uses two greedy search algorithms; one for synthesising a series of Pauli Exponentials and one for synthesising a Clifford operation, with algorithm details provided in the supplementary material 1 5. We benchmark performance for a selection of Quantum Volume circuits, and for a set of small, linear reversible and chemistry circuits from a 2018 Qiskit Developer challenge (referred to as "Challenge Circuits"), routed on connectivity graphs for four IBMQ devices: 5-qubit Quito, 7-qubit Nairobi, 16-qubit Guadalupe and 27-qubit Mumbai.

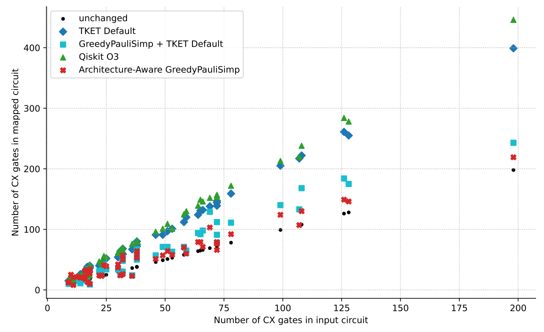
In nearly all cases, our proposed approach, referred to as "Architecture-Aware `GreedyPauli`", significantly outperforms both the default routing available in `TKET` `CXMappingPass` (used as the highest level of optimisation in Qiskit support in `TKET`) and optimising circuits with `GreedyPauliSimp` followed by `CXMappingPass`. Additionally, the Clifford synthesis sub-step matches state-of-the-art performance, finding the same asymptotic limits in CX count [16].

Qiskit compilation shows a noticeable contrast in performance between the two circuit classes, returning in many cases the highest compiled CX count for the "Challenge Circuits", but also returning in nearly all cases the lowest compiled CX count for the Quantum Volume circuits.

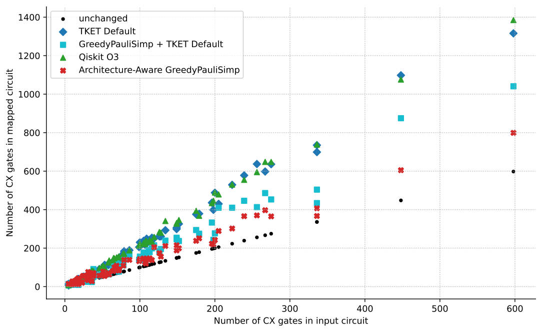
This is a promising performance baseline, showing that the optimisation performance of the greedy search algorithm used in `GreedyPauliSimp` can be updated effectively for compiling to connectivity graphs. However, due to runtime overhead, benchmark circuits are limited to 8 qubits (when the device has enough qubits to support them).

The cost function involves calculating numerous Steiner trees when selecting Clifford gates, and finding minimum Steiner trees is an NP-hard problem. Reducing this overhead is essential to make this technique viable for a general-purpose quantum circuit compiler. We are optimistic that this is achievable; by reducing the search space and using approximate Steiner trees or alternative cost functions during look-ahead, we aim to maintain good performance with reduced complexity in future iterations of this approach.

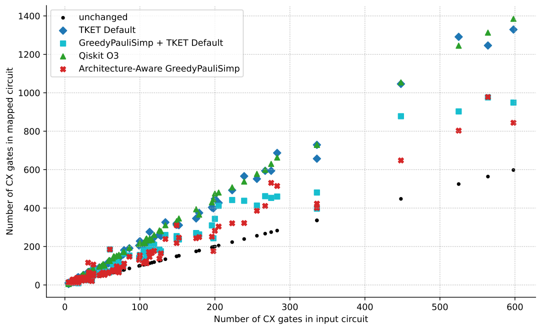
* silas.dilkes@quantinuum.com
† yao.tang@quantinuum.com



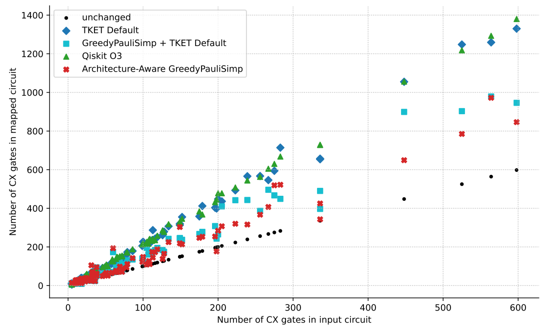
(a) IBMQ Quito Architecture



(b) IBM Nairobi Architecture

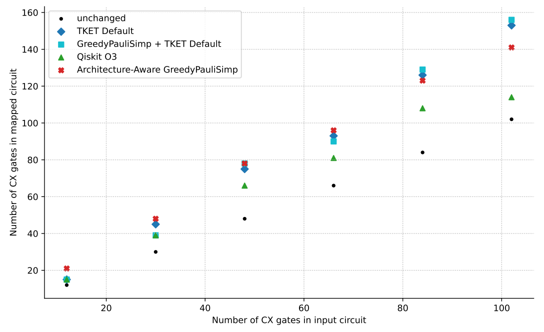


(c) IBMQ Guadalupe Architecture

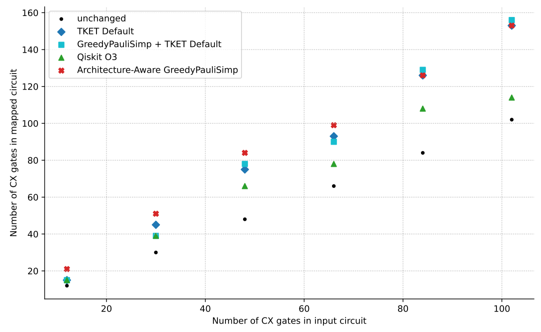


(d) IBMQ Mumbai Architecture

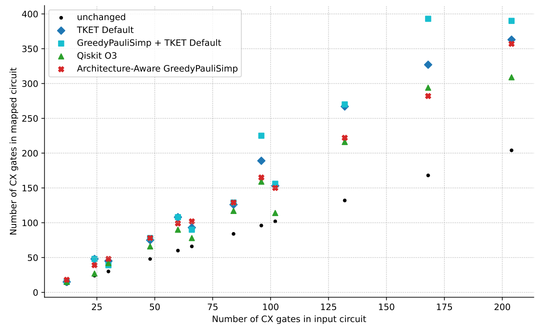
Figure 1: Graphs comparing default 1. TKET routing `CXMappingPass`, 2. a non-architecture-aware greedy search synthesis `GreedyPauliSimp` with `CXMappingPass`, 3. Qiskit level 3 compilation, 4. the proposed method, for the "Challenge Circuits" on four connectivity graphs for IBMQ devices.



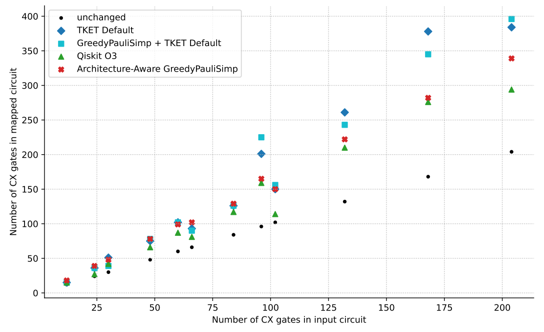
(a) IBMQ Quito Architecture



(b) IBM Nairobi Architecture



(c) IBMQ Guadalupe Architecture



(d) IBMQ Mumbai Architecture

Figure 2: Graphs comparing default 1. TKET routing `CXMappingPass`, 2. a non-architecture-aware greedy search synthesis `GreedyPauliSimp` with `CXMappingPass`, 3. Qiskit level 3 compilation, 4. the proposed method, for Quantum Volume circuits on four connectivity graphs for IBMQ devices.

SUPPLEMENTARY MATERIAL

A. Pauli-Exponential-Clifford Circuits

A *Pauli Exponential* is an n-qubit operation $e^{-i\alpha\pi\hat{P}}$ where $\hat{P} = \bigotimes_{i=0}^{n} P_i$ and $P_i \in \{I, X, Y, Z\}$ is a Pauli matrix. We use the phrase *Pauli String* to refer to the string of Pauli letters P comprising the tensor product in the exponential and we refer to the qubits with non-identity Pauli letters in a Pauli string as the support $S(P)$. A Pauli exponential can be implemented as a single qubit rotation gate $R_{\{X,Y,Z\}}(2\alpha) = e^{-i\alpha\pi\{X,Y,Z\}}$, if $|S(P)| = 1$ for the Pauli string P. Any quantum circuit with only quantum operations can be represented as a series of Pauli exponentials followed by a Clifford circuit $U = T.\prod_{j=0}^{k} e^{-i\alpha\pi_j\hat{P}_j}$, where T represents a Clifford circuit, by applying Clifford conjugations to Pauli exponentials. We refer to such circuits as being in Pauli-Exponential-Clifford form [4].

We propose producing connectivity graph valid circuits from Pauli-Exponential-Clifford circuits by splitting the problem into two sections, first solving for the non-Clifford Pauli exponentials, and second solving for the Clifford circuit. This order is necessary as the Clifford circuit will likely be modified by the solution to the first section.

B. Architecture-Aware Construction of Pauli Exponentials

We can reduce the number of non-identity Pauli letters in a Pauli string by applying Clifford conjugations to the term. For some Clifford operation T acting on the same qubits as the target Pauli exponential, $T.e^{-i\alpha\pi\hat{P}} = e^{\pm i\alpha\pi\hat{Q}}.T$, where $\hat{Q}$ comprises a different Pauli string Q and the angle of rotation may be flipped due to a phase change of -1. Therefore, given a circuit $U = \prod_{j=0}^{k} e^{-i\alpha_j\pi\hat{P}_j}$, we can produce a quantum circuit that respects some connectivity constraints $G(E, V)$ by repeatedly applying Clifford circuits that both respect G and convert P_j to Q_j such that $|S(Q_j)| = 1$.

We restrict the set of allowed Clifford operations to the nine two-qubit gates $A = \{e^{\frac{i\pi}{4}(I_0 - P_0)(I_1 - P_1)},\ P_0, P_1 \in \{X, Y, Z\}\}$ 3 as in [14]. Table I shows pairs of Pauli letters for which each Clifford is able to convert one of the letters to an identity. By inspection, one can see that every pair of non-identity Pauli letters can be mapped to a pair with a comprised of an Identity and a Pauli letter.

Clifford	Pre	Post	Clifford	Pre	Post	Clifford	Pre	Post
XX	XY	IY	**XY**	XX	IX	**XZ**	XX	IX
	XZ	IZ		XZ	IZ		XY	IY
	YX	YI		YY	YI		YZ	YI
	ZX	ZI		ZY	ZI		ZZ	ZI
YX	XX	XI	**YY**	XY	XI	**YZ**	XZ	XI
	YY	IY		YX	IX		YX	IX
	YZ	IZ		YZ	IZ		YY	IY
	ZX	ZI		ZY	ZI		ZZ	ZI
ZX	XX	XI	**ZY**	XY	XI	**ZZ**	XZ	XI
	YX	YI		YY	YI		YZ	YI
	ZY	IY		ZX	IX		ZX	IX
	ZZ	IZ		ZZ	IZ		ZY	IY

Table I: For each considered two-qubit Clifford gate, this table shows which four pairs of Pauli letters will have one letter converted to an Identity after a push through.

Without G, the operations in I are sufficient to reduce any Pauli string to a new string P such that $|S(P)| = 1$. Furthermore, the two-qubit Clifford cost of a Pauli exponential with Pauli string P is straightforwardly $|S(P)| - 1$.

With G, unless $S(P)$ forms a connected subgraph of the connectivity graph, Clifford gates that swap an identity between a pair of Pauli letters need to be considered. This can be achieved by a SWAP gate 4 or by pairs of

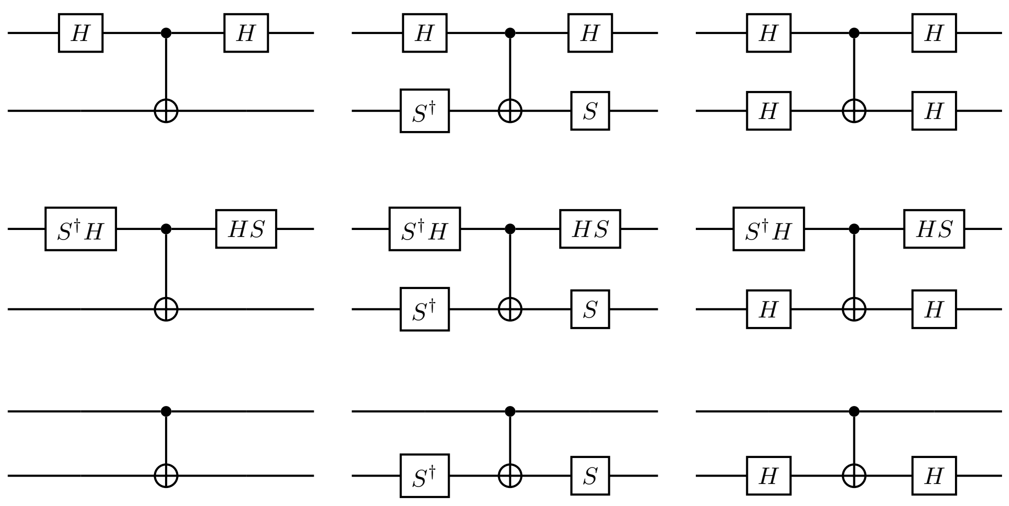


Figure 3: Clifford operations $e^{\frac{i\pi}{4}(I_0-P_0)(I_1-P_1)}$ used for conjugating Pauli exponentials. Rows correspond to X, Y and Z for P_0 respectively, and columns correspond to X, Y and Z for P_1 respectively. We refer to these Clifford operations as *XX*, *XY*, *XZ*, *YX*, *YY*, *YZ*, *ZX*, *ZY* and *ZZ*. Note that the provided constructions are written to display basis changes and in practice simplifications can be applied to these identities, such as by flipping the direction of the CX gate for *XZ* and removing the Hadamard gates.

Clifford gates as in table II.

Pre	Post	Clifford Pair	Pre	Post	Clifford Pair	Pre	Post	Clifford Pair
IZ	**ZI**	ZX, YZ ZX, XZ ZY, YZ ZY, XZ	**IZ**	**XI**	XX, YZ XX, ZZ XY, YZ XY, ZZ	**IZ**	**YI**	YY, XZ YY, ZZ YX, XZ YX, ZZ
IX	**ZI**	ZZ, XX ZZ, YX ZY, XX ZY, YX	**IX**	**XI**	XZ, YX XZ, ZX XY, YX XY, ZX	**IX**	**YI**	YY, XX YY, ZX YZ, XX YZ, ZX
IY	**ZI**	ZZ, XY ZZ, YY ZX, XY ZX, YY	**IY**	**XI**	XZ, YY XZ, ZY XX, YY XX, ZY	**IY**	**YI**	YX, XY YX, ZY YZ, XY YZ, ZY

Table II: Each Pauli letter pair IX/IY/IZ can be swapped to XI/YI/ZI with four possible pairs of Clifford operations.

To generate an accurate cost for realising a Pauli string P given $G = (V, E)$ we use Steiner trees. Given a set of vertices $W \subseteq V$, a Steiner Tree is a subgraph of G with vertices V_T such that $W \subseteq V_T$ and the number of edges in the Steiner Tree subgraph is minimised. Then, the cost of implementing P given G is $|W| + 2|V_T \setminus W| - 1$, where the nodes W are referred to as terminals and the nodes $V_T \setminus W$ are Steiner nodes. Steiner tree's are commonly used in similar techniques [8, 9].

Let's consider a brief example to explain the cost. Consider a connectivity graph corresponding to a 3x3 square grid with $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ and $E = \{(0, 1), (1, 2), (0, 3), (1, 4), (2, 5), (3, 4), (4, 5), (3, 6), (4, 7), (5, 8), (6, 7), (7, 8)\}$, and a Pauli exponential with string $P = I_0Y_1Z_2I_3X_4I_5Y_6Z_7Z_8$. Given a set of vertices $W = \{1, 2, 4, 6, 7, 8\}$ from $S(P)$, we can find the Steiner Tree as highlighted in red in figure 7a with edges $\{(1, 2), (1, 4), (4, 7), (6, 7), (7, 8)\}$. As $|V_T \setminus W| = 0$, we can reduce $|S(P)|$ to 1 with 5 Clifford gates; $[X_1Z_2, Y_1Z_4, Y_6X_7, X_4Y_7, Y_7X_8]$ is one example that produces the string $I_0I_1I_2I_3I_4I_5I_6Z_7I_8$, corresponding to

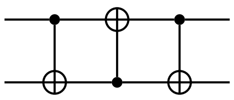


Figure 4: SWAP gate construction as three CX gates.

a Pauli exponential that can be implemented with a single R_z gate.

Alternatively, consider a Pauli exponential with string $P = I_0Y_1Z_2I_3I_4I_5Y_6Z_7Z_8$ as in 7b. With one Steiner node, one swap between an identity and a Pauli letter will be required to reduce the support to 1, meaning 6 Clifford gates are required: $[X_1Z_2, Z_1X_4, Y_1Y_4, Y_6X_7, X_4Y_7, Y_7X_8]$ is one example that produces the string $I_0I_1I_2I_3I_4I_5I_6Z_7I_8$, similarly returning a Pauli exponential that can be implemented with a single R_z gate.

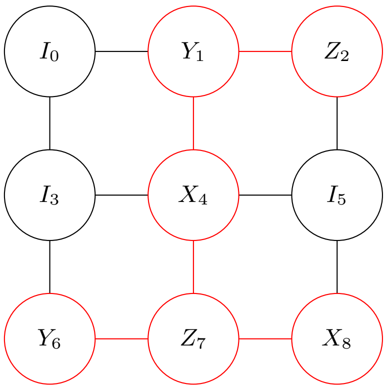


Figure 5: An overlay of $P = I_0Y_1Z_2I_3X_4I_5Y_6Z_7Z_8$ on a 3x3 Square graph, with the connected subgraph produced by $S(P)$ highlighted by in red.

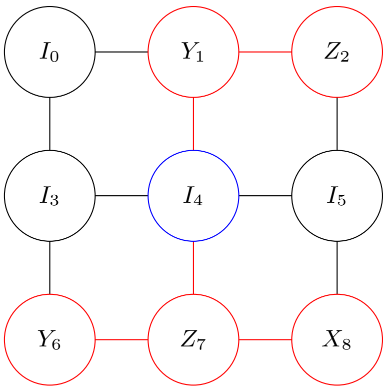


Figure 6: An overlay of $P = I_0Y_1Z_2I_3I_4I_5Y_6Z_7Z_8$ on a 3x3 Square graph, with a possible Steiner tree highlighted in red.

It is clear in both examples that there are many possible minimal sequences of Clifford gates that can reduce the support of a Pauli string to 1.

We produce architecture permitted circuits for $U = \prod_{j=0}^{k} e^{-i\alpha_j \pi \hat{P}_j}$ with a greedy search with look-ahead [12, 13].

Algorithm 1: Greedy synthesis of Pauli exponentials

```
Data: Pauli exponentials $\hat{\mathbf{P}}$, Clifford tableau $T$
Result: Circuit C
while $\hat{\mathbf{P}}$ is not empty do
    $P \leftarrow \hat{\mathbf{P}}[0]$;
    if $S(P) == 1$ then
        C.add($P$);
        $\hat{\mathbf{P}}$.pop($P$);
        continue;
    end
    min_pairs $\leftarrow$ arg $\min_{a,b\in P, a\neq b} dist(a,b)$;
    candidates $\leftarrow$ [] ;
    for $a, b \in$ min_pairs do
        if dist(a,b) == 1 then
            candidates $\leftarrow$ candidates + [AdjacentCliffords(a,b)];
        else
            for $c, d \in$ PertinentSwaps(a,b) do
                candidates $\leftarrow$ candidates + [SwapCliffords(c,d)];
            end
    end
    min_gate $\leftarrow$ Select(candidates, $\hat{\mathbf{P}}$);
    C.add(min_gate);
    Update($\hat{\mathbf{P}}$, min_gate);
    Update($T$, min_gate);
end
```

AdjacentCliffords returns all Clifford gates capable of reducing two non-identity Pauli letters to a single letter, as in I. When given a non-identity Pauli letter and an identity letter, **SwapCliffords** returns all Clifford gates that can swap the identity letter as in Table II, along with the SWAP gate 4.

PertinentSwaps returns all possible edge swaps that can bring two letters closer, as in Table II.

Select evaluates a list of candidate Clifford gates against the remaining Pauli exponential. For each Clifford gate, it calculates a weighted cost based on its impact on the remaining rotations, determined by constructing Steiner trees. The gate with the lowest cost is selected.

Rather than limiting gate candidates to a single Pauli exponential, we can broaden our search to include all dependency-free rotations. However, this approach risks not-terminating due to infinite loops as the minimum pair distance may not decrease. To address this, we exclude any rotation whose overall support increases after applying a selected gate.

C. Architecture-Aware Construction of Clifford circuits

Clifford operations in circuits are often represented by a Clifford Tableau [1], with most synthesis methods for Clifford circuits converting into a Clifford Tableau and then synthesising a new circuit using some normal form [3, 6, 7, 10, 11]. Often these methods proceed by updating an inverted tableau with permitted Clifford operations until it is reduced to the identity. In our approach, instead of constructing a normal form, we reduce each qubit to an identity sequentially, which has precedence in [2, 16].

If a Clifford operation is equivalent to the identity then the propagation of any Pauli exponential through it will leave the Pauli exponential unchanged. If X and Z Pauli letters on a qubit are propagated to a pair of anti-commuting Pauli letters on any qubit, then the Clifford operation is only single qubit Clifford gates and wire swaps away from an identity.

To construct a Clifford circuit, we propagate a pair of single qubit Pauli strings, each containing a single X Pauli and a single Z Pauli on the same qubit—throug, a reversed Clifford tableau. This generates two new Pauli strings. We then apply Clifford operations to these new strings until they are reduced to a pair of anti-commuting Pauli letters on the same qubit. We refer to the propagated strings as a Pauli-Propagation-Pair, a list of pairs of

Pauli letters, where each pair is categorized as either anti-commuting (**A**), if the Pauli letters on the same qubit anti-commute; commuting (**C**), if the Pauli letters commute; or identity (**Id**), if both strings are I.

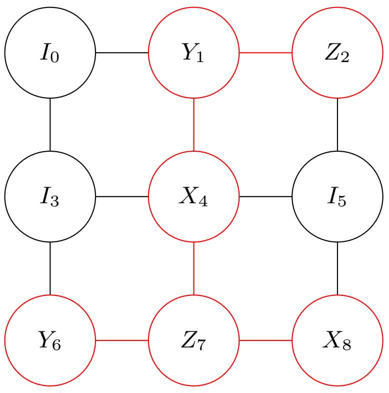
As with the synthesis of Pauli exponentials, we define sets of Clifford operations to act on pairs of qubits, where each qubit is classified as either **A** (anti-commuting), **C** (commuting), or **Id** (identity). These Clifford operations are organised into performing the following actions on these pairs of qubits:

1. $(C, A) \rightarrow (Id, A)$, for all (C, A), see III
2. $(C, A) \rightarrow (A, C)$, for all (C, A), see IV
3. $(Id, C) \rightarrow (C, Id)$, for all (Id, C), see II
4. $(C, C) \rightarrow (C, Id)$, for some (C, C), see V
5. $(A, Id) \rightarrow (Id, A)$, for all (A, Id), see 4
6. $(A, Id) \rightarrow (C, A)$, for all (A, Id), see VI
7. $(A, A) \rightarrow (C, C)$, for all (A, A), see VII

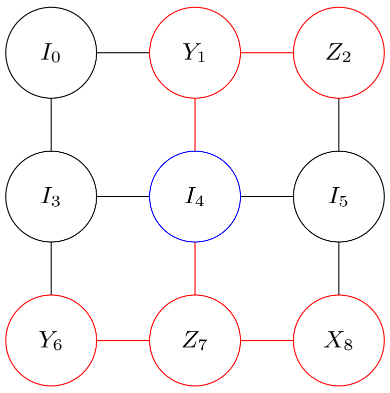
While the first rule would be sufficient for non-architecture-aware constructions, Clifford operations from all 6 operation types may be required to implement a connectivity graph respecting Clifford circuit.

Consider a short example to see how a pair of propagated Pauli strings can be reduced to a single anti-commuting entry. Assume that after propagation we have a pair of Pauli strings $P_X = I_0X_1Y_2X_3I_4I_5Y_6X_7Z_8$ and $P_Z = I_0Y_1Z_2X_3I_4I_5Y_6Y_7I_8$. Figure 7a strings overlaid on a 3x3 square grid architecture, with **A** vertices highlighted in red and **C** vertices highlighted in blue.

One possible minimum Steiner tree is shown in figure 7b, which can be solved with 14 Clifford gates: $[X_3X_6, Y_6Z_7, Y_7Z_8, X_4X_7, Y_4Y_7, X_4X_7, X_1X_4, Z_1Z_2, Y_1Y_2, Y_1Y_4, S_1, Z_0X_1, Z_1X_0, Z_0X_1]$, as in appendix B. Section E shows a selection of the Steiner tree's produced by this series of Clifford gates.



(a) An overlay of $P = I_0Y_1Z_2I_3X_4I_5Y_6Z_7Z_8$ on a 3x3 Square graph, with the connected subgraph produced by $S(P)$ highlighted in red.



(b) An overlay of $P = I_0Y_1Z_2I_3I_4I_5Y_6Z_7Z_8$ on a 3x3 Square graph, with a possible Steiner tree highlighted in red.

Figure 7: Comparison of the Pauli string overlays on a 3x3 square grid architecture.

For general Clifford operations we synthesise circuits using a greedy search with look-ahead as outlined in algorithms 2-5.

Algorithm 2: Reduce

```
Data: Tree $s$
Result: Bool if $s$ can be reduced
1  success ← False;
2  for $n \in$ s.leafs() do
3      $p \leftarrow n$.parent;
4      if type(n) == Id then
5          success | = True;
6      else if type(p) == A and type(n) == C then
7          success | = AC_AI($p, n$);
8      else if type(p) == C and type(n) == C then
           // not always possible
9          success | = CC_CI($p, n$);
10     else if type(p) == Id and type(n) == C then
11         success | = IC_CI($p, n$);
12     else
13         continue;
14     $T$.remove($n$)
15 end
```

Algorithm 3: ExpandA

```
Data: Tree $s$, done nodes $N$
Result: Bool success
1  if s has no C leaves then
2      success ← False;
3      return;
4  end
5  $n \leftarrow s$.C_leafs()[0];
6  $a \leftarrow \arg\ \min_{a \in T, type(A)=A} dist(a, n)$;
7  $b \leftarrow \arg\ \min_{b \in neighbour(a)} dist(b, n)$;
8  if type(b) == Id then
9      if $b \in N.values()$ then
           // if b is the destination of a previously fixed propagation we use a SWAP gate
10         success ← AI_IA($a, b$);
11         $N[b] \leftarrow a$;
12     else
13         success ← AI_CA($a, b$);
14 else if type(b) == C then
15     success ← AC_CA($a, b$);
```

Algorithm 4: ContractA

```
Data: Tree $s$
1  $n \leftarrow s$.A_leafs()[0];
2  $p \leftarrow n$.parent;
3  if type(p) == A then
4      AA_CC($p, n$);
5  else if type(b) == C then
6      CA_AC($p, n$);
7  else
8      IA_AI($p, n$);
```

Algorithm 5: Clifford Synthesis

```
Data: Tableau $T$
$N \leftarrow \{\}$;
while $T$ not done do
    $r \leftarrow$ SelectRow(T);
    // each row represents a propagation
    $s \leftarrow SteinerTree(r)$;
    // tree synthesis
    while $s.size() > 1$ do
        while Reduce(s) do
            repeat;
        end
        if ExpandA(s, N) then
            continue;
        end
        Contract(s);
    end
    N.add(s.nodes[0]);
end
```

`SelectRow` is performed by testing out tree synthesis for all remaining rows and then picking the cheapest. AC_AI, CC_CI, IC_CI, AI_IA, AC_CA and AA_CC are as in 7.

D. Pair Propagation for Clifford Synthesis

We share selection of Clifford operations for constructing architecture-aware Clifford gates, as described in Section C.

A	**C**	Clifford	**A**	**C**	Clifford	**A**	**C**	Clifford
	XX	ZX		XX	YX		XX	ZX
	XI	YX		XI	ZX		XI	XX
	YY	ZY		YY	YY		YY	ZY
XY	YI	YY	XZ	YI	ZY	YX	YI	XY
	ZZ	ZZ		ZZ	YZ		ZZ	ZZ
	ZI	YZ		ZI	ZZ		ZI	XZ
	IX	XX		IX	XX		IX	YX
	IY	XY		IY	XY		IY	YY
	IZ	XZ		IZ	XZ		IZ	YZ
	XX	XX		XX	YX		XX	XX
	XI	ZX		XI	XX		XI	YX
	YY	XY		YY	YY		YY	XY
YZ	YI	ZY	ZX	YI	XY	ZY	YI	YY
	ZZ	XZ		ZZ	YZ		ZZ	XZ
	ZI	ZZ		ZI	XZ		ZI	YZ
	IX	YX		IX	ZX		IX	ZX
	IY	YY		IY	ZY		IY	ZY
	IZ	YZ		IZ	ZZ		IZ	ZZ

Table III: For each combination of anti-commuting pair of Paulis **A** and commuting pair of Paulis **C**, shares a Clifford operation that maps the commuting pair **C** to **Id**, or $(A, C) \rightarrow (A, Id)$. The resulting letters can be found from applying I.

A	**C**	Clifford	**A**	**C**	Clifford	**A**	**C**	Clifford
XY	XX	XY,XZ,YY,YZ	XZ	XX	XY,XZ,ZY,ZZ	YX	XX	XY,XZ,YY,YZ
	XI	XY,XZ,ZY,ZZ		XI	XY,XZ,YY,YZ		XI	YY,YZ,ZY,ZZ
	YY	XX,XZ,YX,YZ		YY	XX,XZ,ZX,ZZ		YY	XX,XZ,YX,YZ
	YI	XX,XZ,ZX,ZZ		YI	XX,XZ,YX,YZ		YI	YX,YZ,ZX,ZZ
	ZZ	XX,XY,YX,YY		ZZ	XX,XY,ZX,ZY		ZZ	XX,XY,YX,YY
	ZI	XX,XY,ZX,ZY		ZI	XX,XY,YX,YY		ZI	YX,YY,ZX,ZY
	IX	YY,YZ,ZY,ZZ		IX	YY,YZ,ZY,ZZ		IX	XY,XZ,ZY,ZZ
	IY	YX,YZ,ZX,ZZ		IY	YX,YZ,ZX,ZZ		IY	XX,XZ,ZX,ZZ
	IZ	YX,YY,ZX,ZY		IZ	YX,YY,ZX,ZY		IZ	XX,XY,ZX,ZY
YZ	XX	YY,YZ,ZY,ZZ	ZX	XX	XY,XZ,ZY,ZZ	ZY	XX	YY,YZ,ZY,ZZ
	XI	XY,XZ,YY,YZ		XI	YY,YZ,ZY,ZZ		XI	XY,XZ,ZY,ZZ
	YY	YX,YZ,ZX,ZZ		YY	XX,XZ,ZX,ZZ		YY	YX,YZ,ZX,ZZ
	YI	XX,XZ,YX,YZ		YI	YX,YZ,ZX,ZZ		YI	XX,XZ,ZX,ZZ
	ZZ	YX,YY,ZX,ZY		ZZ	XX,XY,ZX,ZY		ZZ	YX,YY,ZX,ZY
	ZI	XX,XY,YX,YY		ZI	YX,YY,ZX,ZY		ZI	XX,XY,ZX,ZY
	IX	XY,XZ,ZY,ZZ		IX	XY,XZ,YY,YZ		IX	XY,XZ,YY,YZ
	IY	XX,XZ,ZX,ZZ		IY	XX,XZ,YX,YZ		IY	XX,XZ,YX,YZ
	IZ	XX,XY,ZX,ZY		IZ	XX,XY,YX,YY		IZ	XX,XY,YX,YY

Table IV: For each combination of anti-commuting pair of Paulis **A** and commuting pair of Paulis **C**, shares four possible Clifford operations that maps **A** to some **C** and **C** to some **A**, or $(C, A) \to (A, C)$. The resulting letters can be found from applying I.

C	**C**	Clifford	**C**	**C**	Clifford	**C**	**C**	Clifford
XX	XX	YX,ZX	XI	XI	YX,ZX	YY	XX	XX,ZX
	YY	YY,ZY		YI	YY,ZY		YY	XY,ZY
	ZZ	YZ,ZZ		ZI	YZ,ZZ		ZZ	XZ,ZZ
YI	XI	XX,ZX	ZZ	XX	XX,YX	ZI	XI	XX,YX
	YI	XY,ZY		YY	XY,YY		YI	XY,Y
	ZI	XZ,ZZ		ZZ	XZ,YZ		ZI	XZ,YZ
IX	IX	YX,ZX	IY	IX	XX,ZX	IZ	IX	XX,YX
	IY	YY,ZY		IY	XY,ZY		IY	XY,YY
	IZ	YZ,ZZ		IZ	XZ,ZZ		IZ	XZ,YZ

Table V: Combinations of pairs of commuting pairs of Pauli letters for which one of the provided two Clifford operations will reduce the second commuting pair to **Id**, or $(C, C) \to (C, Id)$. The resulting letters can be found from applying I.

Clifford 0	Clifford 1	Clifford 0	Clifford 1	Clifford 0	Clifford 1	Clifford 0	Clifford 1
XX	YY	XX	YZ	XX	ZY	XX	ZZ
XY	YX	XY	YZ	XY	ZX	XY	ZZ
XZ	YX	XZ	YY	XZ	ZX	XZ	ZY
YX	XY	YX	XZ	YX	ZY	YX	ZZ
YY	XX	YY	XZ	YY	ZX	YY	ZZ
YZ	XX	YZ	XY	YZ	ZX	YZ	ZY
ZX	XY	ZX	XZ	ZX	YY	ZX	YZ
ZY	XX	ZY	XZ	ZY	YX	ZY	YZ
ZZ	XX	ZZ	XY	ZZ	YX	ZZ	YY

Table VI: Any of these pairs of Clifford gates applied in sequence will convert $(A, Id) \to (C, A)$. The resulting letters can be found from applying II.

A	**A**	Clifford	**A**	**A**	Clifford	**A**	**A**	Clifford
XY	XY	XY,XZ,YX,YZ,ZX,ZY	XZ	XY	XY,XZ,YX,YY,ZX,ZZ	YX	XY	XX,XZ,YY,YZ,ZX,ZY
	XZ	XY,XZ,YX,YY,ZX,ZZ		XZ	XY,XZ,YX,YZ,ZX,ZY		XZ	XX,XY,YY,YZ,ZX,ZZ
	YX	XX,XZ,YY,YZ,ZX,ZY		YX	XX,XZ,YX,YY,ZY,ZZ		YX	XY,XZ,YX,YZ,ZX,ZY
	YZ	XX,XZ,YX,YY,ZY,ZZ		YZ	XX,XZ,YY,YZ,ZX,ZY		YZ	XX,XY,YX,YZ,ZY,ZZ
	ZX	XX,XY,YY,YZ,ZX,ZZ		ZX	XX,XY,YX,YZ,ZY,ZZ		ZX	XY,XZ,YX,YY,ZX,ZZ
	ZY	XX,XY,YX,YZ,ZY,ZZ		ZY	XX,XY,YY,YZ,ZX,ZZ		ZY	XX,XZ,YX,YY,ZY,ZZ
YZ	XY	XX,XY,YY,YZ,ZX,ZZ	ZX	XY	XX,XZ,YX,YY,ZY,ZZ	ZY	XY	XX,XY,YX,YZ,ZY,ZZ
	XZ	XX,XZ,YY,YZ,ZX,ZY		XZ	XX,XY,YX,YZ,ZY,ZZ		XZ	XX,XZ,YX,YY,ZY,ZZ
	YX	XX,XY,YX,YZ,ZY,ZZ		YX	XY,XZ,YX,YY,ZX,ZZ		YX	XX,XY,YY,YZ,ZX,ZZ
	YZ	XY,XZ,YX,YZ,ZX,ZY		YZ	XX,XY,YY,YZ,ZX,ZZ		YZ	XY,XZ,YX,YY,ZX,ZZ
	ZX	XX,XZ,YX,YY,ZY,ZZ		ZX	XY,XZ,YX,YZ,ZX,ZY		ZX	XX,XZ,YY,YZ,ZX,ZY
	ZY	XY,XZ,YX,YY,ZX,ZZ		ZY	XX,XZ,YY,YZ,ZX,ZY		ZY	XY,XZ,YX,YZ,ZX,ZY

Table VII: For each combination of pair of anti-commuting pair of Paulis (A, A) shares six possible Clifford operations that maps $(A, A) \to (C, C)$. The resulting letters can be found from applying I.

E. Clifford Synthesis Example

We share a selection of Steiner tree diagrams showing how reducing a pair of Pauli strings produced by single X and Z propagation to the identity can be completed using table look ups.

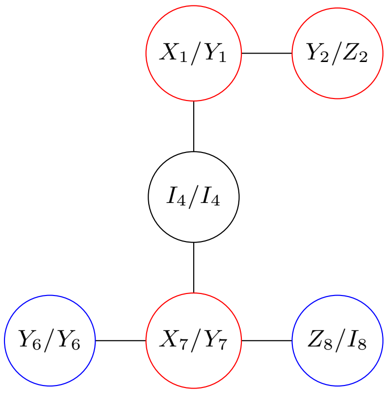


Figure 8: A possible Steiner tree from the graph after applying X_3X_6.

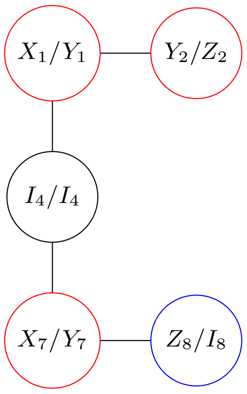


Figure 9: Steiner tree after applying Y_6Y_7.

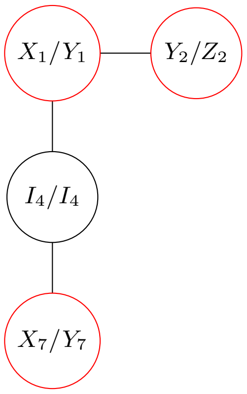


Figure 10: Steiner tree after applying Y_7Y_8.

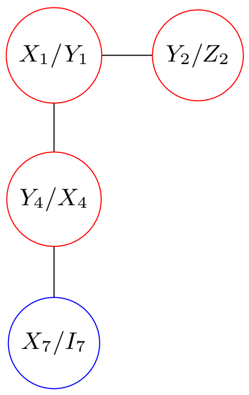


Figure 11: Steiner tree after applying X_4X_7 and Y_4Y_7.

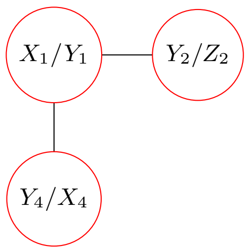


Figure 12: Steiner tree after applying X_4X_7.

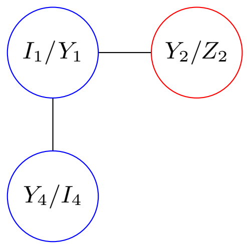


Figure 13: Steiner tree after applying X_1X_4.

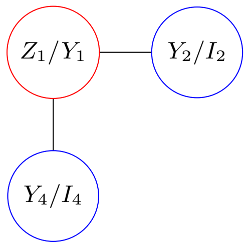


Figure 14: Steiner tree after applying Z_1Z_2.

[1] Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. Phys. Rev. A **70**, 052328 (Nov 2004). doi:10.1103/PhysRevA.70.052328, `https://link.aps.`

org/doi/10.1103/PhysRevA.70.052328

[2] van den Berg, E.: A simple method for sampling random clifford operators (2021), https://arxiv.org/abs/2008.06011

[3] Bravyi, S., Maslov, D.: Hadamard-free circuits expose the structure of the clifford group. IEEE Transactions on Information Theory **67**(7), 4546–4563 (Jul 2021). doi:10.1109/tit.2021.3081415, http://dx.doi.org/10.1109/TIT.2021.3081415

[4] Cole, O.: Quantum circuit optimisation through stabilizer reduction of pauli exponentials (2022), https://www.cs.ox.ac.uk/people/aleks.kissinger/theses/cole-thesis.pdf

[5] Cowtan, A., Dilkes, S., Duncan, R., Krajenbrink, A., Simmons, W., Sivarajah, S.: On the qubit routing problem. In: Proceedings of the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 135, pp. 5:1–5:32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). doi:10.4230/LIPIcs.TQC.2019.5, https://doi.org/10.4230/LIPIcs.TQC.2019.5

[6] Dehaene, J., De Moor, B.: Clifford group, stabilizer states, and linear and quadratic operations over gf(2). Phys. Rev. A **68**, 042318 (Oct 2003). doi:10.1103/PhysRevA.68.042318, https://link.aps.org/doi/10.1103/PhysRevA.68.042318

[7] Duncan, R., Kissinger, A., Perdrix, S., van de Wetering, J.: Graph-theoretic simplification of quantum circuits with the zx-calculus. Quantum **4**, 279 (Jun 2020). doi:10.22331/q-2020-06-04-279, http://dx.doi.org/10.22331/q-2020-06-04-279

[8] Meijer-van de Griend, A., Li, S.M.: Dynamic qubit routing with cnot circuit synthesis for quantum compilation. Electronic Proceedings in Theoretical Computer Science **394**, 363–399 (Nov 2023). doi:10.4204/eptcs.394.18, http://dx.doi.org/10.4204/EPTCS.394.18

[9] Martiel, S., Brugière, T.G.d.: Architecture aware compilation of quantum circuits via lazy synthesis. Quantum **6**, 729 (Jun 2022). doi:10.22331/q-2022-06-07-729, https://doi.org/10.22331/q-2022-06-07-729

[10] Maslov, D., Roetteler, M.: Shorter stabilizer circuits via bruhat decomposition and quantum circuit transformations. IEEE Transactions on Information Theory **64**(7), 4729–4738 (Jul 2018). doi:10.1109/tit.2018.2825602, http://dx.doi.org/10.1109/TIT.2018.2825602

[11] den Nest, M.V.: Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond (2009), https://arxiv.org/abs/0811.0898

[12] Paykin, J., Schmitz, A.T., Ibrahim, M., Wu, X.C., Matsuura, A.Y.: Pcoast: A pauli-based quantum circuit optimization framework (2023), https://arxiv.org/abs/2305.10966

[13] Schmitz, A.T., Ibrahim, M., Sawaya, N.P.D., Guerreschi, G.G., Paykin, J., Wu, X.C., Matsuura, A.Y.: Optimization at the interface of unitary and non-unitary quantum operations in pcoast (2023), https://arxiv.org/abs/2305.09843

[14] Schmitz, A.T., Sawaya, N.P.D., Johri, S., Matsuura, A.Y.: Graph optimization perspective for low-depth trotter-suzuki decomposition. Phys. Rev. A **109**, 042418 (Apr 2024). doi:10.1103/PhysRevA.109.042418, https://link.aps.org/doi/10.1103/PhysRevA.109.042418

[15] Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., Duncan, R.: t|ket⟩: a retargetable compiler for nisq devices. Quantum Science and Technology **6**(1), 014003 (Nov 2020). doi:10.1088/2058-9565/ab8e92, http://dx.doi.org/10.1088/2058-9565/ab8e92

[16] Winderl, D., Huang, Q., van de Griend, A.M., Yeung, R.: Architecture-aware synthesis of stabilizer circuits from clifford tableaus (2023), https://arxiv.org/abs/2309.08972