ZX Polynomial Synthesis

David Winderl

April 23, 2022

1 Introduction

The Noisy intermediate state quantum devices era (NISQ era) in quantum computing can be seen as the era of quantum devices with 50-100 qubits, which provide restricted connectivity between qubits and nosy qubits with a limited coherence time [1]. This issue motivates efficient algorithms to reduce the number of gates applied to circuits in an automated way, similar to the simplification process of classical compilers. Various researchers have proposed methods that reduce the number of gates, and the depth of the quantum circuit [2, 3, 4, 5]. Nevertheless, providing a routed circuit, so a circuit that suffices the connectivity graph of the device was always considered a process that is done after optimization. Griend et al. [6] has proposed a recursive algorithm for synthesizing so-called Z polynomials. Moreover, they showed that optimization and routing in combination are pretty effective for Z Polynomials. Hence they proposed the avenue of future work to extend their algorithm to Pauli gadgets. This guided research aims to analyze the problem of general phase gadget synthesis. Herefore we provide two extensions to the algorithm by Griend et al. [6]. One extension focuses on splitting the ZX Polynomial into smaller commuting regions, while the other tries to identify blocking columns and resolve them beforehand. Next, we provide an algorithm based on the A* search heuristic. Finally, we compare our results to the Pauli-Gadet simplification-pass by tket combined with their internal routing strategy and our optimization strategy with pyzx concerning one qubit gates.

2 Preliminaries

2.1 ZX-Calculus

The ZX-calculus is a rigorous graphical language that can represent arbitrary linear maps. Hence one can always describe quantum circuits in terms of the ZX calculus. Moreover, synthesis is guaranteed if a certain structure (gflow) is present, as seen in [7, 3]. The ZX-Calculus consists of two types of tensors (so-called Z and X spiders) in a Penrose graphical notation similar to that of tensor networks, such that each connecting leg between spiders represents a contraction operation. See equation 1 for an example of an X and Z Spider and their corresponding tensor representation.

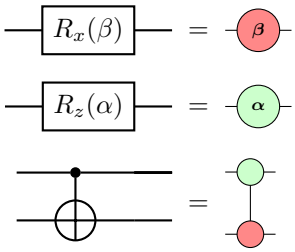
$$\begin{aligned} m \;\beta\; n &= |+\rangle^{\otimes m} \langle +|^{\otimes n} + e^{i\beta} |-\rangle^{\otimes m} \langle -|^{\otimes n} \\ m \;\alpha\; n &= |0\rangle^{\otimes m} \langle 0|^{\otimes n} + e^{i\alpha} |1\rangle^{\otimes m} \langle 1|^{\otimes n} \end{aligned} \tag{1}$$

Note that if the phase is 0 or 2π, it is a general convention to denote spiders as empty spiders as in equation 2.

$$\begin{aligned} m \;\; n &= |+\rangle^{\otimes m} \langle +|^{\otimes n} + |-\rangle^{\otimes m} \langle -|^{\otimes n} \\ m \;\; n &= |0\rangle^{\otimes m} \langle 0|^{\otimes n} + |1\rangle^{\otimes m} \langle 1|^{\otimes n} \end{aligned} \tag{2}$$

Since the ZX-Calculus is rigorous, various rewrite rules exist, not subject to this guided research. The exact rules can be found in Ref [8]. At last, it is relevant to note how to convert quantum circuits to ZX Calculus notation. A Z-spider with two legs corresponds to the R_z-Gate of quantum computing, and an X-Spider with two legs to the R_x gate. With that knowledge, we can represent any $\{R_z, R_x, CNOT\}$-Circuit in terms of the ZX-Calculus

by simply replacing the gates with equations 3, 4 and 5.



(3)

(4)

(5)

2.2 Phase and Pauli Gadets

2.2.1 Phase Gadets

According to the definition of Cowtan et al. [9], one can define Z-Phase gadgets as follows:

Definition 1 *The Z-Phase gadgets* $\Phi_z(\alpha, n) : \mathbb{C} \to \mathbb{C}$ *is a family of unitary maps with the recursive definition:*

$$\Phi_z(\alpha, n) = \textit{CNOT(n, n-1)}; I \otimes \Phi_z(\alpha, n); \textit{CNOT(n, n-1)}$$
$$\Phi_z(\alpha, 0) = R_z(\alpha)$$

Here CNOT(n, n-1) is the CNOT-gate applied to qubit n and n-1. Similarly, one can introduce an X-Phase gadget:

Definition 2 *The X-Phase gadets* $\Phi_x(\alpha, n) : \mathbb{C} \to \mathbb{C}$ *is a family of unitary maps, with the recursive definition:*

$$\Phi_x(\alpha, n) = \textit{CNOT(n-1, n)}; I \otimes \Phi_x(\alpha, n); \textit{CNOT(n-1, n)}$$
$$\Phi_x(\alpha, 0) = R_x(\alpha)$$

We can create phase gadgets on quantum circuits as CNOT ladders combined with a single rotation on the Z or X-Axis and then the reversed CNOT ladder. See Cowtan et al. [9] for an example of the recursive definition of a Z-phase gadget on a quantum circuit. For completeness, we present an example of how to convert an X-Phase gadget to ZX-Calculus (see Figure 1). We start with the representation in circuit form, seen in Figure 1. In order to represent this gadget in ZX-Calculus, we use the rules presented in Section 2.1, obtaining the left-hand side of Figure 2. We are then able to use the rewrite rules in [8] to convert it to the right-hand side representation. Due to the convenience of this form for the purposes of commuting the gadgets, we represent them in this form in this guided research. Intuitively phase gadgets can be understood as a higher-level language concept for certain

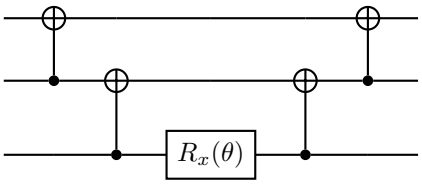


Figure 1: Quantum circuit, that describes a X phase gadget

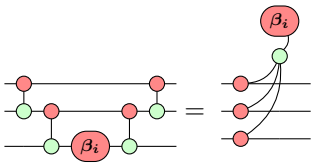
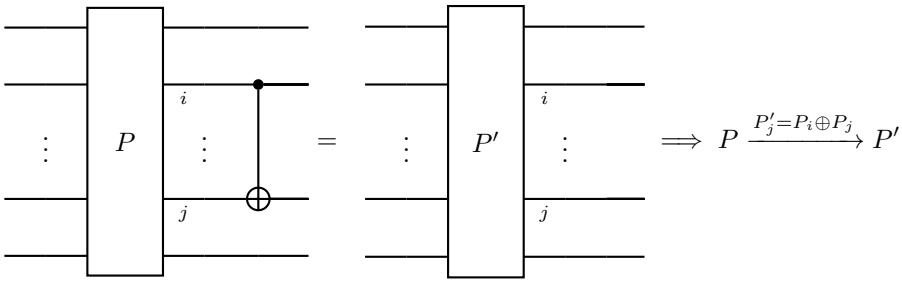


Figure 2: Representation of a X phase gadget in the ZX-Calculus. One can see on the left, the form derrived by the rules from section 2.1 and to the right a more convenient notation, used throughout this guided research.

quantum circuits, in the sense that they provide an abstraction to a combination of CNOT ladders with R_x or R_z gates. For instance, writing a variational ansatz with an operator consisting of the sum of the tensor-product of Paulis can be more expressive in terms of phase gadgets. We can combine multiple phase gadgets with a linear transformation (global parties) in the end. This concept will be referred to as phase Polynomial. Depending on the type of phase gadgets inside the phase Polynomial, we will refer to this concept as Z Polynomial, X Polynomial, or ZX Polynomial. Note that there exits the extension of Pauli gadgets to phase gadgets, where each leg can have a different type of X, Y, and Z. Those can be built with a ZX Polynomial and hence are left out of discussion in this guided research since a ZX Polynomial suffices in terms of universality.

2.3 Parity Maps

A parity map (in later context also parity matrix) is a linear reversible map on bit-strings. Such a parity map can be represented as a matrix in GF(2)[1]. It is well-known that such maps correspond to the action of purely CNOT sub-circuits in a quantum circuit [1]. They can be computed by the following equation:



(6)

Kissinger et. al [1] have given an algorithm, which is also able to synthesize a quantum circuit, out of a parity map utilizing Steiner-trees. Their algorithm is architecture aware, meaning that the quantum circuit will fit onto the connectivity graph. They named their algorithm *recursive steiner-gauss* and it is mentioned throughout this guided research as such.

2.4 Conversion of Quantum Circuits to ZX Polynomials

For quantum circuits, the Clifford+T group (Gates: $\{H, S, T, CNOT\}$) forms a universal gate set of a Quantum Circuit. One is able to synthesize arbitrary unitaries into a sequence of $R_x(\theta)$, $R_z(\alpha)$ and CNOT gates. Namely, by using a variant of the ZY-Decomposition:

$$U = e^{i\alpha} R_z(\beta) R_x(\gamma) R_z(\delta) \tag{7}$$

So by finding values for α, β, γ, and δ, we can re-base our (arbitrary) quantum circuit (up to a global phase) into a circuit consisting only of R_x, R_z and CNOT Gates. In the following, the decomposition for the one qubit gates from the Clifford+T group is given:

$$H = e^{i\frac{\pi}{2}} R_z(\frac{\pi}{2}) R_x(\frac{\pi}{2}) R_z(\frac{\pi}{2}) \tag{8}$$

$$S = e^{i\frac{\pi}{4}} R_z(\frac{\pi}{2}) R_x(0) R_z(0) \tag{9}$$

$$T = e^{i\frac{\pi}{8}} R_z(\frac{\pi}{4}) R_x(0) R_z(0) \tag{10}$$

$$\tag{11}$$

In the following the process, in order to convert any R_x, R_z, CNOT-Circuit into a ZX Polynomial is outlined:

1. Provide two parity maps Z-Parity maps and X-Parity maps [2]
2. Iterate overall quantum gates
3. In case of a R_z gate at qubit q, add a Z-Phase gadget with legs corresponding to the column q of the Z-parities
4. In case of a R_x gate at qubit q, add an X-Phase gadget with legs corresponding to the column q of the X-parities

[1] GF(2) is a field consisting of two numbers where addition and multiplication are well defined.

[2] We refer to a Z-Parity map if we perform a row addition in GF(2) from control to target. In contrast to an X-Parity map, if a row addition from target to control is done. This is due to the color duality of Z and X Phase gadgets

5. In case of a CNOT gate with control c and target t, add rows $c \to t$ on the Z-parities and $t \to c$ on the X-parities
6. In the end, the global parities of the ZX Polynomial are the Z-parities

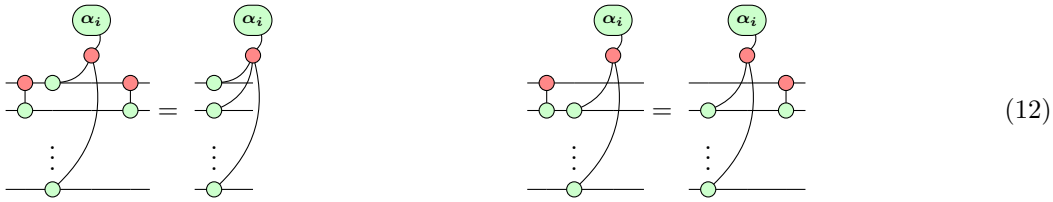
3 Synthesis Algorithms

The goal of architecture-aware ZX Polynomial synthesis is to create a quantum circuit from a ZX Polynomial, such that according to some connectivity graph, CNOTs are placed only onto specific qubits. A naive way of doing this would be to first decompose each phase gadget in the ZX polynomial according to its definition (definition 2 or definition 1) and later on route the circuit. Nevertheless, there exist algorithms that could provide an improvement in this process. Griend et al. [6] provided a heuristic algorithm for the architecture aware synthesis of Z Polynomials. They optimized the order of phase gadgets by propagating already reduced phase gadgets through the Z Polynomial. Their algorithm outperforms the current tket and staq implementation for routing. However, their approach natively does not work for synthesizing ZX Polynomials since one cannot commute an arbitrary X phase gadget through a Z phase gadget. This work mainly focuses on extending the approach by Griend et al. [6] to ZX Polynomials. We then later compare it to the approach by Cowtan et al. [9], who have proposed a simplification strategy that decomposes the CNOT ladders optimally and performs Clifford simplifications in between the phase gadgets. We used the *PauliSimp* transformation pass of the tket framework in order to implement the algorithm by Cowtan et al. [9]. For a fair comparison, the algorithm is routed by the routing pass of the tket framework.

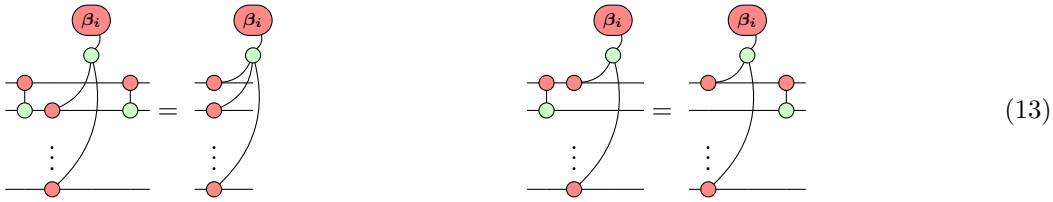
3.1 Algorithmic Preliminaries

3.1.1 Phase gadget rewrite rules

Propagation of CNOT Gates: One can propagate a CNOT through a X-Phase or Z-Phase gadget by equation 13 and 12. Essentially a leg is added or removed to the Z-phase gadget if and only if there is a leg on the target of the CNOT. For the X-Phase gadget, this is analogous but inverted by the color duality of the ZX-Calculus.

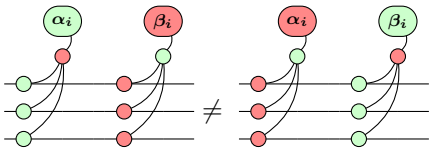


(12)

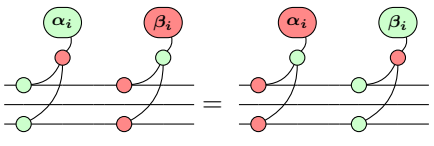


(13)

Commutation Rules for ZX-Phase gadets: In general, two ZX-Phase gadgets do not commute, and there is, to our knowledge, no general decomposition or another way to alter their position similar to a swap. Nevertheless, following [10], one can find that two ZX-phase-gadgets commute if and only if they are either of the same type or have an even number of legs on the same wires. So given equation 14, we can see that this commutation cannot take place by the theorem mentioned above. Whereas for equation 15, we can count an even amount of legs and hence commute the two phase-gadgets.



(14)



(15)

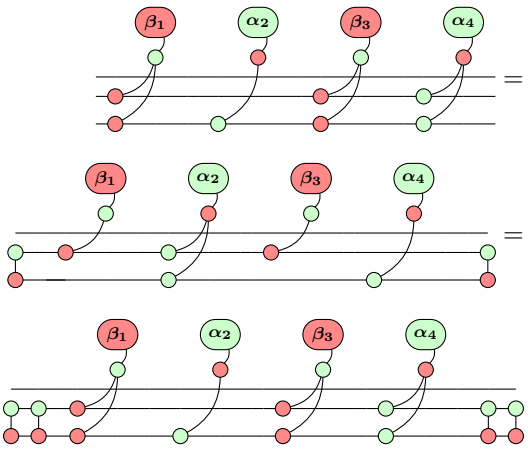


Figure 3: By incrementally decomposing the phase-gadet onto one qubit, we end up in an infinite loop. Even worse, if one would reduce the phase gadget α_2 onto qubit one, it would produce the same infinite loop with phase gadget α_4.

Reducing Z and X-Phase gadgets onto one qubit: For now, only the order of the phase gadgets was considered. Nevertheless, it is also essential to reconcile how to reduce a phase gadget onto one qubit. While 2 and 1 are indicating that this can always be done for the last leg of the phase gadget, a different approach is needed for the choice of an arbitrary qubit. Hence the CNOT Propagation rules 13, 12 can be applied as follows for the reduction on an arbitrary qubit:

(16)

For ZX Polynomial synthesis, we can propagate the CNOTs to the right of the reduced phase gadget through the rest of the Polynomial. In contrast, the CNOTs to the left need to remain constant.

Remark 1 *One could also try to propagate the left CNOTs through the left-most part of the ZX Polynomial. Nevertheless, as it turns out, there are limitations introduced by already reduced phase gadgets, which can lead to infinite loops of interactions that need to be resolved. The constraints and possibilities of such a method are still unknown to us and hence are left for future work. See figure 3 for such a infinite loop of interactions.*

3.1.2 Correspondence between Commutation of Phase gadets and Graph theory

As pointed out by Griend et al. [6] the task of synthesizing Z-Phase gadgets is equivalent to finding an optimal order of phase gadgets and finding an optimal placement of the remaining R_z-rotations. In this section, the resemblance between graph theory and the optimal order will be explained. Assume a graph $G = (V, E)$, where the vertices are described by the phase-gadgets of the phase polynomial combined with an empty "starting"-vertex S and an empty "stopping"-vertex G. The last vertices mark the entry point and exit point of the ZX-Diagram. An edge is created between two nodes, symbolizing that the two phase gadgets can be swapped. So in order to swap two phase gadgets, all gadgets in between need to commute with the two gadgets. [3] See Figure 4 as an example of the creation of such a graph. Also, assume (even though this is not trivial) that the weight of each edge is the "gain" one can achieve in this swap by optimal placement of rotations onto qubits. Finding an optimal placement then corresponds to finding the optimal hamiltonian path throughout the graph [4]. Given that path, the order of the vertices corresponds to a valid permutation of the phase gadgets. Within this framework, one can see why the heuristic algorithm by Griend et al. [6] tends to perform well on Z-Phase gadgets. Because their graph is fully connected, each combination of vertices reveals a hamiltonian path. In other words, one can entirely focus on the qubit placement without "wasting" computational time on the optimal permutation of phase gadgets.

[3] Note that the two phase gadgets also need to be swapped

[4] An Hamiltonian path is a path through a circuit, which visits each vertex exactly once.

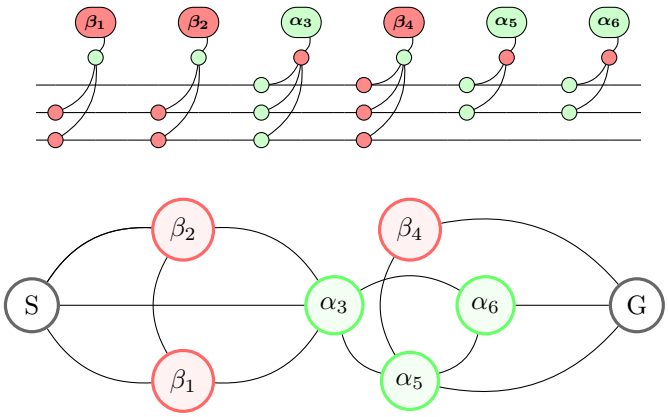


Figure 4: A ZX-Phase polynomial combined with the graph describing all possible permutations of the phase gadet above as hamiltonian paths from S to G.

3.1.3 Representation of the ZX Polynomial as a matrix

Griend et al. [6] represented their Z Polynomial as a binary matrix. In this context, an application of a CNOT had the effect of a row addition on the matrix. In order to generalize this concept, a matrix consisting of the elements $\{0, 1, 2\}$ is used. Each row represents the parity of its phase gadget. We can then apply the rules 13 and 12 in order to mimic a propagation of a CNOT through the ZX Polynomial.

3.2 A*-Approach

The A*-Algorithm is a heuristic search algorithm with applications in many fields due to its completeness, optimality, and optimal efficiency. It can be seen as a heuristic extension to Dijkstra's algorithm for optimal pathfinding. In order to use this method, one needs to define a heuristic function $h(x)$, which estimates the distance to the goal node. Then from the start node each possible path is extracted and the function $f(x) = g(x) + h(x)$ [5] is computed as cost. This process is done iteratively until the goal node is reached. Optimally is provided (either heuristically or guaranteed) by sorting the queue by $f(x)$ and by only picking the first best option. We labeled the area in front of all non-reduced phase gadgets like the 'frontier' [6], then identified the gadgets that could be propagated adjacent to the frontier. Hence, for each of those phase gadgets, the task boils down to finding a suitable placement of its rotation. We store these placements in a vector, with which we can generate an optimized circuit. See Figure 5 for a small example. An architecture-aware circuit can then be

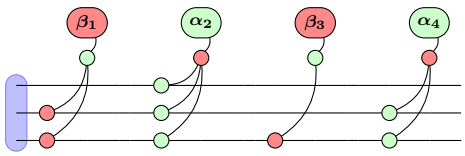


Figure 5: Example of the frontier in A* to reduce phase gadgets onto one qubit. Assume the Blue area as the frontier of the ZX Polynomial. Then one could select β_1 or α_2 as phase gadgets for reducing onto one qubit in the next step.

recreated out of this vector by reducing each phase-gadget onto the corresponding qubit. We can perform this task by applying equation 16. The CNOTs to the left will be summarized as a parity matrix and can then be re-synthesized by the *recursive steiner-gauss* algorithm. The CNOTs to the right will get propagated through the matrix and are applied to the global parties. Next, we will elaborate on evaluating the distance of one reduction and our heuristic function.

[5] $g(x)$ describes the distance from the start
[6] The blue area in Fig 5

3.2.1 distance of one reduction :

It is unfeasible to create a circuit on every evaluation of the distance function. So in order to calculate the distance of one step, we computed a *recursive steiner-gauss* for the CNOTs that are stored to the left and counted, which is represented by c_l. Next, we compute the *recursive steiner-gauss* on the global parties before (gp_o) and after (gp) propagation of the CNOTs to the right and count the amount of CNOTs. One can then find the cost of reducing this phase gadget as follows:

$$d = c_l + gp - gp_o \tag{17}$$

3.2.2 Heuristic Function

We used the number of parities left in the matrix as a heuristic function. One can calculate those by mapping all twos to ones and then summing up the matrix.

$$h(\text{zx\_matrix}, \text{columns}) = \sum_{i \in \text{columns}} \sum_{q \in qubits} \text{zx\_matrix}_{i,q} \tag{18}$$

3.2.3 General Approach

Since our distance function requires us to store the matrix of the phase gadget and the global parities, they would be stored in a possibly exponentially growing queue. Hence we limited the size of the queue to five elements. Given the distance and heuristic discussed above, one can find the algorithm outlined in 1

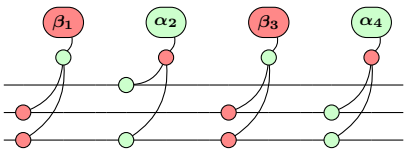
3.3 Extensions of the Synthesis Algorithm by Griend et al. [6]

Griend et al. [6] has proposed a recursive approach in order to synthesize Z-Phase gadgets. In order to do this, they represented the phase gadget as a binary matrix in a similar manner to section 3.1.3. A propagation of CNOT through the Z Polynomial corresponds to a row addition modulo two in their context. They then try to skew the binary matrix as far as possible by selecting the qubit with the most zeros or ones as possible and then grouping both into two submatrices P^0 and P^1 on their base recurse. For P^0, nothing is to be done on this particular qubit. Hence they apply their base recurse again. For P^1, they apply a CNOT so that most ones are removed from the binary matrix and then continue splitting again. Selecting targets and controls according to an architecture achieves architecture aware synthesis. Note that one could use this algorithm on the ZX Polynomial if a ZX Polynomial would only consist of commuting phase gadgets. The following will discuss two methods of extending the approach by Griend et al. [6] to ZX Polynomials. We reference their algorithm as Arianne's-synth since that was the name of the method provided in their implementation.

3.3.1 Extension: blocking-Arianne's-synth

Definition 3 *We can define a blocking column as the column in the matrix which does not commute through the region to the right.*

Take, for instance, the following ZX Polynomial in equation 19. One can see that the phase gadget α_2 does not commute with β_1. So we cannot propagate β_1 to the left to synthesize it with β_3 and α_4. Hence it is necessary to first reduce those two phase gadgets (α_2 and β_1) before the others to preserve the order of the ZX Polynomial.



(19)

This yields the extension to check before a base recurs whether or not there are blocking columns. Next, in each recursion step, one needs to see if a different step already reduced the visited columns.

3.3.2 Extention: split-Arianne's-synth

Recalling the graph model of section 3.1.2, we can find the commuting region by finding a CLIQUE [7] inside of the graph. This approach iteratively cuts the ZX Polynomial into commuting regions by finding a MAX-CLIQUE that contains the first node. Then each region is synthesized by the algorithm of Griend et al. [6] and the resulting global parities are propagated through the rest of the circuit.

[7] A CLIQUE in a Graph is a subset of vertices, such that an edge connects every two vertices in the subset

Algorithm 1: A* Algorithm for ZX Polynomial synthesis

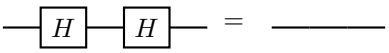
cm = // Generate the CNOT map by adding for each i,j the rows of a parity matrix and reducing it via steiner gauss.

def *objective_fun(assignment, matrix, gp))*
    gp_o = gp.copy();
    // Note that the extraction step has side effects on the matrix and global parities;
    cx = // Extract the Cnots of the current phase gadget;
    h = $\|cm \circ (\text{gp} - \text{gp}_o)\| + \sum_{(c,t)\in \text{CX}_l} cm_{c,t}$;
    d = $\sum_{i \leq pivot} \sum_q \text{columns})\text{matrix}_{i,q}$;
    return d, h

def *a_star_search(matrix, phases, global_parities, architecture)*
    best_queue = [([], 0.0, matrix, global_parities)];
    while $|best\_queue| > 0$ **do**
        // Current Assigmnent, distance matrix and global parities;
        c_q_a, c_dist, c_matrix, c_gp = best_queue.pop();
        if $|q\_assignment| = N$ **then**
            // Found a sensible solution ;
            return c_q_a
        end
        pivots = // Get the next phase gadgets, which can moved to the frontier ;
        for $pivot \in pivots$ **do**
            non_zero_q = // All Legs that are non zero in the current matrix;
            next_best = ∞;
            for $i \in non\_zero\_q$ **do**
                c_q_a_ = c_q_a + [i];
                c_matrix_ = c_matrix.copy();
                c_gp_ = c_gp_.copy();
                dist, h = objective_fun(c_q_a_, c_matrix_, c_gp_);
                new_dist = dist + c_dist;
                if $new\_dist + h \leq next\_best$ **then**
                    // Update the new next_best value
                end
            end
            // Insert the next_best solution for this gadget into the queue
        end
    end

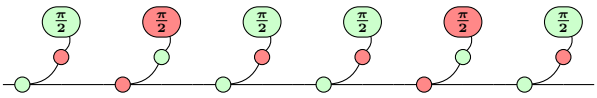
4 Optimization

So far only synthesis of ZX Polynomials was mentioned, nevertheless it is also worth noting, that by the procedure outlined in section 2.4, one can obtain a sub optimal ZX Polynomial. In order to make this clear, consider the one qubit case of two H-gates:



(20)

We can rewrite this as phase gadgets as follows:

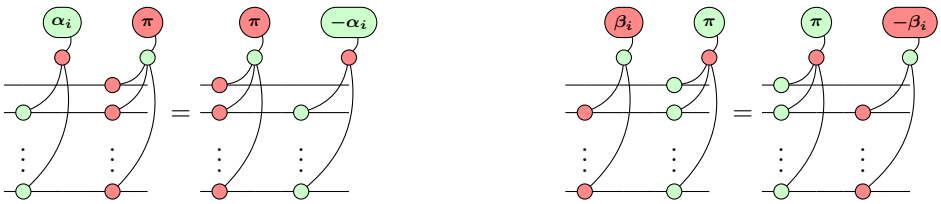


One can see that while the best option would be a ZX Polynomial describing identity, we found one with six phase gadgets. Hence it is necessary to create an optimization strategy. We developed such an optimization strategy based on the idea of peephole optimization, inspired by Nam et al. [2]. The general goal of the optimization strategy is to merge and remove as many phase gadgets as possible. Hence we need to consider the following three operations:

- Moving phase gadgets through the circuit
- Removing phase gadgets
- Merging phase gadgets

4.0.1 Moving Phase gadgets

Regarding the movement of phase gadgets, we can do this by consequently applying commutation rules outlined in section 3.1.1. Nevertheless, one can also show that if we have a phase gadget with phase π, we can move this through arbitrary other phase gadgets by updating the phase. Assume two phase gadgets of unequal color, which do not share an equal amount of legs. One of the gadgets has the phase π. Then the following two equations hold[8]:



(21)

4.0.2 Removing and merging Phase gadgets

We can remove a Phase gadget iff its phase corresponds to 0 or 2π. In this case, the Z or X Spider in the middle will be the identity matrix, and the phase gadget "collapses". We can merge two phase gadgets iff all of their legs are equal by the following rule derived by Cowtan et al. [9]:

(22)

4.0.3 Optimization Algorithm

Our optimization approach is based on the idea of peephole optimization as done by Nam et al. [2]. As a first step, we are moving each phase gadget through the ZX Polynomial by the rules outlined in section 4.0.1. We combine the two phase gadgets if we find another phase gadget with the same amount of legs. Otherwise, we will restore the position of the phase gadget we moved through the circuit. As a second step, we remove all phase gadgets, which have a phase of 0 or 2π. We then continue those two steps until convergence.

[8]Note that for reasons of conciseness, we omitted the proof. As an outline, one can decompose the π phase gadget, then move all CNOTs through the other phase gadget. Since both share an unequal amount of legs, there will exist a leg on the qubit with the π rotation, and the phase gets multiplied by -1 following commutation rules of Cowtan et al. [9]

5 Evaluations

5.1 Random Phase Gadget Synthesis

In order to verify our results, we compared our algorithm to the algorithm by Cowtan et al. [9] combined with the routing procedure from tket. In order to do this, random ZX Polynomials were created and synthesized using a "fully connected", a "line", and a "circle" architecture. For each architecture, we created ZX Polynomials with 3, 4, and 5 qubits with the number of phase gadgets in the range $[10, 30, 50, 70, 90, 110, 120, 130, 140, 150]$. We used a small number compared to the analysis of Griend et al. [6] due to the high computational cost of the blocking column extension of Arianne's synthesis algorithm. Hence, we also performed an analysis with the creation of 3, 5, and 7 qubits and the number of phase gadgets in the range $[100, 300, 500, 700, 900, 1000]$ for the remaining algorithms (A*, tket and split Arianne's synthesis algorithm). We determined the reduction in circuit depth, CNOTs, and single-qubit gates after optimization and used the ratio of these measures (in percentage) before and after optimization to compare the different algorithms. Overall, we found that split-Arianne's synth and the A* algorithm performed better for random phase gadgets on fully connected architectures than tket. However, this performance boost discontinues for sparser architectures like a line or a circle. Even worse: with an increasing number of qubits, they worsen the CNOT count.

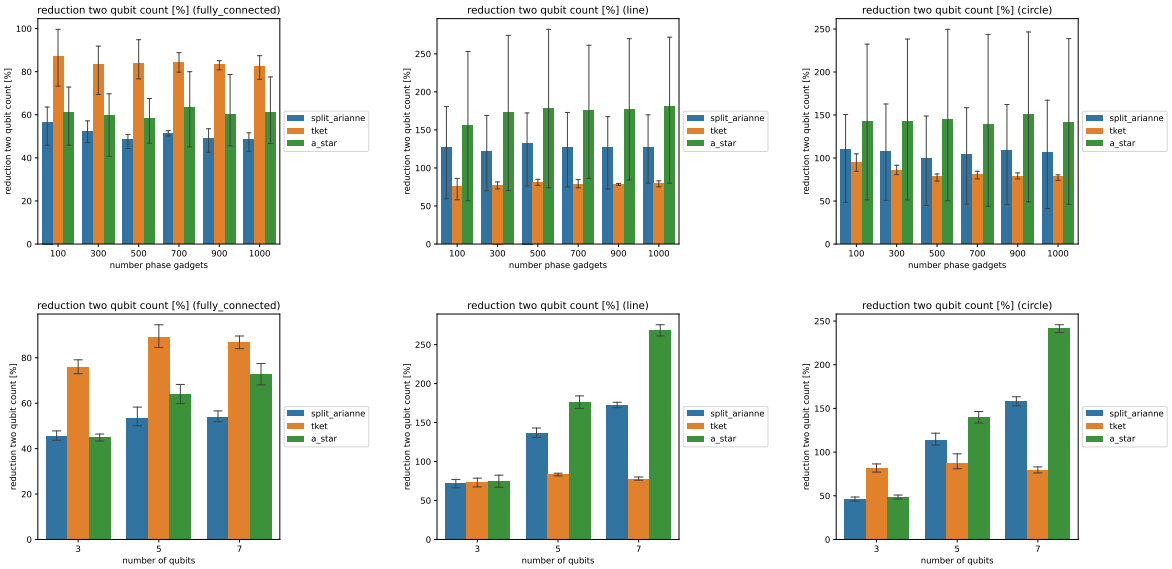


Figure 6: One can see an evaluation of ZX Polynomials with up to 1000 phase gadgets. One can observe the variation in the number of phase gadgets in the first row. The variation of qubits in the second.

5.2 Performance of Optimization

The performance of the optimization algorithm was evaluated as follows: Create a random Clifford+T circuit, optimize it with the algorithm described in section 4 and synthesize it with either A* or the split extension to Arianne's synthesis algorithm. We compared our results to pyzx, another optimization framework based on the ZX-Calculus [3]. We only evaluated the performance on a fully connected architecture since pyzx does not have an architecture-aware extension. Note that here we used the same metric described in the last section. In figure 8 one can see that while the depth and CNOT-count blew up due to the synthesis algorithms, the reduction in terms of one qubit-gates is comparable. Our optimization procedure is highly effective in reducing the number of phase gadgets (since each rotation corresponds to one phase gadget). Nevertheless, for larger circuits pyzx outperforms our approach.

5.3 Real world examples

We also evaluated the performance of the algorithms on some real-world examples (See table 1). By name, the circuits provided by Cowtan et al. [9]. We found a very similar picture. While we did not outperform tket, it is still clear that the algorithms combined with the optimization process can reduce the one-qubit and two-qubit count. This effect discontinues for sparser architectures.

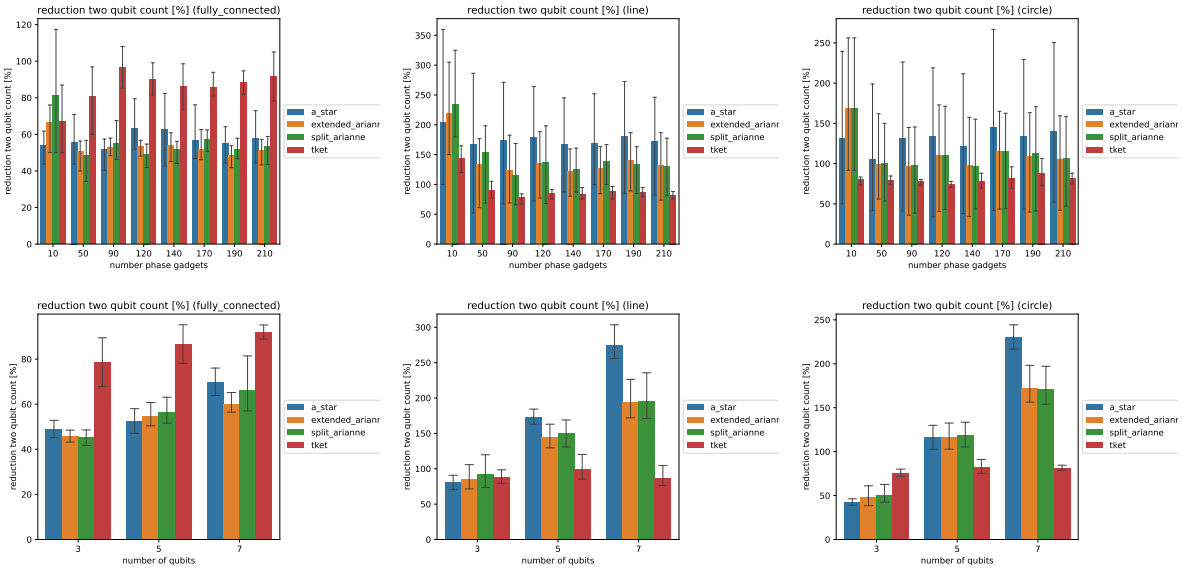


Figure 7: One can see an evaluation of ZX Polynomials with up to 140 phase gadgets. One can observe the variation in the number of phase gadgets in the first row. The variation of qubits in the second column

FULLY_CONNECTED	**tket**		**A***		**split-arianne**		**Default**	
Name	#CX	Depth	#CX	Depth	#CX	Depth	#CX	Depth
H2_P_sto3g	20	20	28	28	49	44	56	52
H2_BK_sto3g	20	20	28	26	50	42	50	42
H2_JW_sto3g	30	25	44	40	66	55	56	52
LINE	**tket**		**A***		**split arianne**		**Default**	
Name	#CX	Depth	#CX	Depth	#CX	Depth	#CX	Depth
H2_P_sto3g	20	20	60	60	60	60	38	36
H2_BK_sto3g	20	20	50	45	50	45	38	38
H2_JW_sto3g	30	25	144	129	115	98	56	52

Table 1: Results of testing against small sample circuits provided by Cowtan et al. [9]

6 Conclusion

Throughout this guided research, different methods for synthesizing ZX Polynomials were introduced. It showed that there exist ways for an efficient approximation and good extensions for the synthesis algorithm by Griend et al. [6]. We also showed that the optimization strategy on ZX Polynomials is both simple and effective in reducing the number of rotations in the circuit. Griend et al. [6] concluded in their work that *"...This shows that naive synthesis combined with clever routing is not competitive with architecture-aware synthesis methods"*. However, we cannot hold this statement based on our results and algorithms. In real-world examples, tket performed much better with their approach of Pauli-Simplification and routing afterwards. Also, pyzx tends to outperform the synthesis approach in terms of CNOT count and depth. Hence it is questionable if this statement can be generalized to Pauli gadgets. Nevertheless, the algorithm of Griend et al. [6] is quite effective if all phase gadgets in the ZX Polynomial commute. This can be the case for real-world quantum circuits (QAOA applied to MAX CUT, for instance. [11]). One can conclude from this work that the concept of ZX Polynomials offers an easy and effective way for optimization and synthesis. It could be exciting to see a combination of routing and synthesizing ZX Polynomials in the sense that during synthesis, architecture constraints are noted (as a heuristic as in A* or by the propagation of CNOTs in the split extension of Arianne's synthesis algorithm), but later the circuit is naively routed. Other future work aspects of this guided research would be to find an optimal permutation of qubits for a specific architecture and how this affects the question of performance of architecture-aware synthesis.

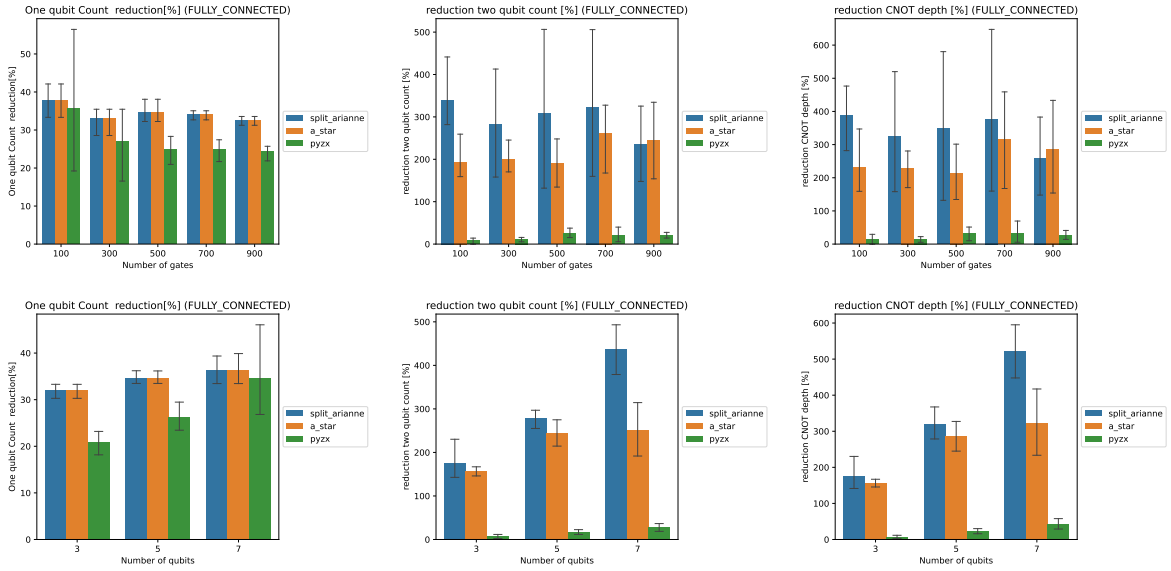


Figure 8: One can see an comparison of the number of one qubit gates, CNOT gates and depth of our optimization procedure compared to running pyzx on an arbitrary Circuit.

References

[1] Aleks Kissinger and Arianne Meijer van de Griend. Cnot circuit extraction for topologically-constrained quantum memories. 4 2019.

[2] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4, 5 2018.

[3] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. 2 2019.

[4] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, Oct 2014.

[5] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t—ket⟩: a retargetable compiler for nisq devices. *Quantum Science and Technology*, 6(1):014003, Nov 2020.

[6] Arianne Meijer van de Griend and Ross Duncan. Architecture-aware synthesis of phase polynomials for nisq devices. 4 2020.

[7] Aleks Kissinger and John van de Wetering. Simulating quantum circuits with zx-calculus reduced stabiliser decompositions. 9 2021.

[8] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes*. Cambridge University Press, 2017.

[9] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. Phase gadget synthesis for shallow circuits. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 318:213–228, 5 2020.

[10] Richie Yeung. Diagrammatic design and study of ansätze for quantum machine learning. 2020.

[11] Ritajit Majumdar, Dhiraj Madan, Debasmita Bhoumik, Dhinakaran Vinayagamurthy, Shesha Raghunathan, and Susmita Sur-Kolay. Optimizing ansatz design in qaoa for max-cut. 6 2021.