Qbit Layouts for Quantum Computation Using Non-Abelian Anyons

von
Martin Wosnitzka

Bachelorarbeit in Physik
vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen
im August 2015
angefertigt im
Institut für Quanteninformation
bei
Univ.-Prof. Dr. David DiVincenzo
Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 10.08.2015
Contents

1 Some Basics on Quantum Computation 1

2 Introduction 5

3 Stabilizer Code Formalism 7
   3.1 Stabilizer and Stabilized Subspace .......................... 7
   3.2 Error-Correction within the Stabilizer Code .................. 8

4 The Surface Code 10
   4.1 Defining the Code ......................................... 10
   4.2 Performing Measurement Circuits and Correcting Errors ...... 13
   4.3 Processing Information: Logical Operators and Braiding Holes 15

5 Anyons 19
   5.1 Anyon Theory ............................................. 19
       5.1.1 Fusing Anyons ....................................... 20
       5.1.2 Braiding Anyons ...................................... 22
   5.2 Fibonacci Anyons .......................................... 24

6 Fibonacci Levin-Wen Model 26
   6.1 Viewing the Levin-Wen model as a Stabilizer Code .......... 26
   6.2 Vertex Measurement Circuit .................................. 28
   6.3 F-move ..................................................... 28
   6.4 Two Plaquette Measurement Circuits ......................... 29

7 Optimal Linking of Qbit Layouts 33
   7.1 Posing the Problem ......................................... 33
   7.2 The Linear Optimization Approach ............................ 35
       7.2.1 The Linear Optimization Formulation .................. 35
       7.2.2 Solvability and Performance .......................... 35
   7.3 Example: Plaquette Reduction Layout ........................ 36
   7.4 Scaling a Unit Cell ......................................... 39
       7.4.1 The Unit Cell ......................................... 40
       7.4.2 Equivalent Connections .................................. 41
       7.4.3 Algorithm for Finding the Linking and Solution ...... 43

8 Layouts for the Surface Code 46
   8.1 Surface Code Unit Cell ...................................... 46
   8.2 Rotated Patch with reused Ancillary Qbits .................... 47

9 Acknowledgments 52

10 References 52
1 Some Basics on Quantum Computation

This thesis discusses different aspects of quantum computation - theoretical ones as well as
more practical ones. Before giving the introduction we want to state some basics about quan-
tum computation.

Qbits and Gates

While a classical computer operates on bits that can only have the values 0 and 1, a quantum
computer operates on Qbits. Qbits are two level quantum systems and can thus be associated
with any normed state $|\psi\rangle$ within a two dimensional vector space $V_Q$ with basis vectors $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \alpha, \beta \in \mathbb{C} \quad \text{and} \quad |\alpha|^2 + |\beta|^2 = 1$$

In an analogous manner $n$ Qbits can be associated with a state within the $2^n$ dimensional
vector space $\tilde{V} = \otimes_{i=1}^n V_{Q,i}$ where $V_{Q,i}$ is the two dimensional vector space associated with
Qbit $i$. Hence $\tilde{V}$ is simply the $n$-fold tensor product of the 1-Qbit vector spaces and its
computational basis are the $2^n$ orthogonal states that can be formed as a $n$-fold product of
the 1-Qbit basis states $|0\rangle$ and $|1\rangle$.

The operations on Qbits are carried out by gates. A gate is a linear operator that takes a
normed state into another normed state and is reversible. The set of operators $U$ that satisfy
this condition is the set of unitary operators.

A gate that acts on only one Qbit is called a 1-Qbit gate (and accordingly a gate that acts
on $n$ Qbits is called $n$-Qbit gate). Three important 1-Qbit gates are the $X$-, $Y$- and $Z$-gate
whose matrix representations in the $\{|0\rangle, |1\rangle\}$ basis are the Pauli matrices $\sigma_x, \sigma_y$ and $\sigma_z$:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Given this representation, $\{|0\rangle, |1\rangle\}$ is the basis of eigenvectors of $Z$. We will call this basis
the computational basis or $Z$-basis. Of course we could choose another basis to represent the
state of a Qbit, for example $|+\rangle$ and $|-\rangle$ where

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \quad |-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle).$$

The vectors $|+\rangle$ and $|-\rangle$ form a basis of eigenvectors of $X$.

We want to show an important 2-Qbit gate that will appear in the thesis. Two Qbits are
associated with any state $|\phi\rangle$ that has the following computational basis representation

$$|\phi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} \quad \alpha, \beta, \gamma, \delta \in \mathbb{C} \quad \text{and} \quad |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$$

The 2-Qbit gate we want to introduce is the CNOT (controlled-NOT) gate $C$. Its matrix representation in the computational basis is

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

This gate flips the state of the second Qbit from $|0\rangle$ to $|1\rangle$ or from $|1\rangle$ to $|0\rangle$ respectively if and only if the state of the first Qbit is $|1\rangle$. Otherwise the state remains unchanged. Here we read the Qbit from left to right so in the state $|01\rangle = |0\rangle \otimes |1\rangle$ we call the Qbit in state $|0\rangle$ the first one and the Qbit in state $|1\rangle$ the second one. With regard to its action we call the first one control Qbit and the second one target Qbit when considering the CNOT gate.

**Circuit Diagrams**

A sequence of gates acting on some Qbits is called a circuit and can be visualized by a circuit diagram. Some circuit diagrams are shown in figure 1 a) to c). On the left of a circuit diagram we note the Qbits that are acted on. If we know their initial state we can write it down. In this case the outcome of a circuit can be calculated explicitly and is often noted on the right side. The left and right side are connected by one line per Qbit. On these lines we note the gates that are applied to the Qbits. The diagram is read from left to right, so gates on the left are applied before gates on the right are applied. Let us go through the examples of figure 1 a) to c) one by one as this will give us the opportunity to introduce some more important basics.
In diagram a) there are two Qbits. One in state $|\psi_1\rangle$ and the second in $|\psi_2\rangle$. As we do not know what states $|\psi_1\rangle$ and $|\psi_2\rangle$ actually are (there is no basis representation given) we could as well just write 1 and 2 on the left side to symbolize that there are two Qbits called 1 and 2 that are acted on but we do not know their initial state. The diagram tells us that we first apply a $X$-gate to the first Qbit, then apply a $Z$-gate to the second Qbit and in the end a $Y$-gate to the first. We could potentially write $YX|\psi_1\rangle$ and $Z|\psi_2\rangle$ on the right side as these are the resulting states of the circuit. However, as we do not know $|\psi_1\rangle$ and $|\psi_2\rangle$ this is somehow trivial.

Diagram b) shows a CNOT gate acting on two Qbits in states $|1\rangle$ and $|0\rangle$. To see this we remember that a CNOT gate flips the target Qbit (which is essentially the same as applying the $X$-gate) if and only if the control Qbit is in state $|1\rangle$. The dot on the upper line means that the corresponding Qbit functions as a control Qbit for another gate and the vertical line connects the dot with the very gate. This time the initial state is explicitly known and thus we can calculate the outcome - which is $|1\rangle$ for both Qbits - and write it on the right side. Note that this notation allows us to represent arbitrary controlled gates. Suppose we have a 1-Qbit gate $W$. If we replace $X$ with $W$ in the circuit we have the representation of a controlled $W$-gate, that is $W$ is applied to the target Qbit if and only if the control Qbit is in state $|1\rangle$.

Diagram c) shows a circuit equation. Both of the shown circuits act in the same way on the Qbits even though their sequence of gates is not the same. On the left side we have a $X$-gate on Qbit 3 that is controlled by Qbit 1 and 2, so the $X$-gate is only applied to Qbit 3 if both Qbits 1 and 2 are in the state $|1\rangle$. This circuit is thus a 3-Qbit gate. A NOT (or $X$) gate that is controlled by $n - 1$ Qbits is called a $n$-Toffoli gate. The circuit on the right side makes use of five controlled 2-Qbit gates. One can check that this equation holds true by applying both circuits to the 8 basis states of the 3-Qbit system. Note however, that the equation is
essentially true because we have

\[ A^2 = X \quad \text{and} \quad AA^\dagger = I \]

for the 1-Qbit gate \( A \).

For a more general method and proof of such decompositions of 3 and higher Qbit gates into 2-Qbit gates see [1]. At this point we shall only note that the \( n \)-Toffoli gate can always be build up out of 1- and 2-Qbit gates. In fact using only 1- and 2-Qbit gates is enough to approximate any gate to arbitrary precision (see [2]). Moreover, constructing higher Qbit gates is technically very challenging. Therefore throughout the thesis we will always assume that any gate we make use of is actually made up of only 1- and 2-Qbit gates.

We conclude this chapter by giving the definition of universal quantum computation: A set of gates is said to be universal for quantum computation if any gate can be approximated to arbitrary precision by a sequence of these gates. This implies that 1- and 2-Qbit gates are universal for quantum computation.
2 Introduction

One problem of the practical realization of quantum computers is the liability of quantum systems to perturbations. The interaction of a quantum system with its environment often makes it impossible to preserve a certain state for some time, and recovery of corrupted states is impossible for many systems as well. However, if one wants to reliably store quantum information the preservation of a quantum state is required. Consequently, there is a great interest in finding systems that do allow to preserve a state fault tolerantly despite perturbations.

Stabilizer codes are one way of storing quantum information fault-tolerantly. Within such codes the occurrence of errors can be detected and corrected. The thesis will begin with a look at the stabilizer formalism which is the theoretical framework of stabilizer codes.

To fill this framework the following section discusses the surface code. The surface code is a stabilizer code. We will apply the theoretical terms of the stabilizer formalism to this example and see how the correction of errors is carried out within the surface code. With regard to the practical part of this thesis we will as well focus on the circuits that are required when realizing the surface code experimentally.

Before proceeding with the second theoretical main part we will briefly hint at how the stored quantum information can be processed within the surface code. It is a useful motivation for the next section.

This next section is about anyons. When exchanging anyonic particles they behave differently from bosons and fermions. Anyons can be used to encode Qbits and their exchange statistics allows to process quantum information by exchanging them. This way of processing is called topological quantum computation because the outcome of an exchange depends on its topology only.

Again we will begin with a small extract of general anyon theory and give an example afterwards. This example are so called Fibonacci anyons which are particularly interesting because they can be used to perform universal quantum computation.

The subsequent section introduces the Fibonacci Levin-Wen model. This model is a spin model whose ground state can be viewed as a stabilizer code. On top of that it can realize Fibonacci anyons and thus has the potential for universal quantum computation. We will define the model and afterwards focus on the required circuits again.

The concluding practical oriented part of the thesis is about the efficient implementation of quantum circuits, especially those we introduced for the surface code and Fibonacci Levin-Wen model. A certain circuit that one wishes to be performed by a layout of Qbits will require this layout of Qbits to have certain couplings of Qbits. As there are experimental limitations on how one Qbit can be coupled with other Qbits, it is in fact questionable whether some of the circuits we discuss in the earlier sections are actually practicable or not. Moreover, we are naturally interested in a material saving way of coupling Qbits.

We will consider layouts of a given number of Qbits and required couplings, as well as layouts
that can potentially be scaled to any size by replicating a unit cell of Qbits. For both of these cases we will formulate an algorithm that decides whether a circuit is practicable or not and - if existent - gives the optimal way of linking the Qbits. These two algorithms will be applied to some examples of the surface code and Levin-Wen model.
3 Stabilizer Code Formalism

We are interested in quantum codes that are tolerant to errors. Errors can either occur by interaction of the system with the environment or when processing a code, i.e. the gates we perform will in practice be defective. We are concerned about the first type of errors. So called stabilizer codes are a class of codes that allow for fault tolerant storage of quantum information. In this first chapter we will introduce the stabilizer formalism to prepare for the the surface code which is an important example of stabilizer codes and will be shown in the following chapter. We will do so by giving a selection of key definitions and propositions of [3].

3.1 Stabilizer and Stabilized Subspace

Consider a set of \( n \) Qbits and let \( V \) be the associated \( 2^n \) dimensional vector space. Suppose that there is an operator \( O \) and a state \( |\psi\rangle \in V \) such that \( O|\psi\rangle = |\psi\rangle \). We will then call \( |\psi\rangle \) stabilized by \( O \). The main idea of stabilizer codes is to choose a set of operators in a way that the subspace of stabilized states encodes a logical Qbit which is at least protected from “simple” errors. In order to see how this works and what exactly we mean by simple errors let us first introduce some further definitions and propositions.

Let \( G_n \) be the \( n \)-fold tensor product of the Pauli group.

\[
G_n = \{ \pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ \}^n
\]

where \( I \) is the 1-Qbit identity operator and \( X, Y \) and \( Z \) are the three 1-Qbit operators we introduced in the first chapter. Note that the elements of \( G_n \) are actually \( n \)-fold tensor products of gates. But as the matrix representation of these gates in the \{\( |0\rangle, |1\rangle \}\} basis form the Pauli group we call \( G_n \) the Pauli group. \( G_n \) is a group under matrix multiplication because \( G_1 \) is a group under matrix multiplication. Let \( S \) be a subgroup of \( G_n \) and \( V_S \subset V \) the subset of states that are stabilized by every operator in \( S \). The vector space \( V_S \) is a subspace as every linear combination of stabilized states is stabilized as well. We call \( S \) the stabilizer of \( V_S \).

As \( S \) is a group it can be written using generators: Let \( O_1, \ldots, O_m \in G_n \) be \( m \) arbitrary operators in \( G_n \). We set

\[
\langle O_1, \ldots, O_m \rangle = \{ O \in G_n \mid \exists k \in \mathbb{N} : O = \prod_{i=1}^{k} P_i, \quad P_i \in \{O_1, \ldots, O_m\} \text{ for } 1 \leq i \leq k \}.
\]

If \( S = \langle g_1, \ldots, g_r \rangle \) we call \( g_1, \ldots, g_r \) the generators of \( S \). Furthermore, we call the generators independent if \( \langle g_1, \ldots, g_{p-1}, g_{p+1}, \ldots, g_r \rangle \neq S \) for all \( 1 \leq p \leq r \). Note that \( \langle g_1, \ldots, g_{p-1}, g_{p+1}, \ldots, g_r \rangle = \langle g_1, \ldots, g_r \rangle \) implies \( g_p \in \langle g_1, \ldots, g_{p-1}, g_{p+1}, \ldots, g_r \rangle \).

Using the generators representation there is a very useful proposition.
Proposition (Dimension of Stabilized Subspaces). Let $S = \langle g_1, \ldots, g_{n-k} \rangle$ and let the generators be independent. Suppose further that all the generators commute and that $-I_n \notin S$. Then $V_S$ has dimension $2^k$.

Although we do not want to proof this proposition (for the proof see [3]) we note that it is easy to see why the additional requirements of commuting generators and $-I_n \notin S$ are needed: If $O_1, O_2 \in S$ and $O_1 O_2 \neq O_2 O_1$ then $O_1 O_2 = -O_2 O_1$ must follow as all the elements of the Pauli group either commute or anti-commute. But this implies $O_2 O_1 |\psi\rangle = |\psi\rangle = -O_1 O_2 |\psi\rangle = -|\psi\rangle$ for $|\psi\rangle \in V_S$. The same follows from $-I \in S$ because $|\psi\rangle = -I |\psi\rangle = -|\psi\rangle$ as $|\psi\rangle$ is stabilized by the elements of $S$. In both cases $V_S$ would be trivial. Note further that $g^2 = I$ is implied for all $g \in S$ by $-I \notin S$ because $O^2 = \pm I$ for all $O \in G_n$ and $S$ is a group.

3.2 Error-Correction within the Stabilizer Code

The proposition we just stated tells us that we can use a stabilizer $S$ to create a subspace $V_S$ and that we can calculate the dimension of this subspace $V_S$. For example we could find $n-1$ independent and commuting generators and the corresponding subspace $V_S$ would have dimension 2. We could then use this subspace to store the quantum information of one logical Qbit. At first it seems not very useful to have $n$ physical Qbits encoding one logical Qbit, when we could potentially use every single one of the $n$ Qbits itself to store the same information. But single physical Qbits are very liable to errors and in order to correct these errors one would in general have to measure the Qbit. But measuring the Qbit leads to a collapse of the wave function and thus to a loss of the information stored. In contrast to single physical Qbits there is a set of errors that can be corrected within the surface code:

Proposition (Correctable Errors for Stabilizer Codes). Let $S$ be the stabilizer of $V_S$ and let $Z(S) = \{O \in G_n \mid \forall g \in S : [g, E] = 0\}$ be the centralizer of $S$. Suppose further there is a set of operators $\{E_1, \ldots, E_m\} \subset G_n$ such that $E_i^\dagger E_j \notin Z(S) \setminus S$ for all $1 \leq j, k \leq m$. Then $E_i$ is correctable for all $1 \leq i \leq m$ when acting upon the subspace $V_S$.

We do not want to proof this proposition in detail (again see [3] for the proof). However, we can capture the basic idea of this proposition and give a less formal argument:

Suppose the system is in a state $|\psi\rangle \in V_S$. We say that an error occurs if an operator $E_j \in G_n$ unintentionally acts upon this state (for example by means of interaction of the system with the environment). We constantly measure the generators of $S$. They have eigenvalues $\pm 1$. The elements of the Pauli group all commute or anti-commute so we either have $g E_j |\psi\rangle = E_j |\psi\rangle$ or $g E_j |\psi\rangle = -E_j |\psi\rangle$. Hence, $E_j |\psi\rangle$ is always an eigenstate of $g$, the result of a measurement is never uncertain and the measurement does not change the state. We call the results of these measurements error syndrome.

If an error occurs and we measure the generators afterwards there are two possible cases. Either one or more of the measurement results yields $-1$ or all of the results yield $1$. If one or more of them yield $-1$ we have left the subspace $V_S$. We then choose any error $E_k$ that yields the same error syndrome and apply $E_k^\dagger$, it does not matter whether $E_j$ is the unique error.
that yields the syndrome we measured and thus $E_j = E_k$ or not, having the same syndrome is sufficient for a correction of the error as we will show now.

Let $g \in S$ be an arbitrary generator of $S$. We have $E_k \in G_n$ and therefore one can check that $E_k^\dagger = \pm E_k$. Moreover, $E_j$ and $E_k$ have the same error syndrome and thus we have $[g, E_k] = [g, E_j] = 0$ or $\{g, E_k\} = \{g, E_j\} = 0$. Consequently we obtain:

$$[g, E_k^\dagger E_j] = 0 \Rightarrow E_k^\dagger E_j \in S$$

The last implication follows from the assumption that $E_k^\dagger E_j \notin Z(S) \setminus S$. So $E_k^\dagger E_j |\psi\rangle = |\psi\rangle$ and the error has been corrected. This is the central idea of the proposition above.

If all of the measurement results of the error syndrome yield 1 there are two possible cases. If $E_j \in S$ we do not have to worry as $E$ stabilizes the current state anyway. But if $E_j \in Z(S) \setminus S$ we have $E_j |\psi\rangle \neq |\psi\rangle$ but still $E_j |\psi\rangle$ is an element of $V_S$. So the state has been corrupted but we have no way of detecting that an error occurred or which error occurred.

To sum up this result we can say that an error $E_j$ can be corrected within a stabilizer code unless there is another error $E_k$ that yields the same error syndrome and for which $E_k^\dagger E_j$ is an element of $Z(S) \setminus S$ or $E_j$ is an element of $Z(S) \setminus S$ itself. This is why storing quantum information using a stabilizer code does make sense and is in especially more favorable than storing the information in single physical Qbits.

Regarding the last proposition, we introduce the weight of an operator and the distance of a code. The weight of an operator $O \in G_n$ is the number of factors in the tensor product that are not identity. The distance of a stabilizer code is the minimum weight of an operator in $Z(S) \setminus S$. Given these definitions, we can state that a stabilizer code with distance $d$ can correct all errors with weight smaller or equal to $(d - 1)/2$. In this sense a stabilizer code is tolerant against “small” errors.
4 The Surface Code

The surface code is an example of stabilizer codes. We want to introduce it to give a practical application of the formalism we introduced in the previous chapter and because it has experimental relevance.

4.1 Defining the Code

The surface code is defined on a square lattice with a physical Qbit on every edge of the lattice as shown in figure 2. Suppose that every side has a length of $b$ edges ($b \in \mathbb{N}$). The total number of Qbits is $b^2 + (b - 1)^2$. We call the boundary on the left and right smooth and the boundary on the top and bottom rough.

The generators of the stabilizer $S$ are also shown in figure 2. For every star $s$ of the lattice we have a star generator $A_s$ that acts with a $X$-gate on all four or three (at the smooth boundary) Qbits that are associated with the edges that form this star. The “stars” at the top and bottom boundary that only have one edge are not part of the stabilizer. For every plaquette $p$ of the lattice we have a plaquette generator $B_p$ that acts with a $Z$-gate on all the four or three (at the rough boundary) Qbits that are associated with the edges forming this plaquette. There are $b(b - 1)$ plaquette operators and $b(b - 1)$ vertex operators. So all in all we have $b^2 + (b - 1)^2 - 1$ generators.

![Figure 2: The surface code. The Qbits (blue dots) are placed on the edges of the square lattice. The operator $A_s$ acts with $X$-gates on the four Qbits of the corresponding star and the operator $B_p$ acts with $Z$-gates on the four Qbits of the corresponding plaquette.](image)

We want to make use of the proposition about the dimension of stabilized subspaces to show that $S$ is actually a stabilizer that stabilizes a two dimensional subspace. All of the generators
commute. To see this we only have to check generators $A_s$ and $B_p$ where both operators act on common Qbits. But as a plaquette and vertex have either zero or exactly two common edges $[A_s, B_p] = 0$ follows from $\{X, Z\} = 0$.

Furthermore, we have $-I \notin S$. To proof this we let $\hat{g} = \prod_{i=1}^{m} g_i$ with $g_i \in S$ be an arbitrary product of generators. Then every generator is either a star operator or a plaquette operator. Suppose there are $m_1$ star operators and $m_2 = m - m_1$ plaquette operators in this product. As all generators commute we can arrange the order of the generators in a way that we first apply all plaquette generators and afterwards all star operators.

$$\prod_{i=1}^{m} g_i = \prod_{i=1}^{m_1} A_{S,i} \prod_{j=1}^{m_2} B_{p,j}$$

Let $X_k \in G_n$ denote the operator in $G_n$ that acts on Qbit $k$ with $X$ and with identity on all other Qbits, and accordingly for $Z_l$. Then we can rewrite the star operator as $A_S = X_{(S,1)}X_{(S,2)}X_{(S,3)}X_{(S,4)}$ where $(S,1), (S,2), (S,3)$ and $(S, 4)$ are the four Qbits that form the star $S$. The same goes for plaquette operators and operators that only act on three Qbits. As all the operators $X_k$ and $Z_l$ except for the case $k = l$ commute, we can arrange our product in the following way:

$$\prod_{i=1}^{m_1} A_{S,i} \prod_{j=1}^{m_2} B_{p,j} = \prod_{i=1}^{m_1} X_{(S,i1)}X_{(S,i2)}X_{(S,i3)}X_{(S,i4)} \prod_{j=1}^{m_2} Z_{(P,j1)}Z_{(P,j2)}Z_{(P,j3)}Z_{(P,j4)}$$

$$= X_1^{m_{x_1}}Z_1^{m_{z_1}}X_2^{m_{x_2}}Z_2^{m_{z_2}}...X_n^{m_{x_n}}Z_n^{m_{z_n}}$$

where $m_{x_1}$ and $m_{z_1}$ denote the total number of $X_i$ and $Z_i$ operators that occur in our generators. We have

$$X_k^{m_{x_k}}Z_k^{m_{z_k}} = \begin{cases} I, & \text{if } m_{x_k} \text{ and } m_{z_k} \text{ even} \\ Z_k, & \text{if } m_{x_k} \text{ even, and } m_{z_k} \text{ odd} \\ X_k, & \text{if } m_{x_k} \text{ odd, and } m_{z_k} \text{ even} \\ -iY_k, & \text{if } m_{x_k} \text{ and } m_{z_k} \text{ odd} \end{cases}$$

and thus our arbitrary chosen product can never yield $-I$.

The generators are independent too. The fact that a plaquette operator can not be generated by a set of some other plaquette operators is based on the existence of the smooth boundary. The same is true for vertex operators and the rough boundary. Consequently on an all smooth boundary surface code there would be one dependent star operator and on an all rough boundary surface code there would be one dependent plaquette operator.

All in all the requirements of the proposition are given and the stabilized subspace $V_S$ has dimension 2 and can thus store the information of one logical Qbit.

Next we want to know the distance of the surface code. To find it we think about which operators commute with the generators (these operators form $Z(S)$). As a global phase does not change the physical properties of the wave function the elements of $Z(S)$ are only defined up to a multiplicative factor. So it is sufficient to consider only errors that act upon Qbits with identity, $X$, $Y$ or $Z$. Given an operator $E \in G_n$. $E$ is a tensor product of $n$ Pauli matrices.
We have $\sigma_x \sigma_z = -i \sigma_y$. So we can write $E$ as $E = c (e_1 \otimes e_2 \otimes \ldots \otimes e_n)$ where $c \in \{\pm 1, \pm i\}$ and $e_i \in \{I_i, X_i, Z_i, X_iZ_i\}$ is an operator that acts only within the Hilbert space of Qbit $i$. As operators that act within different Hilbert spaces commute we can write $E = c E_x E_z$ where $E_x$ only contains identity and $X$ operators and $E_z$ only contains identity and $Z$ operators.

Making use of this notation let us now look for the operators $E$ that commute with the generators. The generators $B_p$ commute with every operator which either acts only with $Z$ on Qbits or for which every plaquette contains either 0, 2 or 4 Qbits that are acted on with $X$ because $X$ anti-commutes with $Z$. It follows that $B_p$ always commutes with $E_z$. If we ignore the boundary for a moment $B_p$ commutes with $E_x$ if the Qbits that are acted on with $X$ by $E_x$ form closed loops as shown in figure 3 a). But these closed loops of $X$ operators can be constructed by a product of $A_s$ operators because the $A_s$ operators are in a sense the smallest possible closed $X$ loops and multiplying neighboring $A_s$ operators will enlarge these closed loops. If we take the boundary into account now, there is a different kind of $E_x$ operators that commute with $B_p$ but can in fact not be constructed by a product of $A_s$. Those are operators $E_x$ for which the $X$ operators form a string that goes from one smooth boundary to the other - see figure 3 b). From an intuitive topological point of view these operators can not be constructed by a product of $A_s$ operators because you can “open” and “close” $X$ loops at the smooth boundary but you can not move the string along the rough boundary. So a string of $X$ operators that starts at a smooth boundary and ends at the same smooth boundary can be pulled together to form a closed loops and thus be constructed by $A_s$ operators. But if it ends at the opposite smooth boundary you can not pull it together to form a closed loop because you would have to pass the rough boundary to do so. For a more profound argument on the topological properties of the generators see [4].
Accordingly the operators $A_s$ commute with every operator which either acts only with $X$ on Qbits or for which every star contains either 0, 2 or 4 Qbits that are acted on with $Z$. They always commute with $E_x$ and if the $Z$ operators form closed loops they commute with $E_z$ (see figure 3 a) as well. Furthermore, $E_z$ can be constructed by a product of $B_p$ operators if the $Z$ operators form closed loops. A string of $Z$ operators that starts at a rough boundary and ends at the opposing boundary commutes with $A_s$ but can not be constructed by $B_p$ generators (see figure 3 b).

All in all we have that $Z(S)$ is the set of operators $E \in G_n$ that have a decomposition $E = E_x E_z$ such that: $X$ operators form closed loops or strings that start at a smooth boundary and end at a smooth boundary and $Z$ operators form closed loops or strings that start at a rough boundary and end at a rough boundary. The set $Z(S) \setminus S$ is the subset of operators that include at least one $X$ string from a smooth boundary to the opposing smooth boundary or one $Z$ string from a rough boundary to the opposing rough boundary. Therefore the distance of the surface code is the length of its sides $b$ and by enlarging the square lattice we can enlarge the distance.

### 4.2 Performing Measurement Circuits and Correcting Errors

We need to constantly measure the star and plaquette operators to look for errors. We do this by performing measurement gates. The gates are shown in figure 4. Each of them uses one ancillary Qbit. As opposed to the Qbits of the surface code - which we will sometimes call data Qbit to distinguish them clearly from ancillary Qbits - we can measure ancillary Qbits without destroying the quantum information stored in the code. Two Qbits that are part of
a 2-Qbit gate need to be coupled with each other in some way. Such coupling can be realized by a so-called transmission line resonator (TLR). TLRs are essentially waveguides that allow directed propagation of electro-magnetic waves. When two Qbits are positioned inside a TLR, then the exchange of photons (virtual or not) realize the desired Qbit-Qbit interaction.

![Diagram of a measurement circuit for the star operator $A_s$.](image)

**Figure 4:** a) Measurement circuit for the star operator $A_s$. The symbol on the right side of the bottom line denotes a $X$-measurement of the ancillary Qbit. Representing the CNOT-gate in the $\{\vert + \rangle, \vert - \rangle\}$ basis one can check that the outcome of the measurement is 1 if the eigenvalue of $A_s$ is 1 and $-1$ if the eigenvalue of $A_s$ is $-1$. b) Measurement circuit for the plaquette operator $B_p$. The symbol on the right side of the bottom line denotes a $Z$-measurement of the ancillary Qbit. Representing the CNOT-gate in the $\{\vert 0 \rangle, \vert 1 \rangle\}$ basis one can check that the outcome of the measurement is 1 if the eigenvalue of $B_p$ is 1 and $-1$ if the eigenvalue of $B_p$ is $-1$.

So we note for later purposes that in the surface code for every measurement circuit the four (or three at the boundary) data Qbits that form the corresponding star or plaquette need to be coupled to the ancillary Qbit of this circuit, but they do not need to be coupled to each other.

We remember that the outcome of all generators $A_s$ and $B_p$ together is called the error syndrome. Let us take a brief look at how to correct an error once we have measured its error syndrome. To do so we focus on $X$ errors only as this is sufficient to understand how error correction is done in general and what problems can occur. A single $X$ error on one Qbit will result in an error syndrome where two neighboring plaquette measurements yield $-1$ instead of 1. We can correct it by simply applying the same $X$ again (and we know to which Qbit we need to apply the operator because there is only one Qbit that is adjacent to both plaquettes.
Figure 5: The $X$-strings in a) and b) yield the same error syndrome. While both errors can be corrected by any homotopy equivalent string, one can not use the error string of a) to correct the error string of b) and vice versa.

- see figure 5 a). Similarly a string of $X$ errors that does not end at the boundary will yield an error syndrome that has $-1$ for the two plaquettes that are located at the endpoints of the string (see figure 5 a) ). This time we can correct the error by applying any string of $X$ operators that connects these two plaquettes - that is any homotopy equivalent string. This works because both these strings - the one error string that actually occurred and the one we applied to correct the error - together form a closed loop of $X$ operators and as we have seen above those closed loops are elements of $S$ and thus do not change the state. However, there is another class of errors that yield the same syndrome and that can not be corrected by such a string. That is the class of all errors which have two $X$ strings that each start at the boundary and end at one of the two plaquettes (see figure 5 b) ) - and of course this might as well happen for error syndromes that have two neighboring plaquette measurements yielding $-1$. We can correct these kinds of errors too (by applying operators that form strings from the boundary to the plaquettes too). But given the error syndrome we do not know which error actually occurred and applying the wrong operator will result in a corruption of the stabilized state. The general solution to this complication is measuring at a sufficient high frequency so that long strings of errors become unlikely to occur. We can then (with a sufficiently high probability) be sure that an error that yields $-1$ for the syndrome measurement of plaquettes that are close to each other is a $Y$ string that connects these two plaquettes and an error that yields $-1$ for a single plaquette close to the boundary is a $X$ string from the boundary to this plaquette.

4.3 Processing Information: Logical Operators and Braiding Holes

Until now we have only talked about storing information. But of course we would also like to process this information. In the next chapter we will have a look at anyons and how they enable us to perform topological quantum computation which is one way of processing the information stored. The surface code allows for topological quantum computation via anyons but there are other possibilities to process quantum information within the surface code as...
Following [3] we want to first hint at one way that is closely related to the stabilizer formalism and afterwards show how topological quantum computation can in principle be realized within the surface code.

Suppose we have a stabilizer \( S = \langle g_1, \ldots, g_{n-k} \rangle \) that stabilizes a subspace \( V_S \) of dimension \( 2^k \). We can expand the set of generators by adding \( k \) operators \( \bar{Z}_1, \ldots, \bar{Z}_k \in G_n \) such that \( g_1, \ldots, g_{n-k}, \bar{Z}_1, \ldots, \bar{Z}_k \) are independent and commute with each other (up to a phase factor they stabilize a unique state). Furthermore, we can find \( \bar{X}_1, \ldots, \bar{X}_k \) such that \( \bar{X}_j \) anti-commutes with \( \bar{Z}_j \) but commutes with all other \( \bar{Z}_i, i \in \{1, \ldots, k\} \setminus \{j\} \), and with all generators \( g_1, \ldots, g_{n-k} \). Note that this implies \( \bar{X}_j, \ldots, \bar{X}_k, \bar{Z}_1, \ldots, \bar{Z}_k \in Z(S) \setminus S \). We can then define a basis \( |x_1, \ldots, x_k\rangle_L \), \( x_i \in \{0, 1\} \), of \( V_S \) where \( |x_1, \ldots, x_k\rangle_L \) is the unique state that is stabilized by \( \langle g_1, \ldots, g_{n-k}, (-1)^{x_1} \bar{Z}_1, \ldots, (-1)^{x_k} \bar{Z}_k \rangle \). The operators satisfy

\[
\bar{Z}_j |x_1, \ldots, x_k\rangle_L = (-1)^{x_j} |x_1, \ldots, x_k\rangle_L \\
\bar{X}_j |x_1, \ldots, x_k\rangle_L = \begin{cases} |x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_k\rangle_L & \text{if } x_j = 0 \\ |x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_k\rangle_L & \text{if } x_j = 1 \end{cases}
\]

We call \( |x_1, \ldots, x_k\rangle_L \) the logical computation basis and \( \bar{Z}_j \) and \( \bar{X}_j \) the logical \( Z \) and \( X \) operators.

Considering the surface code, the operators \( \bar{Z} \) and \( \bar{X} \) (here we have \( k = 1 \) ) are the operators in \( Z(S) \setminus S \) we discussed earlier when we determined the distance of the code. The operator \( \bar{Z} \) is any \( Z \) string going from one rough boundary to the other and \( \bar{X} \) is any \( X \) string going from one smooth boundary to the other. See also figure 3 b).

The logical \( \bar{X} \) and \( \bar{Z} \) operators for an encoded logical Qbit can be used to perform quantum gates. One example for this is the realization of a CNOT gate via so called lattice surgery with the surface code (see [4]). This realization of a CNOT gate makes use of two Qbits encoded separately in two different surface code sheets and their respective logical \( \bar{X} \) and \( \bar{Z} \) operators.

There is another way of performing a CNOT gate within the surface code that makes use of so called topological Qbits and braiding. Considering this method is a good start into the next chapter. So we want to show it briefly.

The main idea is to create “holes” in the stabilizer. This means we remove a generator from the set \( S \). Suppose for example we have a square lattice that has smooth boundaries on three of its sides and a rough boundary on the fourth side. The corresponding surface code initially encodes no logical Qbit because the dimension of the stabilized subspace is zero as the number of generators is equal to the number of Qbits (for this to hold true we have to add the two star generators at the corners where the smooth boundaries meet - these two act on two Qbits only). Now we remove a plaquette operator from \( S \) and call this plaquette a smooth hole. The new stabilized subspace \( V_{S,s} \) has dimension 2 and thus encodes one logical Qbit. The hole and its logical operators are shown in figure 6 a). The logical \( \bar{Z}_s \) is a closed loop of
Z operators around the smooth hole and the logical $X_s$ operator is a string of $X$ operators that connects the hole with one of the smooth boundaries. We denote the corresponding basis states of the logical (smooth) Qbit by $|0\rangle_s$ and $|1\rangle_s$. In the same way we can create a “rough” hole and a new stabilized subspace $V_{S,p}$ by removing a star operator. The logical $Z_r$ operator is a $Z$ string from the rough hole to the rough boundary and the logical $X_r$ operator is a $X$ loop around the rough hole (see figure 6 a)). We denote the corresponding basis states of the logical (rough) Qbit by $|0\rangle_r$ and $|1\rangle_r$.

Creating a smooth and a rough hole at the same time will yield a stabilized subspace $V_{S'} = V_{S,s} \otimes V_{S,r}$ of dimension 4 that encodes two Qbits that correspond to the two holes. We choose the basis states to be the tensor products of $\{ |0\rangle_s, |1\rangle_s \}$ and $\{ |0\rangle_r, |1\rangle_r \}$.

Now imagine we move the smooth hole around the rough hole (see figure 6 b)). Afterwards the $X_s$ string goes from the smooth boundary around the rough hole to the smooth hole and the $Z_r$ string goes from the rough boundary around the smooth hole to the rough hole. This means the move performs the transformations $X_s \rightarrow X_s \otimes X_r$ because the additional $X$ loop around the rough hole is the $X_r$ operator. The $Z_r$ operator is transformed accordingly: $Z_r \rightarrow Z_s \otimes Z_r$. The other two logical operators $Z_s$ and $X_r$ remain the same. These transformations are the exact transformations a CNOT enacts on two Qbits if we take the smooth Qbit to be the control Qbit and the rough Qbit to be the target Qbit.

Figure 6: a) The black square and star denote a plaquette and star operator respectively that we remove from the stabilizer to create holes. Both holes encode a logical Qbit. Their logical $X$ operators are drawn as red strings and their logical $Z$ operator as blue strings. b) After moving the smooth hole around the rough hole the logical $X$ operator of the smooth hole will make an additional loop around the rough hole (green loop in the figure). Accordingly the logical $Z$-operator of the rough hole will make an additional loop around the smooth hole (not drawn in the figure here).

So we can realize the CNOT gate by simply moving (braiding) holes. Braiding of holes can practically be done by changing the choice of generators we measure. Again [4] gives a more detailed argument and takes a slightly different approach (starting with an all smooth
boundary lattice and creating two smooth and rough holes each that together encode one logical Qbit). Using braiding of holes to process quantum information is called topological quantum computation because the outcome of a braid depends only on its topological properties. Although we can perform a CNOT gate within the surface code when braiding holes, topological quantum computation is not universal for the surface code. However, there are codes that allow for universal topological quantum computation. We will introduce one of them in the next but one chapter. But before we have a look at anyons.
5 Anyons

In this chapter we want to introduce anyons. While anyons are a very interesting subject in their own right and do not only occur in relation to quantum computation, it turns out that they provide a possibility to process quantum information. The surface code as well as the Levin-Wen model give rise to anyons (“holes” in a surface code can actually be viewed as anyons). Within the latter they even allow for universal quantum computation.

We will present a general theory for anyons that includes the two central properties of anyons: Fusing and braiding. After that we will give the example of Fibonacci anyons. The chapter - text and figures - closely follows [5].

5.1 Anyon Theory

While the anyon model itself is a mathematical formalism we want to give a short motivation for it. Although there is of course a more sophisticated reasoning behind the introduction of anyon theory, we only want to consider the following idea:

It is known from quantum mechanics that exchanging two bosons in four dimensional space-time will not change the wave function of these two particles, while exchanging two fermions will change the algebraic sign of the wave function. Here it does not matter how we exchange the particles, i.e. which path they take in space to reach the position of the other particle. The same holds true when we exchange more than two particles. From a topological point of view one can justify this by noting that the world line trajectories of two exchanges of the particles are homotopy equivalent.

Now suppose that we do not exchange the particles in four dimensional space-time but in three dimensional space-time (two space dimensions and one time dimension). In this case the way of exchanging does matter, because the world line trajectories of two exchanges might not be homotopy equivalent. For example you can not continuously deform the trajectory of a clockwise exchange into the trajectory of a counter clockwise exchange. In fact the trajectories might be arbitrary braided. Consequently, in two space dimensions there are particles that are neither bosons nor fermions - they are anyons. Exchanging and braiding them can change their wave function. But not only by multiplying with $-1$. In principle the anyonic wave function can obtain any phase factor, or the wave function might even change into a state that is not linearly dependent of the original state - all by simply braiding the anyons.

Apart from these braiding statistics we want to allow anyons to fuse and split as well. Combining these two ideas gives rise to the anyon model that we will introduce now.
5.1.1 Fusing Anyons

Throughout the chapter we note $A$ to be a set of anyonic particles. The particles of this set can fuse with each other and fusion of particles $a, b \in A$ is noted by

$$a \times b = \sum_{c \in A} N_{ab}^c c$$

The operation $a \times b$ is commutative and associative. However, it is not closed as the sum of two particles $a + b$ is not an element of $A$. The sum on the right-hand-side of the equation merely means that the two particles $a$ and $b$ can fuse to form different particles, i.e. to every particle for which $N_{ab}^c \neq 0$. The non-negative integer $N_{ab}^c$ tells us in how many ways the two particles can fuse.

The set $A$ always contains the neutral (or “vacuum”) particle 1. For this particle we have $N_{1a}^c = \delta_{ac}$ so fusing with vacuum will always preserve the particle. Moreover, for every particle $a \in A$ there is an anti particle $\bar{a} \in A$ for which we have $\bar{\bar{a}} = a$ and $N_{1a}^e = \delta_{\bar{a}b}$, so a particle can only become vacuum when fused with its antiparticle and the way of fusing is unique as well. For the vacuum 1 we have $\bar{1} = 1$.

At this point we can make a fundamental distinction. If $\sum_{c \in A} N_{ab}^c = 1$ for every particle $b$, then the particle $a$ is called Abelian. An anyon model that contains only Abelian particles is called Abelian model. If there are particles $a$ and $b$ such that $\sum_{c \in A} N_{ab}^c > 1$ the model is called non-Abelian.

The ket-vectors representing the wave function of a system of anyons are elements of so called splitting spaces, while the bra-vectors are elements of so called fusion spaces. To every fusion product corresponds a fusion space $V_{c}^{ab}$ and a splitting space $V_{ab}^{c}$ which is dual to the fusion space. The dimension of these two spaces is $N_{ab}^c$ and thus depends on the number of different ways the two particles can fuse. The notation of a basis ket-vector in the splitting space will be $|a, b; c, \mu \rangle \in V_{ab}^{c}$ where $\mu = 1, .., N_{ab}^c$ (analogously the basis bra-vectors are $\langle a, b; c, \mu | \in V_c^{ab}$). Using these splitting and fusion spaces we can construct spaces that correspond to the splitting and fusion of more than two particles. For example the space of a particle $d$ that splits to three particles $a, b$ and $c$ is isomorphic to both of the following decompositions

$$V_d^{abc} \cong \bigoplus_{e \in A} \left( V_e^{ab} \otimes V_d^{ec} \right) \cong \bigoplus_{e \in A} \left( V_d^{ae} \otimes V_c^{be} \right)$$

where $\bigoplus$ denotes the direct sum of vector subspaces. Therefore the dimension of $V_d^{abc}$ is $\sum_{e \in A} N_e^{ac} N_e^{bd}$. The same works accordingly for a space that corresponds to the fusion of three particles to one. In general one can construct a space of $n$ particles $b_1, .., b_n$ that split into $m$ particles $a_1, .., a_m$ in an iterative manner:

$$V_{b_1, \ldots, b_n}^{a_1, \ldots, a_m} = \bigoplus_{e \in A} V_{a_1, \ldots, a_m, e}^{b_1, \ldots, b_{n-1}, b_n} = \bigoplus_{e \in A} V_{e}^{a_1, \ldots, a_m, e} \otimes V_{b_1, \ldots, b_m}^{a_1, \ldots, a_m, e}$$

The elements of the splitting space can graphically be represented by splitting branches as
shown in figure 7 a). They can be stacked to form elements of spaces that correspond to a particle splitting into more than two particles as show in figure 7 b). Of course there is a graphical representation of elements of the fusion space as well. But we do not need these diagrams here.

![Figure 7: a) Graphical representation of an element of the splitting space $V^{ab}_{c}$. b) Stacking three diagrams of the type in a) yields an element of $V^{abcd}_{e}$. Note that we left out the fourth index of the splitting space basis vectors in this notation. Thus the state denoted is only unique if the splitting spaces all have dimension 1.](image)

Now let us focus on the two different ways of decomposing $V^{abc}_{d}$. We want the space $V^{abc}_{d}$ to be unique regardless of which way we use to decompose it. Physically this means that the state of the system will be within the same subspace no matter if particle $d$ first splits into $c$ and another particle that in turn splits to $a$ and $b$ or first splits into $a$ and another particle that in turn splits to $b$ and $c$. For this associative consistency to be fulfilled the two direct sums of vector spaces must be isomorphic, as we already stated above. Consequently there must be an isomorphism between the two. This isomorphism is given by the $F$-symbol that performs the basis transformation and is thus unitary:

$$|a, b; e, \alpha ; f, \mu , \nu \rangle = \sum_{f, \mu , \nu} \left[ F^{abc}_{d} \right]_{(e, \alpha , \beta )(f, \mu , \nu)} |b, c; f, \mu \rangle |a, f; d, \nu \rangle$$

This change of basis is often referred to as the $F$-move. Physically fusion and splitting that includes the vacuum does not change the state. Formally this is imposed by requiring $F^{abc}_{d}$ to be 1 if $a$, $b$ or $c$ is the vacuum.

Unitarity and the requirement of $F^{abc}_{d}$ to be 1 if the vacuum is involved are not sufficient for the $F$-symbol, it needs to have another important property: Suppose we start with a certain basis decomposition and we want to transform it into another decomposition of the same space. In general there might be different sequences of $F$-moves that end up in the same decomposition, as graphically shown in figure 8. So we must ensure that two different sequences of $F$-moves that start and end with the same decomposition do in fact lead to
the same result. It turns out that this consistency can be obtained for arbitrary numbers of
anyons by imposing one additional equation called the Pentagon equation:

\[
\sum_{\delta} F_{e}^{fcd}_{(g,\beta,\gamma)(l,\delta,\nu)} \left( f,\alpha,\delta \right) (k,\lambda,\mu) = \sum_{h,\sigma,\psi,\rho} F_{e}^{abc}_{(f,\alpha,\beta)(h,\sigma,\psi)} \left( g,\sigma,\gamma \right) (k,\lambda,\rho) \left( F_{e}^{hde}_{(g,\sigma,\gamma)(k,\lambda,\rho)} \left( h,\psi,\rho \right) (l,\mu,\nu) \right)
\]

Figure 8 is the graphical representation of the Pentagon equation. The fusion rules and a
\( F \)-symbol that satisfies the requirements we named above are two of the three properties that
define an anyon model. The third one will be introduced next.

5.1.2 Braiding Anyons

As already mentioned in the introduction of this chapter anyons have another special prop-
erty: Their behavior under braiding. Let \( R_{ab} \) be a clockwise exchange of anyons \( a \) and \( b \)
(see figure 9). Then \( R_{ab} \) is associated with an operator that acts on the splitting space \( V^{ab} \).
Applying it is called a R-move and it is defined as follows:

\[
R_{ab}|a, b; c, \mu\rangle = \sum_{\nu=1}^{N_{ab}} (R^{ab}_{e})_{\mu\nu} |b, a; c, \nu\rangle
\]
Similarly $R_{ab}^{-1}$ is a counterclockwise exchange of the anyons $a$ and $b$ (see figure 9) and is associated with the inverse operator of $R_{ab}$

$$ R_{ab}^{-1}|a, b; c, \mu\rangle = \sum_{\nu=1}^{N_{ab}^{c}} \left( R_{c}^{ba} \right)^{-1}_{\mu\nu} |b, a; c, \nu\rangle $$

Figure 9: Graphical representation of the clockwise and counterclockwise exchange for two anyons $a$ and $b$.

The matrix $\left[ R_{c}^{ab} \right]$ is unitary (and thus $R_{ab}^{1} = R_{ab}^{-1}$). If $N_{c}^{ab} = 1$ it is simply a number that can give an arbitrary phase, but if $N_{c}^{ab} > 1$ braiding the anyons $a$ and $b$ may yield a state that is not linearly dependent of the original state - though staying within the splitting space. The action of braiding can be incorporated into our splitting diagrams as show in figure 10.

Figure 10: Graphical representation and equation for $R_{ab}|a, b; c, \mu\rangle$

braiding and fusing must be compatible, i.e. it does not matter whether one first splits particle $d$ into particles $b$ and $c$ and afterwards braids both of them with particle $a$ or whether one first braids particles $a$ and $d$ and afterwards splits $d$ into $b$ and $c$. This requirement imposes two additional consistency equations onto $F$ and $R$. They are known as the Hexagon equations:
The Hexagon equations can be represented graphically too (see figure 11).

The last requirement for $R$ is $R_{a}^{11} = R_{b}^{1h} = 1$ as braiding with vacuum must not change the state from a physical point of view.

Adding a $R$-matrix that satisfies all of the named conditions to our set of fusion rules and $F$-symbol completes the anyon model.

### 5.2 Fibonacci Anyons

We want to give an example for an anyon model. The model we choose are the so called Fibonacci anyons. The Fibonacci anyon model consists of only two particles, the vacuum 1 and another particle $\tau$. The fusion rules for these two particles are given by

\[ 1 \times 1 = 1 \quad 1 \times \tau = \tau \quad \tau \times \tau = 1 + \tau \]
We see from these rules that the Fibonacci anyon model is Non-Abelian. They are called Fibonacci anyons because the sequence $dim \left(V_1^{11}\right), dim \left(V_1^{111}\right), dim \left(V_1^{1111}\right), ...$ is a Fibonacci sequence.

All the splitting and fusion spaces have dimension one. Therefore $\left[F_{abc}^d\right]$ reduces to a $2 \times 2$ matrix for all possible combinations of $a, b, c$ and $d$. If one of the four particles is vacuum, then the elements of the matrices $\left[F_{abc}^d\right]$ are 1 if the corresponding state is permitted by the fusion rules or 0 if it is not permitted. Only the $\left[F_{\tau\tau\tau\tau}\right]$ matrix is not trivial. It is given by

$$\left[F_{\tau\tau\tau\tau}\right] = \begin{pmatrix} \phi^{-1} & \phi^{-1/2} \\ \phi^{-1/2} & -\phi^{-1} \end{pmatrix}$$

where $\phi$ is the golden ratio $\frac{\sqrt{5}+1}{2}$.

As the splitting and fusion spaces all have dimension one, the $R$-matrices reduce to numbers. Braiding with vacuum does not change the state and therefore $R_{\tau}^{11} = R_{11}^{1} = 1$. The two remaining $R$-matrices are

$$R_{\tau}^{\tau\tau} = e^{-i4\pi/5} \quad R_{1}^{\tau\tau} = e^{i3\pi/5}$$

At this point the Fibonacci anyon model is simply a set of fusion rules, $F$- and $R$-symbols one can define and that happens to satisfy the constraints of an anyon model we listed earlier - notably the Pentagon equation and the two Hexagon equations as one can check. However, it turns out that braiding of Fibonacci anyons allows for universal quantum computation. On top of that they can be realized in the so called Fibonacci Levin-Wen model. This model will be introduced in the next chapter.
6 Fibonacci Levin-Wen Model

Levin-Wen models are spin lattice models. In general they can be used to realize any anyon theory. We will take a look at the Hamiltonian of the Fibonacci Levin-Wen model and see that its ground state subspace can be viewed as a stabilizer code. Subsequently we focus on the measurement circuits of the stabilizers as we already did with the surface code. To do so it will be necessary to make use of the $F$-move.

For a more detailed introduction to Levin-Wen models in general see [6] and for measurement circuits for the Fibonacci Levin-Wen model in particular see [7] and [8]. Most of the figures in this chapter are taken from [7] and [8]. There is an accordant remark in their captions.

6.1 Viewing the Levin-Wen model as a Stabilizer Code

Levin-Wen models are a general class of spin lattice models. We want to focus on the Fibonacci Levin-Wen model. It is defined on a hexagonal lattice. The spins are $\frac{1}{2}$-spins and are located on the edges, see figure 12. As $\frac{1}{2}$-spins are two level systems they can be viewed as Qbits as well. The Hamiltonian of this model is given by

$$ H = -\sum_v Q_v - \sum_p B_p $$

where the operators $Q_v$ are associated with the vertices of the lattice and act on the three Qbits that form the vertex while the operators $B_p$ are associated with the plaquettes of the lattice and act on twelve Qbits: The six Qbits that form the hexagonal plaquette plus the six Qbits that are connected to the hexagonal plaquette. The two operators are shown in figure 12 too.

The operators $Q_v$ and $B_p$ are not elements of $G_n$. The vertex operator $Q_v$ acts on the three Qbits $i,j$ and $k$ of a vertex as follows:

$$ Q_v |ijk\rangle = \begin{cases} |ijk\rangle & \text{if} \ |ijk\rangle = |000\rangle, |011\rangle, |101\rangle, |110\rangle, |111\rangle \\ 0 & \text{else} \end{cases} $$

The plaquette operator $B_p$ is more complicated and writing it down and explaining it explicitly is beyond the scope of this thesis. But its eigenvalues are 1 and 0 as well. So the ground state subspace of the system can be viewed as a stabilizer code as all the ground states satisfy $Q_v |\psi\rangle = B_p |\psi\rangle = |\psi\rangle$. However, opposing to the surface code, we do not introduce the Levin-Wen model to translate the stabilizer formalism into a practical example.

The lower excited states of the system give rise to Fibonacci anyons. Again this works quite similar to what we have seen for the “holes” in the surface code. An “excited plaquette” (which basically just means that the state of the system is not stabilized by the corresponding plaquette operator anymore) can be viewed as a quasiparticle. This particle is the anyon we labeled $\tau$ in the Fibonacci anyon model and can at the same time be used to encode a Qbit. Braiding these particles is universal for quantum computation.
Figure 12: In the Fibonacci Levin-Wen model the Qbits (blue dots) are placed on the edges of a hexagonal lattice. The vertex operator $Q_v$ acts on the three Qbits that form the corresponding vertex and the plaquette operator $B_p$ acts on the six Qbits that form the corresponding plaquette plus the six Qbits on the edges connected to the plaquette.
If one wants to actually realize the ground state subspace stabilizer code or the braiding of excited plaquettes it is essential to be able to perform the measurement of $Q_v$ and $B_p$. Like we did with the surface code we want to focus on these measurement circuits here as well.

### 6.2 Vertex Measurement Circuit

The vertex measurement is rather easy to perform and its circuit is shown in figure 13. It is similar to the plaquette measurement for the surface code and by considering all possible basis states of the three Qbits that are involved one quickly verifies that the outcome is indeed the eigenvalue of $Q_v$. The main (and apart from the number of Qbits only) difference between this circuit and the plaquette measurement circuit for the surface code is the four-Qbit Toffoli gate. It is required only because $Q_v|111⟩ = |111⟩$ and this equation is related to the non-Abelian nature of the Fibonacci Levin-Wen model. In fact if $Q_v|111⟩ = 0$ the operator $Q_v$ would be an element of the Pauli-group $G_n$.

![Figure 13: Vertex measurement circuit. The outcome of the measurement of the ancillary Qbit is 1 if the eigenvalue of $Q_v$ is 1 and $−1$ if the eigenvalue of $Q_v$ is 0. The figure has been taken from [7].](image)

Unlike the measurement circuit of the surface code the four-Qbit Toffoli gate requires every Qbit of the circuit to be coupled with every other Qbit of the circuit and not only with the ancillary Qbit.

### 6.3 $F$-move

Finding a circuit for measuring the plaquette operators $B_p$ is much more difficult. However, there are possible circuits and we want to have a look at two of them. Both make use of the so called $F$-move which is actually closely related to the $F$-Symbol of Fibonacci anyons.

The $F$-move acts on five Qbits $a, b, c, d$ and $e$ of two neighboring vertices. Its action can be graphically represented as shown in figure 14 a). The $F$-symbol in this figure is the $F$-symbol of Fibonacci anyons we introduced earlier. After performing the $F$-move on a ground state
of the Levin-Wen model the resulting state is not a ground state of the original Levin-Wen model anymore. However, we can locally redraw the lattice as shown in figure 14 b). The resulting state will then be a ground state of the new Levin-Wen model defined on this new lattice which includes two plaquettes with only five edges and two plaquettes with seven edges. Equivalent to this statement is the fact that the plaquette operator $B_p$ commutes with the $F$-move. Consequently the value of $B_p$ remains unchanged after the $F$-move even though the plaquette might have been reduced or expanded.

The $F$-move can be realized by the circuit shown in figure 15. The $F$-matrix applied on Qbit $e$ in this circuit is

$$F = \begin{pmatrix} \phi^{-1} & \phi^{-1/2} \\ \phi^{-1/2} & -\phi^{-1} \end{pmatrix}$$

where $\phi = \frac{\sqrt{5}+1}{2}$ is the golden ratio and therefore $F$ is the same matrix as $[F_{\tau\tau\tau}]$ for Fibonacci anyons. Note that in this circuit all five Qbits need to be coupled with each other because of the five-Qbit controlled $F$-gate.

As mentioned earlier we can make use of the $F$-move to measure the plaquette operator $B_p$ in two different ways. We will show them now.

### 6.4 Two Plaquette Measurement Circuits

The first measurement circuit will be called the plaquette reduction circuit. As stated above the value of $B_p$ remains unchanged when deforming the lattice by a $F$-move. Thus we can apply a sequence of $F$-moves that reduces an initially six sided plaquette to a one sided plaquette, a so called “tadpole”, see figure 16 a). Given a tadpole of Qbits $a$ and $b$ we can use the small $S$-circuit shown in figure 16 b) to measure the value of the reduced one.
Figure 15: Circuit realization of the $F$-move. The figure has been taken from [7].

Figure 16: a) Reducing a hexagonal plaquette to a tadpole by a series of $F$-moves. b) Measurement circuit for the one-sided plaquette $b$ of the tadpole. The measurement of the ancillary Qbit yields 1 if the value of the one-sided plaquette is 1 and it yields $-1$ if the value of the one-sided plaquette is 0. Both figures have been taken from [7].

sided plaquette formed by Qbit $b$ while preserving the state of the tadpole. The matrix representation of the 1-Qbit $S$-gate is given by

$$S = \frac{1}{\sqrt{1 + \phi^2}} \begin{pmatrix} 1 & \phi \\ \phi & -1 \end{pmatrix}$$

After the measurement of the tadpole we simply need to reverse the sequence of $F$-moves to obtain the original hexagonal plaquette. The value of this hexagonal plaquette is the value measured for the tadpole. The whole plaquette reduction circuit which makes use of one ancillary Qbit is shown in figure 17. In this circuit we make use of a $F'$-gate which is essentially a four Qbit $F$-move that reduces a two-sided plaquette to a tadpole. To perform the plaquette reduction circuit there are a lot of different couplings needed. However, apart from the Qbit labeled 12 in figure 17 none of the Qbits need to be coupled to every other Qbit but only to a few of them. In the following chapter we discuss the practical problem of finding an efficient way of connecting Qbits via TLRs. The plaquette reduction circuit will be a example we consider and we will give a detailed table of the couplings needed to perform
the circuit later.

The second measurement circuit we want to show is the plaquette swapping circuit. The idea behind this circuit is to add a tadpole to the lattice and use a sequence of $F$-moves to swap the tadpole for the hexagonal plaquette we want to measure (see figure 18). We initialize the tadpole in a state such that its plaquette operator has eigenvalue 1. So as $F$-moves preserve the value of the plaquette operators we end up with a hexagonal plaquette whose operator has eigenvalue 1 and a tadpole whose operator has the eigenvalue of the initial hexagonal plaquette and can be measured using the $S$-circuit in the same way as in the plaquette reduction circuit. While we reversed the sequence of $F$-moves in the plaquette reduction circuit to undo the reduction of the plaquette we do not need to undo the swap in the plaquette swapping circuit. After performing the circuit the hexagonal plaquette certainly has eigenvalue 1 and the tadpole has the measured eigenvalue. If the tadpole has eigenvalue 1 we can remove it. If it does not we need to move it through the lattice to annihilate it together with another tadpole whose measurement did not yield 1. But this kind of error correction via moving and fusing one sided plaquettes is beyond the scope of the thesis.

The whole plaquette swapping circuit is shown in figure 19. Apart from the $F$-move and $S$-circuit it makes use of the $F'$-move and of the Swap gate which is a 2-Qbit gate that swaps the states of the two Qbits it acts on. Moreover, we need three additional ancillary Qbits to perform this circuit.

Figure 17: Plaquette reduction circuit. It makes use of two $F'$-gates. The first one reduces the two-sided plaquette to the tadpole, and the second one reverses this reduction. The figure has been taken from [7].
Figure 18: Swapping a tadpole for a hexagonal plaquette using a series of $F$-moves. The figure has been taken from [8].

Figure 19: Swapping plaquette circuit. The figure has been taken from [8].
7 Optimal Linking of Qbit Layouts

In this chapter we consider the following problem: Suppose we have a certain set of Qbits and quantum gates that we wish to perform on these Qbits. To perform the gates the Qbits need to be coupled to each other and we would like to find the optimal way of linking the Qbits.

We start with a detailed description of this problem, followed by the description of the approach we take to find the optimal solution and the discussion of a example taken from the Levin-Wen model. Afterwards we consider the familiar problem of linking the unit cell of a lattice in order to be able to scale the lattice, its Qbits and the amount of gates to arbitrary size.

7.1 Posing the Problem

Suppose we are given a layout of \( n \) Qbits (labeled \( q_1, ..., q_n \)) fixed at certain positions \( \vec{r}_1, ..., \vec{r}_n \).

Using this layout we want to perform certain quantum circuits and as stated in the first chapter we assume that the performance of each one of these circuits demands a certain set of two Qbit couplings (i.e. the gates in use in these circuits are always constructed out of 1- or 2-Qbit gates). Consequently there is a set of 2-Qbit couplings \( E \subset (Q \times Q) \) (where \( Q = \{q_1, ..., q_n\} \)) that contains those couplings needed in order to perform all the circuits. Note that \( G = (Q, E) \) defines an undirected graph.

As already mentioned earlier one possibility to couple Qbits are transmission line resonators (TLR). It is quite clear that there was a unique “best” way of linking the Qbits if one TLR could only connect two Qbits: We would simply use the TLRs indicated by \( E \). However, there are indeed experiments showing that there is the possibility of connecting more than two Qbits at once. Suppose \( m \) is the maximal number of Qbits per TLR. Then, if \( m > 2 \), the problem of finding the best way (whatever that means - we will specify it shortly) of linking the Qbits is not trivial anymore and there will in general be solutions that are more favorable than just using 2-Qbit TLRs as indicated by \( E \) (see also figure 20). For later purposes we should keep in mind that a TLR connecting \( k \) Qbits \( q_1, ..., q_k \) can do so in different ways. We identify a TLR connecting \( k \) Qbits with a \( k \)-tuple. \( (q_1, ..., q_k) \) might in principle yield a different TLR from \( (q_{\sigma(1)}, ..., q_{\sigma(k)}) \) where \( \sigma \in \Pi_k \) denotes a \( k \)-permutation. This is simply because the TLR will physically look different if the Qbits are connected in a different order - however the performance of circuits will not be affected by a different order of linking.
The same experiments we just mentioned suggest additionally that the number of TLRs a Qbit can be part of are limited in practice. We denote the maximal number of TLRs per Qbit by \( p \).

Usual values for \( m \) and \( p \) are around 5.

To obtain a conception of what we call an optimal solution for the linking of Qbits we assume that there is a cost \( c_{i,j} \in \mathbb{R}^+ \) associated with every pair of Qbits \((q_i, q_j) \in (Q \times Q)\). A 2-Qbit TLR \((q_k, q_l)\) will then have the cost \( c_{k,l} \). For a \( r \)-Qbit TLR \( t = (q_{i_1}, ..., q_{i_r}) \) we set the cost to be \( \sum_{s=1}^{r-1} c_{i_s, i_{s+1}} \). As pointed out earlier there are different ways of linking a given set of \( r \) Qbits by a \( r \)-Qbit TLR. One of these ways will yield minimum costs. We will choose this TLR to be “the” TLR connecting the \( r \) Qbits and will in what follows ignore all the other possible ways of linking these Qbits (if there are more than one TLR yielding minimal costs we simply choose one of them and ignore the others). At this point we will introduce the second notation for the TLRs. We will have \( x_{i_1, ..., i_r} \) to note the TLR that connects the \( r \) Qbits \( q_{i_1}, ..., q_{i_r} \) and we further assume that a TLR does not link to a Qbit more than once. It is worth noting here that finding the TLR with minimal cost is actually a Traveling Salesman Problem, that can in practice be easily solved for small values of \( r \leq m \) by simply checking all possible ways of linking. However, if we imagine higher values of \( r \) this step might get computationally more intensive.

Now that we have defined these costs for all possible TLRs we can finally define the optimal solution. We will call a linking of the Qbits optimal if it allows for performance of all the circuits (that is every coupling in \( E \) is actually realized by the TLRs used), the number of its TLRs per Qbit is below or equal to \( p \) for all the Qbits and its costs (which are the sum of the costs of all its TLRs) are lower or equal to the costs of all other possible linking that fulfill these two requirements.
7.2 The Linear Optimization Approach

In order to find the optimal solution for the linking problem we posed above we aim to use Linear Optimization. We first introduce the formulation of our problem in the language of Linear Optimization and subsequently discuss solvability and algorithm performance for our problem.

7.2.1 The Linear Optimization Formulation

We set

\[ y = (x_{1,2}, x_{1,3}, \ldots, x_{n-1,n}, x_{1,2,3}, x_{1,2,4}, \ldots, x_{1,2,\ldots,m}, \ldots, x_{n-m+1,n-m+2,\ldots,n})^t \]

that is, \( y \) is simply the vector of all possible TLRs that link \( m \) or less Qbits. Or in other words: The vector of all TLRs we are allowed to use. Moreover, we set \( c \) to be the corresponding cost-vector, i.e. if \( y_i \) denotes the \( i \)-th component of \( y \), then \( c_i \) is the cost of the corresponding TLR. Furthermore, let \( d \) be the dimension of \( y \) and \( c \) and \( X = \{y_i | 1 \leq i \leq d\} \) the set of all TLRs. One finds that \( d = \sum_{i=2}^{m} \binom{n}{i} \).

Given the notations we have introduced so far it is now straightforward to formulate the problem we posed above by the means of Linear Optimization:

\[
\begin{align*}
\text{minimize} & \quad c^t y \\
\text{given} & \quad \sum_{x_k \in X : x_k\text{ yields a coupling of } q_i \text{ and } q_j} x_k \geq 1 \quad \forall (i, j) \in E \\
& \quad \sum_{x_k \in X : x_k\text{ links to } q_i} x_k \leq p \quad 1 \leq i \leq n \\
& \quad y \in \{0, 1\}^d 
\end{align*}
\]

Constraint (C3) implies that in the context of Linear Optimization we associate a transmission wire variable \( x_{i_1, i_2} \) with 0 or 1 and do not allow it to obtain any other value. If the variable is 1 the corresponding TLR will be used for our layout, if it is 0 it will not.

Solving this (binary) Linear Optimization problem is equivalent to the problem of finding the optimal linking in the sense we defined it above.

7.2.2 Solvability and Performance

Now that we have found a way of describing our problem in the context of Linear Optimization let us look at the solvability of such problems.

For Linear Optimization problems that are not restricted to integer (or binary) variables there are algorithms that can decide whether the problem is solvable or not and find an optimal solution if existent in polynomial time. Unfortunately this is not true anymore if we restrict
the variable to be integer or binary. The problem then becomes NP-difficult. However, there are algorithms that can solve these problems and find an optimal solution (or proof that the problem is not solvable).

It can be valuable to simplify the problem or adapt the algorithm to the given problem. We want to try the former and simplify the linear program we noted. We will do so by making the number of variables smaller. Our reduction is motivated by the following simple observation:

Given a set of $r$ Qbits $V' = \{q_{i_1}, ..., q_{i_r}\}$ and the set of required couplings between these Qbits $E' = \{(q_i, q_j) \mid q_i, q_j \in V' \text{ and } (q_i, q_j) \in E\}$, we see that $G' = (V', E')$ is a graph. Suppose that $G'$ is not a connected graph. Then it will always be more favorable to replace the TLR $x_{i_1,...,i_r}$ - let us call it a dispensable TLR - by two (or more) TLRs that connect only the two (or more) not connected subgraphs as these TLRs will still yield the same two Qbit connections of $E$ but have lower combined costs (see also figure 21). Therefore we will leave out all the dispensable TLRs. Probably there are more possibilities for simplifying the linear program. But this one alone is already enough to improve the performance of the algorithm we used significantly.

Figure 21: The dashed lines denote the required couplings for the layout and the red lines denote TLRs. None of the three Qbits on the left needs a coupling to one of the two Qbits on the right. Accordingly the 5-Qbit TLR that connects all of the Qbits can always be replaced by a 3-Qbit TLR and a 2-Qbit TLR and the replacement will have lower costs.

7.3 Example: Plaquette Reduction Layout

Using the Linear Optimization approach we want to find the optimal linking of a plaquette reduction layout that can be used to realize one plaquette of the Fibonacci Levin-Wen model and the measurement of its plaquette and vertex operators.

The plaquette reduction circuit that measures the value $B_p$ of a plaquette was shown in figure 17. A corresponding layout that (given the correct linking) allows for performance of this circuit is shown in figure 22. Additionally we require this layout to allow for performance of the measurement circuits of the six vertices surrounding the plaquette. This gives us a total of 19 Qbits (12 Qbits carrying the information on $B_p$ and $Q_v$ and 7 ancillary Qbits). Using the construction of the plaquette reduction circuit and the vertex measurement circuit we can write down all the 2-Qbit couplings we need for the performance (see table 1).
the costs of the TLRs we use the geometrical distance between the Qbits (the diameter of a hexagon is taken to be 2 length units (LU)).

Figure 22: Layout for the performance of the plaquette reduction circuit and the measurement circuits for all six surrounding vertices. Qbits 1 to 12 carry the information. Qbit 0 is ancillary for the plaquette reduction circuit and Qbits 13 to 18 are ancillary for the measurement of the vertices.
Table 1: Couplings needed among the Qbits to perform the plaquette reduction circuit and the measurement circuits for the vertices

<table>
<thead>
<tr>
<th>Qbit number</th>
<th>Couplings needed</th>
<th>TLR used</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>5 Qbit TW (0,6,7,12,18)</td>
</tr>
<tr>
<td>1</td>
<td>6,7,8,12,13</td>
<td>3 Qbit TW (1,2,6,8,12)</td>
</tr>
<tr>
<td>2</td>
<td>7,8,9,12,14</td>
<td>3 Qbit TW (2,7,8,9,14)</td>
</tr>
<tr>
<td>3</td>
<td>8,9,10,12,15</td>
<td>3 Qbit TW (3,4,9,11,12)</td>
</tr>
<tr>
<td>4</td>
<td>9,10,11,12,16</td>
<td>3 Qbit TW (3,8,9,10,15)</td>
</tr>
<tr>
<td>5</td>
<td>10,11,12,17</td>
<td>3 Qbit TW (5,10,11,12,17)</td>
</tr>
<tr>
<td>6</td>
<td>1,7,8,12,18</td>
<td>3 Qbit TW (1,7,13)</td>
</tr>
<tr>
<td>7</td>
<td>1,2,6,8,9,12,13,18</td>
<td>3 Qbit TW (4,10,16)</td>
</tr>
<tr>
<td>8</td>
<td>1,2,6,7,9,10,12,13,14</td>
<td>2 Qbit TW (8,13)</td>
</tr>
<tr>
<td>9</td>
<td>2,3,4,7,8,10,11,12,14,15</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>10</td>
<td>3,4,5,8,9,11,12,15,16</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>11</td>
<td>4,5,9,10,12,16,17</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>12</td>
<td>0,1,2,3,4,5,6,7,8,9,10,11,17,18</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>13</td>
<td>1,7,8</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>14</td>
<td>2,8,9</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>15</td>
<td>3,9,10</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>16</td>
<td>4,10,11</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>17</td>
<td>5,11,12</td>
<td>2 Qbit TW (11,16)</td>
</tr>
<tr>
<td>18</td>
<td>6,7,12</td>
<td>2 Qbit TW (11,16)</td>
</tr>
</tbody>
</table>

Table 2: Connections used in the optimal solution for the plaquette reduction layout

We use $p = m = 5$ as parameters. The program used for Linear Optimization here is lp_solve (free software that can be downloaded from http://sourceforge.net/projects/lpsolve/). It takes the algorithm a few seconds to find an optimal solution. This solution involves 6 5-Qbit-TLRs, 2 3-Qbit-TLRs and 2 2-Qbit-TLRs. The solution is given in figure 23. It looks a little crowded, so the TLRs in use are noted in table 2 separately.
7.4 Scaling a Unit Cell

As seen in the previous two subsections we can in principle solve the linking problem for an arbitrary layout of Qbits and required couplings. But it is also clear that this problem will soon become too complex to be solved by the usual algorithms of (integer) Linear Optimization within a reasonable amount of time if we add more and more Qbits to the layout. Although it might be possible to divide a given layout into several smaller sublayouts that can be solved subsequently yielding additional boundary conditions for the neighboring sub-layouts, we will in practice often be confronted with layouts that are defined on lattices and whose circuits look the same for equivalent parts of the lattice. In such cases it would be more favorable to have some kind of unit cell that includes a certain amount of Qbits and a corresponding set of required couplings, such that this unit cell with all its Qbits and TLRs can be shifted around in a way that all the shifted cells together form the lattice of Qbits and all the shifted TLRs together are sufficient to perform all required circuits (and are as cost-efficient as possible).

Instead of introducing a general theory of these unit cells and their linking we will only
consider one example that is of interest for us, which is again the Fibonacci Levin-Wen model. We choose to analyze the layout that allows for the plaquette swapping circuit on every plaquette as well as for the vertex measurement circuit on every vertex on a lattice of arbitrary size.

7.4.1 The Unit Cell

The plaquette swapping circuit for one plaquette was given in figure 19 and figure 24 a) shows a layout of 22 Qbits that (given the right couplings) can perform this circuit as well as the vertex measurement circuits for the six vertices surrounding the plaquette. Figure 24 b) shows the unit cell we choose to study. As in usual lattice theory there are two linear independent vectors ($\vec{v}_1$ and $\vec{v}_2$) such that the whole lattice can be reproduced by shifting the unit cell by an integer linear combination of these vectors. Moreover, the Qbits within the unit cell have been chosen in such a way that every Qbit of the lattice is a part of exactly one shifted unit cell and can consequently be uniquely defined by the corresponding linear combination of $\vec{v}_1$ and $\vec{v}_2$ and the relative position within the unit cell (see figure 24 for these positions). To be more precise we can set $q_i = (a_i, b_i; c_i)$ where $a_i, b_i \in \mathbb{Z}$ correspond to the integer linear combination of $\vec{v}_1$ and $\vec{v}_2$ of the unit cell in which Qbit $q_i$ is located and $c_i \in 0, 1, 2, \ldots, 8$ denotes the position within the unit cell. Shifting the Qbit $q_i$ can then be done by just adding $k \in \mathbb{Z}$ to $a_i$ ($b_i$) for a shift of $q_i$ by $k$ times $\vec{v}_1$ ($\vec{v}_2$). For later simplicity we introduce the following notation for a Qbit-shift:

Let $q_i = (a_i, b_i; c_i)$. We define $q_i + (k_1 \vec{v}_1 + k_2 \vec{v}_2) := (a_i + k_1, b_i + k_2; c_i)$

And the obvious transfer to a TLR-shift:

Let TLR $t = (q_{i_1}, \ldots, q_{i_r})$. We define

$t + (k_1 \vec{v}_1 + k_2 \vec{v}_2) := (q_{i_1} + (k_1 \vec{v}_1 + k_2 \vec{v}_2), \ldots, q_{i_r} + (k_1 \vec{v}_1 + k_2 \vec{v}_2))$
Figure 24: a) shows a 22-Qbit layout that can perform the plaquette swapping circuit as well as the vertex measurement circuits for every vertex surrounding the plaquette. b) shows the unit cell. It can be used to reproduce a Qbit lattice of arbitrary size whose plaquette layouts all look like the one in a).

In addition to figure 24 we write down table 3. This table contains all couplings needed for the performance of all plaquette swapping and vertex measurement circuits. But only those couplings that include at least one Qbit in the unit cell. For the labeling of the Qbits outside of the unit cell see figure 25 in the next part.

<table>
<thead>
<tr>
<th>Qbit number</th>
<th>couplings needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,2,3,6,7,8,11,13,15,17,46,51,53,60,62</td>
</tr>
<tr>
<td>1</td>
<td>0,6,7,8,11,13,15,17,18,20,21,24,26,60,62</td>
</tr>
<tr>
<td>2</td>
<td>0,3,4,6,8,36,37,42,43,44,46,51,53,60,62</td>
</tr>
<tr>
<td>3</td>
<td>0,2,46</td>
</tr>
<tr>
<td>4</td>
<td>2,36,37</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>0,1,2,7,8,11,18,20,27,28,29,36,37,46</td>
</tr>
<tr>
<td>7</td>
<td>0,1,6,8,11</td>
</tr>
<tr>
<td>8</td>
<td>0,1,2,5,6,7,11,18,20,27,28,29,36,37,46</td>
</tr>
</tbody>
</table>

Table 3: Couplings of the Qbits inside of the unit cell needed to perform all the required circuits

7.4.2 Equivalent Connections

In order to find a suitable linking of the unit cell we can limit our analysis to the 9 Qbits in the unit cell. This is because the required couplings for plaquette swapping and vertex measurement look the same in each unit cell. So if we find a linking of the 9 Qbits in the unit cell, such that all couplings required by $E$ that include at least one of these 9 Qbits are realized, we have consequently found a way of linking the whole lattice that is sufficient for the performance of all plaquette and vertex circuits. This linking will of course include
TLRs linking one or more of the 9 unit cell Qbits and Qbits outside of the unit cell and might as well include TLRs linking only unit cell Qbits. However, all TLRs and couplings required by $E$ that do not include at least one of the 9 unit cell Qbits can and will be ignored.

At this point we need to be careful. One could try to copy the exact same approach we took in the previous subchapter and use the same linear program we used there, making only the single one adjustment, that TLRs and couplings that do not include a Qbit from the unit cell will be ignored. This will yield a solution. But it will most probably (even though not tested in practice) be quite unfavorable compared to what one can achieve with a bit more effort. This is due to the existence of what we will call equivalent connections:

![Diagram](image)

$v1: (\sqrt{3}, -1)$
$v2: (\sqrt{3}, 1)$

Figure 25: The unit cell (labeled as unit cell 0) and six replications of it (labeled unit cell 1 to 6) that have been shifted by the vectors $\vec{v}_1$ and $\vec{v}_2$. These six unit cells form the first ring of cells around unit cell 0. Note that for solving the unit cell problem it is sufficient to only consider the first and second ring. The second ring (consisting of 12 unit cells and arranged around the first ring in a manner that unit cell 18 is right below unit cell 6 and is adjacent to unit cell 7 on its left side and unit cell 17 on its right side) has not been drawn here. Moreover, note that only the Qbits of unit cells 0,2 and 5 have been explicitly drawn to keep the figure clear. Generally the Qbits of unit cell $n$ are labeled as $9 \cdot n + m$ where $m$ indicates the relative position inside the unit cell as indicated by figure 24 b)

See figure 25. Imagine we try to find a unit cell linking by just adopting the linear program from the previous chapter as we just described. Suppose further the algorithm yields among others the two TLRs (6,20) and (2,51). You can easily see that if we afterwards shift the unit cell around and copy the TLRs to form the whole linked layout the TLR (2,51) shifted to unit cell 2 will be the exact same TLR as (6,20) was in the original unit cell. Analogous (6,20) shifted to unit cell 5 will be the exact same TLR as (2,51) in the original unit cell.
So there will be doubled TLRs, that are obviously superfluous. Such TLRs like (6,20) and (2,51) that can be shifted into each other will be called equivalent TLRs. We will give a more precise and general definition of equivalent TLRs shortly but let us hold on to the example of the TLRs (6,20) and (2,51) for a moment.

One might argue that a way of eliminating the doubling of those equivalent TLRs is to just check for them after having shifted the unit cell around. This is true and it should be a rather easy job to do so. However, there might be more kinds of inefficiencies that can not be tracked down as easily. For example the algorithm might yield (6,20) and (2,51,52) instead of only (2,51). Then the TLRs are not doubled but (6,20) can be omitted nonetheless. Or even worse for the afterwards tracking: Suppose the algorithm yields (6,7,20) and (2,4,52). These TLRs are not equivalent and they do not contain each other. However, it is easy to see that replacing both of them by either (6,7,20,22) or (2,4,51,52) (which are in fact equivalent) is a more favorable solution that yields the same couplings.

These examples do not mean that afterwards tracking for inefficiencies is impossible to do. But they show that this tracking might in fact be more difficult to realize than what we will do in the following. Moreover, they give a good overview of what problems we encounter when looking for a optimal - or at least good - solution for the unit cell linking.

Let us now define what we understand precisely as equivalent wires. We will make use of the notation \( q_i = (a_i, b_i; c_i) \) for a Qbit and also of the notations for Qbit- and TLR-shifts we introduced earlier.

Potentially there are infinite equivalent TLRs to every TLR in the lattice. But we are only interested in the linking of one unit cell, say for example the unit cell labeled 0 in figure 25. Its integer linear combination of \( \vec{v}_1 \) and \( \vec{v}_2 \) is (0,0) and we will leave out all TLRs that do not include a Qbit in this unit cell.

Two transmission wires \( t = (q_{i_1}, ..., q_{i_r}) \) and \( t^* = (q_{j_1}, ..., q_{j_r}) \) are called equivalent if:

1) \( \exists s,p \in \mathbb{N} : a_{i_s} = b_{i_s} = a_{j_p} = b_{j_p} = 0 \)

2) \( \exists k_1, k_2 \in \mathbb{Z} : t + (k_1 \vec{v}_1 + k_2 \vec{v}_2) = t^* \)

The first requirement ensures that both TLRs include a Qbit in unit cell 0 and the second requirement ensures that there is a shift that transforms one TLR into the other. This relation is an equivalence relation on the set of all TLRs that include at least one Qbit in unit cell 0. We choose a representative \( s([t]) \) out of every equivalence class \([t] \) and set \( S = \{ s([t]) \mid t \text{ includes a Qbit in unit cell 0} \} \).

### 7.4.3 Algorithm for Finding the Linking and Solution

Let us now formulate an algorithm for finding a linking of our unit cell. To do so we remember that a transmission wire \( t = (q_{i_1}, ..., q_{i_r}) \) is associated with the binary variable \( x_{i_1, ..., i_r} \) in the context of linear programs. We will make use of both ways of denoting a TLR:
Step 1: Reduce $E$ to $E'$ where $E'$ contains only the couplings that include at least one Qbit of the unit cell. Find all the possible TLRs as shown in the Linear Optimization approach excluding all dispensable TLRs with respect to $E'$. The set of these TLRs will be called $A^* = \{t_1, \ldots, t_{d^*}\}$.

Step 2: Expand the set $A^*$ to the set $A$, where $A = \bigcup_{t \in A^*} \{t\}$.

Step 3: Reduce the set $A^*$ to the set $B$, where $B = \bigcup_{t \in A^*} \{s([t])\}$. If $B = \{(q_{i_1}, \ldots, q_{i_r}), \ldots, (q_{d_1}, \ldots, q_{d_s})\}$, let $y^B = \left(x_{i_1 \ldots i_r}, \ldots, x_{d_1 \ldots d_s}\right)^t \in \mathbb{R}^d$ and let $c^B$ be the corresponding cost-vector.

Step 4: Write down the linear program to be solved:

\[
\begin{align*}
\text{minimize} \quad & (c^B)^t y^B \\
\text{given} \quad & \forall t \in A \\
(C1) \quad & t = s([t]) \\
(C2) \quad & \sum_{x_k \in A : x_k \text{ yields a}} x_k \geq 1 \quad \forall (i, j) \in E' \\
(C3) \quad & \sum_{x_k \in A : x_k \text{ contains } q_i} x_k \leq p \quad 0 \leq i \leq 8 \\
(C4) \quad & x_k \in \{0, 1\} \quad \forall x_k \in A
\end{align*}
\]

Step 5: Solve the linear program. For every TLR to be used all equivalent TLRs will be part of the solution too. Choose one TLR out of each equivalence class, the linking that corresponds to these representatives is the solution. The cost yielded by the algorithm is the cost of this linking (the negligence of all TLRs but one in every equivalence class has been taken into account here).

This algorithm has been applied for different values of $m$ and $p$. It turns out that linking the unit cell in a way that allows shifting it around to obtain a linking for the whole lattice without violating the constraints given by $m$ and $p$ is possible for $m = 4$ only if we allow $p \geq 7$. But in fact it is possible to find a wiring that has $p = 5$ for $m = 5$. This solution has a cost (or length of wiring) of 19.2 length units where the distance between two opposing edges of one hexagon is 2 length units. The solution for $m = p = 5$ is shown in figure 26. The TLRs in use are given in table 4.
Figure 26: Solution for the unit cell problem using the described algorithm. 5-Qbit TLRs are marked by continuous lines, 4-Qbit TLRs and 3-Qbit TLRs by dashed lines. Only Qbits that are part of a TLR have been drawn. However, when shifting the unit cell around every Qbit of the lattice (including those not drawn here) will have the couplings required by $E$.

<table>
<thead>
<tr>
<th>TLRs used</th>
<th>equivalent TLR 1</th>
<th>equivalent TLR 2</th>
<th>equivalent TLR 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-Qbit TLR</td>
<td>(6,8,20,27,28)</td>
<td>(0,1,11,60,62)</td>
<td>(2,36,37,51,53)</td>
</tr>
<tr>
<td></td>
<td>(0,9,11,15,17)</td>
<td>(0,2,6,8,36)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1,6,8,18,21)</td>
<td>(0,3,46,51,53)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2,41,42,43,44)</td>
<td>(5,6,7,8,11)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2,46,60,62,154)</td>
<td>(6,8,29,37,46)</td>
<td>(1,15,17,20,55)</td>
</tr>
<tr>
<td>4 Qbit TLR</td>
<td>(0,1,7,13)</td>
<td>(4,36,37,43)</td>
<td>(1,24,26,28,101)</td>
</tr>
<tr>
<td>3 Qbit TLR</td>
<td>(2,3,4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: TLRs used in the solution for unit cell problem. All equivalent TLRs are listed as well. Note that Qbit 101 and 154 are located in unit cell 11 and 17 respectively and are hence part of the second ring of Qbits around the unit cell (see also the remark in the caption of figure 25).
8 Layouts for the Surface Code

Using the algorithms introduced in the previous chapter we want to look at two layout problems related to the surface code. The first one is finding the TLRs of a unit cell for the usual surface code. The second one concerns the optimal linking of a certain patch layout.

8.1 Surface Code Unit Cell

We will look at the surface code unit cell. As already stated in chapter 4 the measurement circuits for the star and plaquette operator of the surface code require one additional ancillary Qbit each and to perform the circuit all four data Qbits need to be coupled with the ancillary Qbit but not with each other. Every Qbit is part of four measurement circuits (two stars and two plaquettes) and hence one finds that every Qbit - data Qbit as well as ancillary Qbit - needs to be connected to the four Qbits that are the nearest to it (see figure 27).

![Figure 27: Layout and couplings for the usual surface code. The “A” and “D” inside the blue dots denote ancillary and data Qbits respectively. The black dashed line is the lattice. The red lines are the two Qbit couplings that are needed to perform plaquette and vertex measurement circuits. The unit cell for this layout is shown on the right.](image)

Using our algorithm we find the following: If we allow TLRs that contain four or more Qbits \( (m \geq 4) \) the algorithm gives the solution shown in figure 28. Suppose the distance between a Qbit and its four nearest neighbors is one length unit, this solution gives a total distance...
The trivial solution that uses the 2-Qbit TLRs that correspond to the 2-Qbit couplings we need to perform the circuits has costs of 8 length units per unit cell and each Qbit is part of 4 TLRs.

Figure 28: Solution for the unit cell problem for the usual surface code layout. On the left side blue dots are Qbits within the unit cell while black dots are Qbits of neighboring unit cells. The red and green lines denote the two 4-Qbit TLRs that are used in a unit cell.

8.2 Rotated Patch with reused Ancillary Qbits

Given the linking we found for the surface code in the previous subchapter, optimal linking of finite surface code patches is done straightforward. Take the square lattice and cut a patch. We get the optimal linking for this patch layout if we simply cut the TLRs along the same cut lines. Consider for example the 13 Qbit surface code patch. It is the smallest possible layout for a surface code and its linking is given in figure 29. The 4-Qbit TLRs that are located at the boundary of the patch and are cut into two parts (one within the patch and one outside of it) are replaced by 2-Qbit TLRs. Everything else remains unchanged. However, we want to have a look at another patch and see what happens if we reuse some ancillary Qbits for multiple measurement circuits.
Look at figure 30. It shows a surface code patch consisting of 41 data Qbits (blue) and 40 ancillary Qbits (green). Each ancillary Qbit is associated with one measurement circuit - vertex or plaquette - and thus with one stabilizer. It turns out that we can cut off some of the Qbits without losing the distance of this code. The cut needed is marked red in the figure. We get a smaller rotated square. This rotated patch still has the same distance. Within this patch are 25 data Qbits (labeled 0 to 24) and 16 ancillary Qbits (labeled 25 to 40). As we would like the stabilized subspace to have dimension 2 the 16 stabilizer associated with the ancillary Qbits are not enough. Consequently we additionally include 8 of the stabilizers that are associated with the ancillary Qbits at the boundary of the rotated patch (marked by a red arc in figure 30 and labeled B1 to B8). Note that these stabilizers only act on two Qbits instead of four and accordingly the ancillary Qbits at the boundary only need to be coupled with two data Qbits.

This rotated patch layout can be used like a usual surface code. The optimal linking of the Qbits in this case is constructed by cutting a surface code lattice (and its TLRs) accordingly - just as we have seen earlier for the 13-Qbit patch. However, we are interested in using some of the ancillary Qbits for more than only one measurement circuit. That will allow us to remove some other ancillary Qbits.

First of all we omit Qbits 26, 29, 31, 34, 37 and 39 (omitted Qbits are marked by a black star in figure 30). All of these Qbits are ancillary plaquette Qbits. The respective plaquette measurement will use the ancillary vertex Qbit that is located in the upper right corner of the plaquette (that is Qbits 27, 30, 32, 35, 38 and 40 respectively). In order to do so these ancillary Qbits will each need two additional couplings with data Qbits. Note that we do not omit Qbits 28 and 36 although they are ancillary plaquette Qbits. That is because the corresponding upper right corner ancillary Qbit is not part of the patch. Of course one could...
use different Qbits that are actually within in the patch to replace these two Qbits. There are various possibilities to alter the choices we made here that seem just as reasonable. Next we omit all the ancillary Qbits at the boundary. The corresponding measurement circuits will instead use the ancillary Qbit within the patch that is the closest to the respective boundary Qbit. This leads to another complication for Qbits B7 and B8 as the Qbits that are the closest to these two are already omitted themselves. Qbit B7 will therefore be replaced by Qbit 38 and Qbit B8 will be replaced by Qbit 30. Note that making this choice will - given the replacements we already made before - result in no additional couplings. In table 5 the omitted Qbits are listed along with the Qbits that replace them in the associated measurement circuit.

Figure 30: Layout for the rotated patch surface code. The rotated patch is the red square. Data Qbits are blue and ancillary Qbits are green. The Qbits at the boundary marked by a red arc are associated with the stabilizers that will be used in addition to the stabilizers that are associated with ancillary Qbits within in the patch. Qbits marked by a black star are omitted and their respective measurement circuits use other Qbits to replace them.
Table 5: Ancillary Qbits we omit to cut down the number of Qbits and ancillary Qbits that replace these omitted Qbits in their measurement circuits

<table>
<thead>
<tr>
<th>omitted Qbit</th>
<th>replacing Qbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>B1</td>
<td>25</td>
</tr>
<tr>
<td>B2</td>
<td>27</td>
</tr>
<tr>
<td>B3</td>
<td>28</td>
</tr>
<tr>
<td>B4</td>
<td>36</td>
</tr>
<tr>
<td>B5</td>
<td>40</td>
</tr>
<tr>
<td>B6</td>
<td>38</td>
</tr>
<tr>
<td>B7</td>
<td>38</td>
</tr>
<tr>
<td>B8</td>
<td>30</td>
</tr>
</tbody>
</table>

Looking at table 5 one can see that Qbit 38 is now ancillary to four measurement circuits. There are layouts that have every ancillary Qbit to take part in only 3 measurements at the most but still have the same total amount of Qbits. But these layouts will require an extra coupling between a data Qbit and an ancillary Qbit. Which one of the two options is favorable is an engineering question.

The optimal linking of Qbits for the layout is shown in figure 31. It uses TLRs with a maximum of 5 Qbits per TLR and allows a Qbit to be part of 3 TLRs. A solution with 2 TLRs per Qbit is possible but its costs are about 8% higher than the solution shown. Allowing for more TLRs per Qbit does not change the optimal solution.
Figure 31: Optimal linking for the rotated patch surface code layout

Combined length (= costs) :
32.9 LU
9 Acknowledgments

I would like to thank David DiVincenzo for his guidance and all the time he took to follow and comment on the progress of my work. Furthermore, I would like to thank him for the permission to use some of his figures in chapter 6. I am very thankful to Fabio Pedrocchi who gave me advice and spent a lot of his time on helpful discussions with me.

10 References