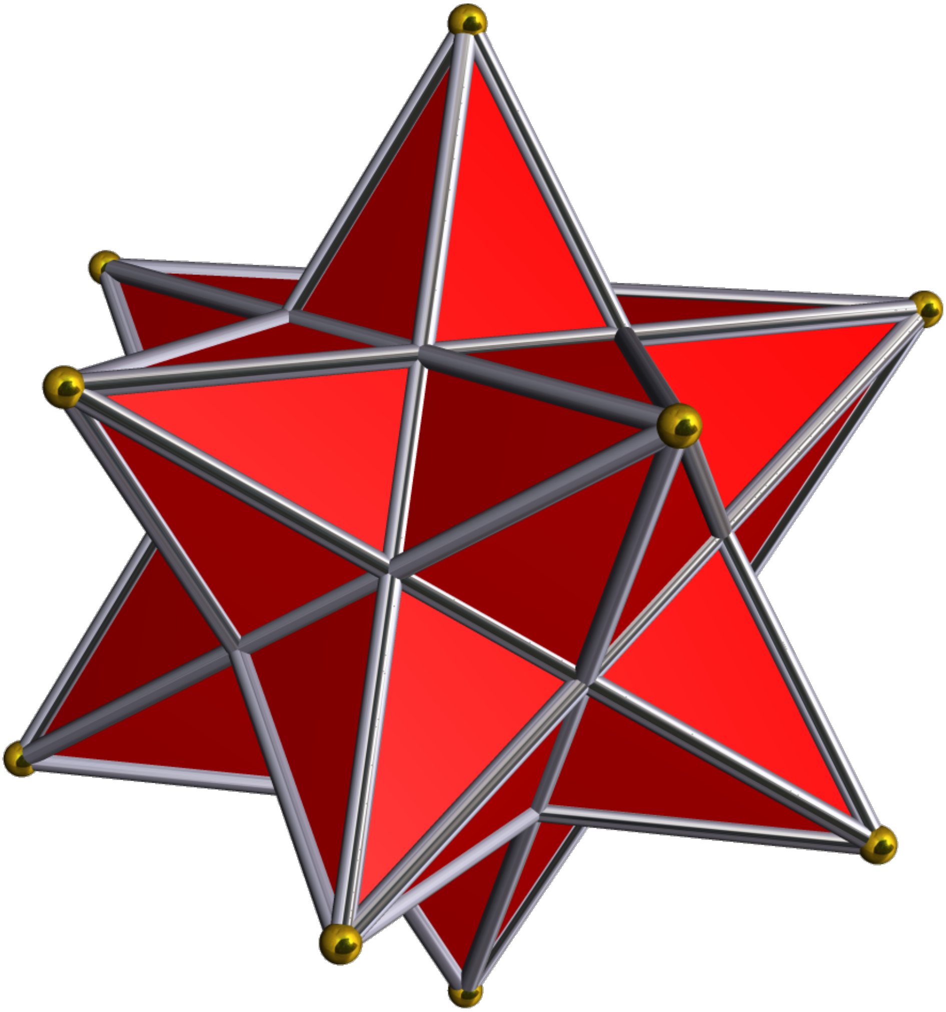
The Small Stellated Dodecahedron Code

Jochem Broshuis



The Small Stellated Dodecahedron Code

Finding interleaved measurement schedules

Bachelor Thesis

Technical University of Delft,
to be publicly defended on monday 18 november 2024

by

Jochem Broshuis

This dissertation has been reviewed by

Prof. B. M. Terhal	Technical University of Delft
Prof. L. DiCarlo	Technical University of Delft

Assessment committee:

Dr. J. L. A. Dubbeldam	Technical University of Delft
Dr. T. Taminiau,	Technical University of Delft

Independent members:

Dr. M. F. Rimbach-Russ,	Technical University of Delft

With the cooperation of

MSc. ir. M. Serra-Peralta,	Technical University of Delft

Contents

ABSTRACT

Quantum computers hold the potential to revolutionize computation by harnessing the unique properties of qubits. However, qubits are highly susceptible to errors, posing a major challenge in building reliable quantum systems. To address this, the development of quantum error correction codes is essential. The surface code has become a well-known code, using a 2D square grid to encode qubits. However, its high qubit overhead may be improved by alternative codes with more efficient encoding schemes.

This thesis focuses on the small stellated dodecahedron code, previously outperformed by the surface code, and proposes a new approach to optimize its performance by developing an algorithm that finds interleaved measurement schedules. The developed algorithm is tested on both the surface code and a code based on the Tetrahemihexahedron, correctly providing interleaved schedules for both. Applying the algorithm to the small stellated dodecahedron reduces the number of time steps required from 10 to 6, significantly boosting error correction performance. Simulations under a depolarizing noise model confirm the fault tolerance of these new schedules and demonstrate a 2.6x improvement in the code's pseudo-threshold compared to the original sequential schedule. While the surface code remains more effective at protecting against errors, performing 1.26 times better, the stellated dodecahedron code offers a compelling alternative for near-term research, requiring fewer qubits while maintaining strong performance.

SUMMARY

Computers have become an essential part of modern life, performing tasks that were once too time-consuming or complex for manual effort. They process and store information in the form of binary digits, or bits, which are represented as strings of 0s and 1s. This classical model of computing has enabled extraordinary advances in technology, science, and industry, but it is fundamentally limited by the constraints of binary logic and classical physics. In 1982, Richard Feynman proposed the concept of quantum computers, a revolutionary idea that utilizes the principles of quantum mechanics to process information in fundamentally new ways. Unlike classical bits, quantum computers use quantum bits, or qubits, which can exist in a superposition of both 0 and 1 simultaneously. Chapter 1 will act as an introduction, starting with a brief explanation about qubits, how to measure them, and will give a short introduction to operators. After the required basics, 3 quantum error correction (QEC) codes will be explained. First, Shor's code, which bundles qubits in packages of 3 to provide protection to incoming errors. By comparing mutual differences, it can be deduced what kind of errors have occurred and what correction has to be applied. Shor's code is an easy example of a class of codes called stabilizer codes. A short description of stabilizer codes will be given, before looking into how to develop syndrome extraction circuits. One well-known quantum error correction code is the surface code, a type of stabilizer code. It places qubits on a 2D grid and applies stabilizing plaquette operators, allowing for the correction of single errors. However, the surface code is limited by spatial constraints, which affect scalability. Hyperbolic surface tiling codes, such as those based on the small stellated dodecahedron, can correct single errors, and offer an alternative without these limitations. A previous study [1] explored the small stellated dodecahedron code but showed that it failed to outperform the surface code in terms of overall error protection. The study left open the question of whether the measurement schedule for qubits was optimal. A sequential schedule requiring 10 time steps was used, first measuring X-type stabilizers, then Z-type stabilizers. This thesis presents a heuristic algorithm to find interleaved schedules, where X and Z stabilizers are measured simultaneously, reducing the number of time steps needed.

The finding of an interleaved schedule boils down to finding an edge coloring on the Tanner graph of the SSD, which will be the main topic of Chapter 2. A normal edge coloring algorithm is made, and a length-5 coloring is found. By propagating Pauli's through CNOT's, it can be seen that this length-5 schedule does not actually measure the stabilizers properly. In order to develop an algorithm that produces proper interleaved schedules, a theorem is used which states that each pair of qubits sharing the same X and Z stabilizers should both either first interact with the X or with the Z stabilizer. Based on this restriction, the edge coloring has to be adjusted to return proper interleaved schedules.

In Chapter 3, it will be explained how, by identifying all the overlapping groups of qubits, the constraints imposed by this theorem are addressed. This results in some edges getting temporarily blocked and removed from the graph. The remaining graph is converted into a line graph and a maximal independent set is chosen and assigned a color. This set gets permanently removed from the graph, and the process of finding a new colorable set starts again. An improvement is made to the algorithm by enabling it to detect certain blocking sets; cycles of blocked edges which only can be restored by the coloring of another blocked edge in the same cycle. This improvement decreases the overall runtime of the algorithm by a factor 3.

The algorithm was used to develop 18 different proper interleaved schedules for the SSD. The found schedules were of length 6, being 4 layers of CNOT's shorter than the sequential schedules used before. Besides the SSD, the algorithm also is tested on the surface code and the Tetrahemihexahedron code, yielding proper interleaved schedules for both. However, for these smaller codes, the detecting of blocking cycles poses no improvements, as the additional runtime required to perform this check nullifies the effect of prematurely halting the algorithm.

The newly found interleaved schedules are then simulated using Stim. A circuit-level depolarizing noise model is used to simulate the random type of errors occurring on the qubits. Using this noise model, Chapters 5 and 6, will focus on testing the schedules to ensure single errors are detected, corrected, and do not spread. Simulations using a circuit-level depolarizing noise model showed that the interleaved schedules of the stellated dodecahedron provide a 2.6 times improvement over sequential schedules when looking at logical error rates. While the distance-3 surface code still offers better protection, the interleaved schedules only perform 1.26 times worse. Since the surface code requires 136 qubits, and the stellated dodecahedron uses only 54, the SSD proves to be a promising overhead-reducing alternative for near-term experiments.

1

QUANTUM ERROR CORRECTION

The world has changed in many ways due to computers, which now play a major role in our daily lives. From managing air traffic control to powering electric cars and enabling electronic communication through the Internet, computers are present in almost every aspect of life. While computers allow us to solve many problems that were simply impossible before their invention, several problems require too much computational power to be practical even for relatively simple inputs, even when using the most high-end computers. The Church-Turing Thesis [2] states that every physical implementation of universal computation can simulate any other implementation with only a polynomial slowdown. This would imply that for classical computers the complexity and scaling of problems would remain the same, despite the overall performance going up.

Feynman was the first to come up with the conjecture that a computer based on quantum mechanics could intrinsically be more powerful than any classical computer [3]. Whilst this might seem sensible when looking at the way quantum mechanics allows ways to process information by entanglement and superposition, which are classically impossible, it was in fact quite revolutionary as it suggests that the strong Church-Turing Thesis could be inapplicable for quantum computers. Feynman's conjecture suggests that while certain problems are complex for classical computers, they may be efficiently solved by quantum computers, potentially making some classically unsolvable problems solvable. However, stating that a computer based on quantum mechanics could outperform a classical computer is of course still a long way from actually building one. Any quantum computer needs a system with long-lived quantum states and ways to interact with them. It has become the standard to consider systems comprised of several two-state subsystems, which are called quantum bits, or in short qubits. Some possible physical realizations of qubits are:

1. the ground and excited states of ions stored in a linear ion trap, with interactions between ions provided through a joint vibrational mode [4],
2. nuclear spin states in polymers, with interactions provided by nuclear magnetic resonance techniques [5],

3. photons in either polarization, with interactions via cavity QED [6],

4. Superconducting qubits, which use superconducting circuits that create distinct energy levels. [7]

All these suggested implementations of qubits share a higher susceptibility to errors than normal bits, around 10^{-4} for qubits [8] and 10^{-14} for bits [9]. Especially the entangled states are exceptionally susceptible to errors. So even when possible errors could be reduced to some small probability per time step, the probability of surviving without an error occurring would be exponentially decaying with the total amount of time. So if an algorithm runs in polynomial time on an error-free computer, it will require exponentially many runs on a noisy computer unless something can be done to control the errors [10].

Now these problems also hold for classical computers, and error correction codes have been developed to minimize corruption. However, these classic codes can not be directly applied to the qubits of a quantum computer. This is partly because classical codes assume all bits can be measured. All qubits can be measured too, but measuring in the computational basis (0 and 1) would collapse the quantum state and destroy any superposition or entanglement. This is why quantum error correction codes use specific measurement strategies involving stabilizers or syndromes, which allow us to extract error information without measuring the actual qubit values. The main difference is the fact that in classic computers only the bit values 0 and 1 had to be saved, but that for quantum computers possible phase differences should also be taken into account. Thus although quantum error correction codes are closely related to classical codes, a new approach has to be taken when regarding superposition and entanglement [10].

A very common quantum error correction code is called the surface code. It is a stabilizer code and uses a standard 2D grid with local stabilizers to store the logical qubits. The surface code has proven to have a good error threshold $p \approx 1\%$ and decent scalability [11]. In this thesis, the small stellated dodecahedron code will serve as a central focus. Previous research [1] made use of sequential syndrome extraction schedules, resulting in an underwhelming performance. The main goal is to develop an algorithm that produces interleaved schedules, attempting to improve the overall performance of the code. In this chapter, a basic summary is given of qubits and noise acting on them. Then stabilizer codes and measuring circuits are explained, before looking into the distance-3 surface code. At last, we will look into the benefits coming from the use of pentagrammic faces and what questions have to be answered for the small stellated dodecahedron (SSD) to work as a stable error correction code.

1.1. QUBITS

The bits in a classical computer are represented by a string of 0s and 1s, which correspond to a vector of the finite field $\mathbb{F}_2$. Qubits in a quantum computer instead are represented by a vector in $\mathbb{C}$. A classical computer with n bits has 2^n possible states, but this is only an n-dimensional vector space over F_2. A quantum computer with n qubits is a state in a 2^n-dimensional complex vector space. For a single qubit, the standard computational vectors are written as $|0\rangle$ and $|1\rangle$. An arbitrary single-qubit state can then be written as

$$|\Psi\rangle = a|0\rangle + b|1\rangle, \tag{1.1}$$

with a and b complex numbers and $|a|^2 + |b|^2 = 1$. As a shorthand notation the states $|+\rangle$ and $|-\rangle$ are used to denote

$$\begin{aligned} |+\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \\ |-\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \end{aligned} \tag{1.2}$$

It is also possible to create entangled states between multiple different qubits, which cannot be expressed as the product of single-qubit states,

$$|\Psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{1.3}$$

In order to measure the qubits one can project them onto any basis of our Hilbert space. For example, measuring with respect to $|0\rangle, |1\rangle$ will return outcome 0 with probability $p(|0\rangle) = |\langle 0|(a|0\rangle + b|1\rangle)|^2 = |a|^2$ and 1 with probability $|b|^2$. However measuring it in the basis $|+\rangle, |-\rangle$ will return + with probability $p(|+\rangle) = |(|0\rangle + |1\rangle|^2 = \frac{|a+b|^2}{2}$ and − with probability $\frac{|a-b|^2}{2}$. Note that measuring any qubit of an entangled state will destroy the entanglement. Throughout most of this thesis, I will write down unnormalized states as they are shorter and easier to read. This does not lead to a loss of generality, as only the relative phase between qubits is relevant.

Operators in quantum mechanics are mathematical entities used to represent physical processes that result in the change of the state vector of the system. A very common basis of operators for single-qubit systems is the group $\mathscr{P}$ formed by the Pauli spin matrices,

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

which can be used to change the state of single qubits. For example $X|0\rangle = |1\rangle$, and also $X|1\rangle = |0\rangle$, where it can be see that the Pauli X exactly flips each bit, thus a Pauli X is often called a bitflip. The Pauli Z has no effect on $|0\rangle$ as $Z|0\rangle = |0\rangle$, but flips the phase of $|1\rangle$, $Z|1\rangle = -|1\rangle$. More interestingly, the Pauli Z works as a bitflip on the states $|+\rangle$ and $|-\rangle$, resulting in $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$. Now the Pauli X operator takes over the function of a phase shift,$X|+\rangle = |+\rangle$ and $X|-\rangle = -|-\rangle$ It can be seen that $iXZ = Y$, such that the Pauli Y has the same effect of combining the Pauli X and Z, with the addition of a phase factor i. Another important characteristic is that hermitian operators in quantum mechanics represent measurements. Specifically, they correspond to observable quantities that can be measured in a quantum system. When a hermitian operator acts on a quantum state, it relates to the possible outcomes of a measurement of that observable, with the operator's eigenvalues representing the potential measurement results. The Pauli's matrices form a basis for all hermitian matrices, enabling a code that protects against the Pauli operators to protect against all hermitian errors.

The Pauli matrices satisfy an important algebraic property, they all pairwise anticommute. That is, $\forall A, B \in \mathscr{P}$ with $A \neq B$

$$\{A,B\} = AB + BA = 0 \tag{1.4}$$

Another useful property is that every Pauli operator satisfies $X^2 = Z^2 = Y^2 = I$, showing that applying an even amount of a Pauli operator after another will not change the original state. Pauli's can also be used on multiple qubits, simply by using tensor products. $X \otimes I \otimes Z = XIZ$ denotes a Pauli X on qubit 1, the identity on qubit 2 and Pauli Z on qubit 3. The set of all tensor products of Pauli's for a system with n qubits forms a group $\mathscr{P}_n$ under multiplication [12].

1.2. NOISE AND SHOR CODE

The most general one-qubit error that can occur is a linear combination of X, Y, Z, and I. As it has been shown that Y can also be expressed in X and Z by $iXZ = Y$ and that the Pauli's form a basis for all unitary matrices, protecting qubits against Pauli X and Z errors offers a protection against all other unitary single qubit-errors. To this end, Shor [13] suggested encoding the logical qubits in different groups of data qubits, like

$$\begin{aligned} |0_L\rangle &= (|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \\ |1_L\rangle &= (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle) \end{aligned} \tag{1.5}$$

can offer a form of protection against incoming errors, since now the data isn't stored in a single qubit, but spread out over 9 different qubits. Suppose a Pauli X error acts on the first qubit, flipping it from $|0\rangle$ to a $|1\rangle$. It is not possible to directly measure the first qubit to find out which value it has, as this would destroy the superposition created by collapsing it into a state corresponding to the outcome of the measurement. Instead by measuring the parity of the first qubit with the second and third qubit, it can be determined if the first qubit had been flipped or not. I.e. first compare the first and second qubits, see that they are different, and conclude that a Pauli X has worked on one of them. By comparing qubits 2 and 3, see that they are the same, but comparing qubits 1 and 3 shows they are different. With the assumption only 1 error has occurred, the conclusion is that qubit 1 was flipped.

Assume a Pauli Z error occurring on the first qubit. This would change the logical qubits into,

$$\begin{aligned} |0_L\rangle &= (|000\rangle - |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \\ |1_L\rangle &= (|000\rangle + |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle) \end{aligned} \tag{1.6}$$

By comparing the sign of the first block of three with the second block of three, it can be seen that a sign error has occurred in one of those blocks. Then by comparing the signs of the first and third blocks of three, it is possible to narrow the sign error down to the first block and flip the sign back to what it should be. This choice for logical 0 and 1 thus seems to protect against a single qubit error.

The code uses 9 data qubits to encode 1 logical qubit, but has poor scaling when increasing the amount of encoded logical qubits and needs multiple rounds of parity measuring [13]. A more structured and better encoding is to be found in other stabilizer codes, which use fewer qubits and provide better error correction capabilities by using a more comprehensive set of stabilizing operators.

1.3. STABILIZER CODE FORMALISM

The stabilizer subgroup S of the Pauli group $\mathscr{P}_n$, is a specific abelian subgroup whose elements, called stabilizers, define a subspace known as the codespace T. As S stabilizes T, it must hold that $T = \{|\psi\rangle \text{ s.t. } M|\psi\rangle = |\psi\rangle, \forall M \in S\}$. S must be an abelian set for all the stabilizers to commute, as only commuting operators can have simultaneous eigenvectors which are needed to form T. As described before, the code only has to protect against all possible errors originating from $\mathscr{P}_n$, as Pauli's form a conventional basis.

In order for the code to correctly distinguish between two errors E_a and E_b, we must always be able to distinguish error E_a acting on the first codeword $|\psi_i\rangle \in T$ from error E_b acting on a different codeword $|\psi_j\rangle \in T$. This can only be true if $E_a|\psi_i\rangle$ is orthogonal to $E_b|\psi_j\rangle$; otherwise, there would be some overlap between the states, meaning that a measurement designed to detect errors could lead to ambiguous results. So it must hold that

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = 0 \text{ , for } i \neq j \tag{1.7}$$

Let's take $M \in S$, $|\psi_i\rangle \in T$ and some error E with $\{M, E\} = ME - EM = 0$. Then $ME|\psi_i\rangle = -E|\psi_i\rangle$, but moreover that

$$\langle\psi_i|E|\psi_j\rangle = \langle\psi_i|ME|\psi_j\rangle = -\langle\psi_i|E|\psi_j\rangle = 0 \tag{1.8}$$

when $E = E_a^\dagger E_b$ and E anti commutes with M for some $M \in S$. Therefore, if $E_a^\dagger E_b$ anti commutes with some element of S for all errors E_a and E_b in some set, the code will correct that set of errors by applying the inverse.

Now most of the time $\mathscr{P}_n$ will contain elements that commute with everything in S, but are not in S. These elements form the centralizer C(S) of S in $\mathscr{P}_n$, which in this case actually is equal to the normalizer N(S) of S in $\mathscr{P}_n$. The normalizer fixes S under commutation. To see why this equals the centralizer, observe that for $A \in \mathscr{P}_n$ and $M \in S$,

$$A^\dagger MA = \pm A^\dagger AM = \pm M \tag{1.9}$$

As $-I \notin S$, since $-I|\psi\rangle \neq |\psi\rangle$, the only way A can keep M in S is through commutation $A^\dagger MA = M$, leading to the fact that N(S)=C(S).

Suppose there is an error $E \in N(S) - S$, then this error will change our codeword $|\psi\rangle \in T$, but it will still remain in T. If $M \in S$ and $|\psi\rangle \in T$ then M and E commute, so

$$ME|\psi\rangle = EM|\psi\rangle = E|\psi\rangle \tag{1.10}$$

showing that the state $E|\psi\rangle$ gets fixed by S, and thus also is in T. However as $E \notin S$, and thus is not a stabilizer of the whole space T, there must exist at least one $|\psi\rangle \in T$ which does not get fixed by E. Errors from this set $N(S) - S$ thus form an undetectable set of errors.

Since the elements of N(S) move encoded states around within T, they have a natural interpretation as encoded operations on the states. Since S fixes T, only N(S)-S acts on T non-trivially. These elements are called logical operators, errors coming from this set are called logical errors. The logical $\overline{X}$ and $\overline{Z}$ operators must commute with all $M \in S$ by being in the normal subgroup, but moreover there must hold that:

1

$$
\begin{aligned}
&[\overline{X}_i, \overline{X}_j] = 0 \\
&[\overline{Z}_i, \overline{Z}_j] = 0 \\
&[\overline{X}_i, \overline{Z}_j] = 0\,, \quad (i \neq j) \\
&\{\overline{X}_i, \overline{Z}_i\} = 0
\end{aligned}
\tag{1.11}
$$

The distance d of a stabilizer code refers to the minimum number of physical qubits that are affected to result in a logical operator. It determines the code's ability to detect and correct errors. Specifically, a code can detect errors of weight up to $d-1$ qubits and can correct errors affecting up to $\lfloor \frac{d-1}{2} \rfloor$ qubits.

The notation [n,k,d] is commonly used to characterize quantum error-correcting codes. Here, n represents the number of physical qubits employed to encode the information, while k indicates the number of logical qubits encoded. The parameter d denotes the minimum distance of the code[10].

1.4. SYNDROME EXTRACTION CIRCUITS

To check whether noise has inflicted an error on one of the qubits, the stabilizers are measured. We will start by looking at the basics of quantum circuits, and build up to these stabilizer measurements. In a quantum circuit time proceeds from left to right, wires represent qubits and single-qubit gates are represented by a rectangle, with the respective operator written inside. Figure 1.1 shows a short list of the most used operators.

A two-qubit operation that will be heavily used is the controlled-NOT. As it is a two-qubit gate, it takes two qubits as input, one known as the control qubit and the other as the target qubit. It is drawn as shown in Figure 1.1. The action of the CNOT is given by $|c\rangle\,|t\rangle \rightarrow |c\rangle\,|c \oplus t\rangle$. That is, if the control qubit is set to $|1\rangle$ then the target qubit is flipped, otherwise the target qubit is left alone. Qubit measurements are depicted with either a box with a meter or a black box with the respective measurement basis written in it [14].

More generally, suppose U is an arbitrary single-qubit operation. A controlled-U operation again is a two-qubit operation, with a control and a target qubit. If the control qubit is set to 1, then U is applied to the target qubit; that is $|1\rangle\,|t\rangle \rightarrow |1\rangle\,U\,|t\rangle$. When the control qubit is set to 0, then I is applied to the target qubit $|0\rangle\,|t\rangle \rightarrow |0\rangle\,I\,|t\rangle$. The controlled-$U$ operation is represented by the circuit shown in Figure 1.2. Controlled operations can be used to measure the occurrence of an error on a qubit, without directly measuring the qubit itself. The following circuit uses an ancilla qubit to measure the hermitian operator M on the bottom qubit.

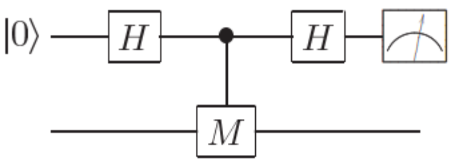


Figure 1.3: Circuit for measuring a single qubit hermitian observable M. The top qubit is the ancilla used for the measurement, and the bottom qubit is being measured.

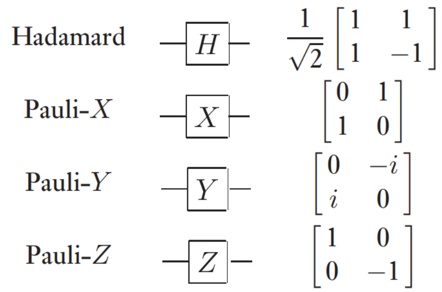


Figure 1.1: Names, symbols and matrices for the common single qubit gates [14].

Figure 1.2: Left: Circuit including a single CNOT and a measurement of the top qubit. The black dot represents the control qubit, the ⊕ the target qubit. Right: Circuit including a single controlled-U. The black dot represents the control qubit, the square with U the action of applying U to the target qubit.

The working of the circuit in Figure 1.3 can be checked to follow Equation 1.12. Here the first qubit denotes the top qubit, the second qubit the bottom qubit.

$$
\begin{aligned}
|0\psi\rangle \xrightarrow{H_1} |0\psi\rangle + |1\psi\rangle \xrightarrow{CM} |0\psi\rangle + |1(M\psi)\rangle \xrightarrow{H_1} |0\psi\rangle + |1\psi\rangle + |0(M\psi)\rangle - |1(M\psi)\rangle \\
= |0\rangle\,(I+M)\,|\psi\rangle + |1\rangle\,(1-M)\,|\psi\rangle
\end{aligned}
\tag{1.12}
$$

Measuring the top qubit in the Z-basis will either result in measuring $|0\rangle$ or $|1\rangle$. By measuring $|0\rangle$, the resulting state for the second qubit must be $(1+M)\,|\psi\rangle$, the $+1$ eigenstate. Similarly, measuring $|1\rangle$ shows that the remaining state must be $(1-M)\,|\psi\rangle$, the -1 eigenstate. When protecting the qubits from noise-induced errors, our primary focus is on detecting if a Pauli X or Pauli Z error has occurred.

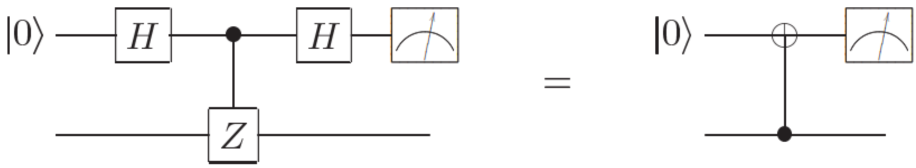


Figure 1.4: Simplification of the circuit used to measure the Z operator into a circuit containing a single CNOT.

Luckily, the measurement circuit for measuring a Pauli Z operator can be greatly simplified to just using a single CNOT, see Figure 1.4. Again, we have an ancilla qubit in $|0\rangle$ and an arbitrary state $|\psi\rangle = a|0\rangle + |b\rangle$. Working out the resultant state of the left circuit, one will end up with the state $|\psi\rangle = a|00\rangle + b|11\rangle$. However, this resultant state is the same as entangling the ancilla qubit with an arbitrary qubit state via a single CNOT, see Equation 1.13.

$$|0\psi\rangle = |0\rangle\,(a|0\rangle + b|1\rangle) \xrightarrow{CNOT} a|00\rangle + b|11\rangle \tag{1.13}$$

A similar thing holds for the measurement of a Pauli X, where again the schedule can be greatly reduced into a form where only a single CNOT is used.

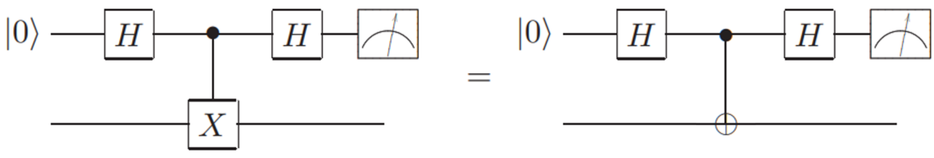


Figure 1.5: Simplification of the circuit used to measure the X operator into a circuit containing a single CNOT.

Depending on the outcome of the measurement of the ancilla qubits, it can deduced in what state the data qubit must be. For example, measuring with a Z stabilizer the circuit ended up in the state $a|00\rangle + b|11\rangle$. If the measurement outcome of the ancilla is $+1$, we know the data qubit must be in $|0\rangle$. Vice versa, if the outcome of the measurement is -1, the data qubit must be in the state $|1\rangle$.

1.4.1. STABILIZER MEASUREMENTS

In the previous section, a circuit has been shown designed to measure the Pauli X and Z using an ancilla qubit. These circuits also work for measuring multi-qubit Pauli strings. For example, suppose a X stabilizer and Z stabilizer are working on qubits 1 and 2, as in Figure 1.6. As the Pauli X and Z anticommute, X_1X_2 and Z_1Z_2 commute allowing the arbitrary state of qubits 1 and 2 to be collapsed into simultaneous eigenvectors. The circuit given in Figure 1.6 can be used the measure the X and Z operators on both qubits.

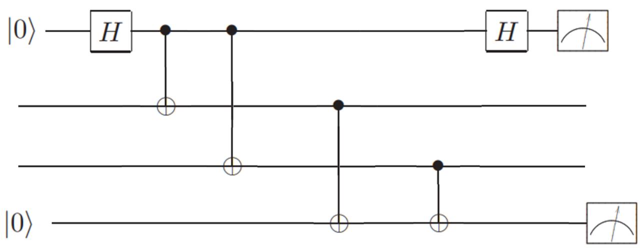


Figure 1.6: Circuit to measure both the X and Z stabilizer acting on two data qubits.

It can be seen that taking any resultant state and remeasuring the stabilizers will return the same state, with the same measurement outcomes. For example, if measurement outcomes indicate $M_Z = +1, M_X = +1$ the resultant state is $|\psi\rangle = |00\rangle + |11\rangle$ corresponding to $a = 1, d = 1$. If this state is remeasured, the state $|\psi\rangle = |00\rangle + |11\rangle$ gets returned.

The stabilizers can be viewed as constraints that project the total quantum state into a stabilized state. In this process, each stabilizer contributes to halving the dimension of the space of states. So, after measuring the ancilla qubits for the first round we end up in a quiescent state which (with no errors), will result in the same measurement outcomes each round. Assume that now an error which anti commutes with one of the stabilizers occurs, then this should in theory be possible to detect, see Equation 1.8. In this case, there is a X_1X_2 stabilizer and a Z_1Z_2 stabilizer. As single Pauli's anticommute with each other, a single Z error on either of the qubits should signal disturb the measurement outcome for X_1X_2. The same should hold for a single X error, disturbing the outcome of measuring Z_1Z_2.

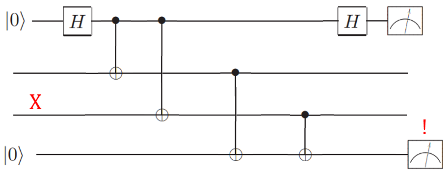


Figure 1.7: A single X error flips the measurement outcome of the Z ancilla qubit.

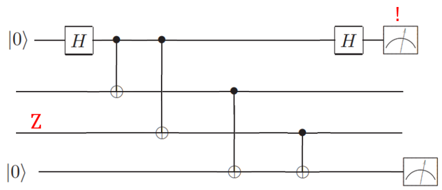


Figure 1.8: A single Z error flips the measurement outcome of the X ancilla qubit.

Taking an arbitrary stabilized state from, measuring the X and Z stabilizers again, but now introducing an X error on the second qubit at the start of the second round, see Figure 1.7. I.e. starting with $a = 1$ and $d = 1$, then the first round of measurements returns $\{+1, +1\}$, however the state gets changed into

$$|\psi_{1,2}\rangle = X_2(|00\rangle + |11\rangle) = |01\rangle + |10\rangle, \tag{1.14}$$

when performing the second round of measurements. Measuring this state will return $\{+1, -1\}$, where the Z stabilizer has now changed into -1 showing that an X error has occurred on one of the qubits. Suppose that instead of an X error, a Z error would have occurred on the qubits, then the X stabilizer's measurement outcome would have been flipped, indicating the occurrence of a Z error. The code now consists of a first-round where the data qubits are put into a stabilized state, followed by subsequent measurement rounds where each measurement outcome difference gets noted [14]. In the end, a minimum weight decoder can effectively identify the errors that have occurred based on the switches in measurement outcomes. By analyzing these switches, it determines the shortest possible error path associated with the observed outcomes [15].

1

1.5. DISTANCE-3 ROTATED SURFACE CODE

A simple example of a stabilizer code is the distance-3 surface code, here 9 data qubits are located on the vertices of a 3x3 square grid, see Figure 1.9. There is a checkerboard pattern of X and Z stabilizers, with weight 2 stabilizers along the edges. All X and Z stabilizers have an overlap on 2 qubits, satisfying the restriction that stabilizers must commute. The 9 qubits make the total state space 2^9 dimensional, however, each stabilizer halves the space by factor 2. The dimension thus gets reduced to $2^{9-8} = 2^1$, showing that this code encodes 1 logical qubit. To find the logical operators we must find a sequence of Pauli X or Pauli Z, which commutes with all stabilizers but is not an element of the stabilizer group itself. A vertical string of Pauli X's can be seen to commute with all stabilizers it runs along, it has even overlaps with every Z stabilizer. The same thing holds for a horizontal string of Pauli Z's. Thus, the logical X is chosen to be a vertical string of Pauli X, i.e. $X_0X_3X_6$ and the logical Z as a horizontal string of Pauli Z, i.e. $Z_6Z_7Z_8$. To measure the parities of the data qubits, ancilla are placed on the place of the stabilizers. A total of 8 ancilla qubits are needed to measure all stabilizers, making the total amount of needed qubits 17 [16].

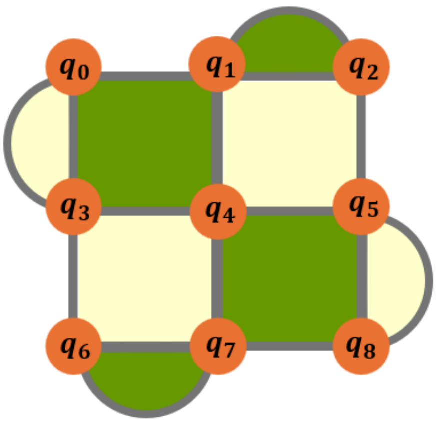


Figure 1.9: Distance-three 17-qubit surface code. Orange circles represent data qubits; Green square and triangle patches represent X stabilizers; yellow patches represent Z stabilizers. Ancilla qubits are placed at the middle of each face.

1.6. SMALL STELLATED DODECAHEDRON

The surface code has become a well-known quantum error correction code thanks to its simple use of a 2D square grid for the qubits. However, the surface code might not be optimal when protecting logical qubits against errors. Since it is a two-dimensional code based on a 2D grid, it must follow the Bravyi-Terhal-Poulin bound [17]. For k logical qubits encoded by n data qubits, it must hold that $kd^2 \leq cn$ for some constant c. Hyperbolic surface codes based on tilings of a hyperbolic surface are however not limited by this bound, they instead follow $\frac{k}{n} \geq c_1$, while the distance of the code can be upper bounded by $c_2 \log(n)$ for the constants c_1 and c_2. Thus these hyperbolic codes can have a more efficient ratio of logical to physical qubits, while their error-correcting distance (which measures how well they correct errors) grows logarithmically with the number of qubits. The downside of using these hyperbolic codes is that they need connections between qubits that aren't close together in a simple 2D space.

Looking at possible relatively small hyperbolic codes could lay a basis for the understanding of bigger hyperbolic codes, and successful results could foreshadow the possible success of other hyperbolic codes. To this end, we look at a regular dodecahedron, which due to its high symmetry also forms a relatively simple basis for applying high-order gates on its encoded logical qubits [18]. It has 12 faces, 20 vertices and 30 edges, see Figure 1.10. To construct a code from the dodecahedron, a qubit is placed on each of the edges. Z stabilizers act on each of the pentagrammic faces and thus have weight 5. A natural choice would seem to choose the weight 3 X stabilizers to act on each of the 20 vertices of the dodecahedron. The overlap of each X and Z stabilizer would be even, and we would have a valid stabilizer code. However, in order to ensure the code actually encodes logical qubits, all the edges are instead extended, creating new vertices where they meet. Instead of the weight 3 X stabilizers, the X stabilizers work on the 5 edges meeting at these newly created vertices. The created stellated dodecahedron now has 12 Z stabilizers and 12 X stabilizers, both having weight 5. It is visually easy to see that each X stabilizer has even overlap with each Z stabilizer. At each location of a stabilizer an ancilla qubit is placed, used to measure the stabilizer of the involved data qubits. As every qubit interacts with 2 X (or Z) stabilizers, resulting in $X^2 = I$, the product of all X stabilizers (or Z) is I. The 12th stabilizer is thus not linearly independent and thus imposes no extra restriction on the state space of the qubits. The amount of logical qubits encoded thus is $30-11-11=8$. In the rest of this thesis, the 12th stabilizers are still taken into consideration, even though it can be seen as a redundancy. Taking it into consideration does make the stellated dodecahedron graph more symmetric and easier to work with.

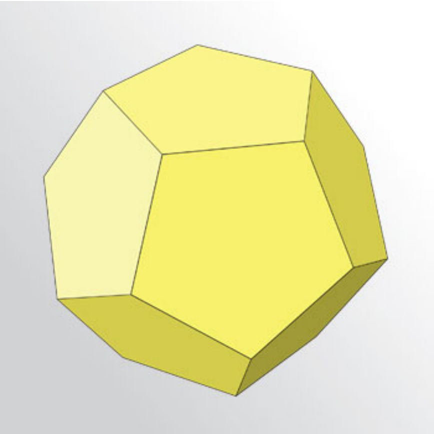


Figure 1.10: Dodecahedron [19].

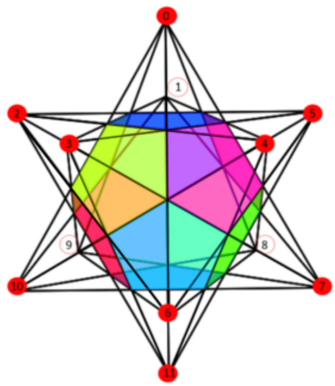


Figure 1.11: The small stellated dodecahedron [1].

A very helpful representation of the dodecahedron is its associated plane graph [20], see Figure 1.12. Here the qubits are still located at the middle of each edge. Each Z-stabilizer is acting on each face of the graph, each X-stabilizer is acting on all edges going into its respective face. The plane graph shows 11 faces, while we have defined 12 Z stabilizers, suggesting that 1 Z stabilizer is missing. The 12th Z stabilizer can be imagined as a face working on the outer 5 edges of the graph. In the same way, the 12th X stabilizer is working on the 5 edges pointing to the outer edges of the graph. The labeling of the qubits in this plane graph can be seen in Figure A.2.

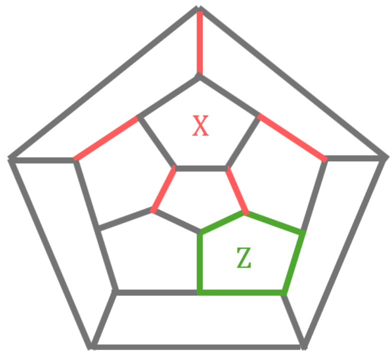
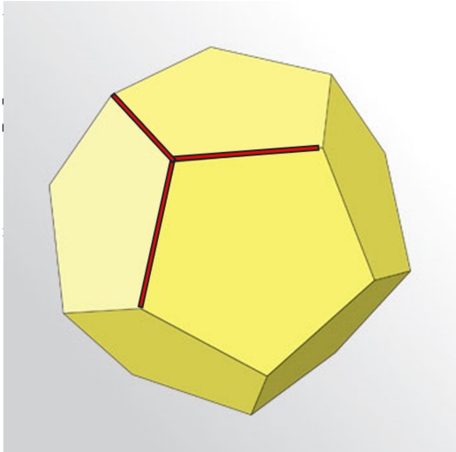


Figure 1.12: Plane graph representation of the small stellated dodecahedron, with 30 edges, 20 vertices, and 11 faces. Qubits are located at the middle of every edge. An X stabilizer has support on the edges shown in red, an Z stabilizer on the green edges. Likewise, for all other faces, there is a Z stabilizer working on the edges of the face and an X stabilizer working on the edges going into the face.

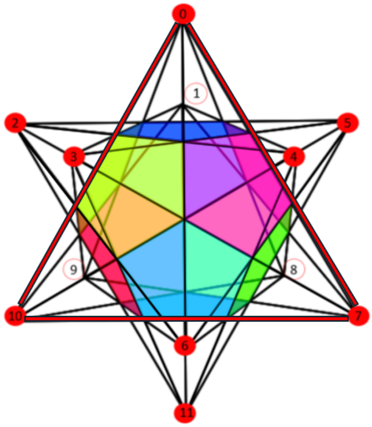
1.6.1. LOGICAL OPERATORS OF THE STELLATED DODECAHEDRON

The set of logical operators $\{\overline{X}_i, \overline{Z}_i\}, i = 1,2...k$ are Pauli operators that act on the encoded logical qubits. They are defined such that they commute with all stabilizers but do not commute with each other, they also are not part of the stabilizer group. So to find the X logical operators a string of X Pauli's has to be found that have an even amount of overlap with each Z. The reverse holds for the logical Z operators. It can be seen that each vertex of the dodecahedron forms a weight 3 logical X operator. This logical X always has an overlap of 2 with each Z stabilizer. I.e. (using the qubit notation as in Figure 1.11) $X_{(0,6)}X_{(2,4)}X_{(3,5)}$ has support on the qubits $(0,6)$ and $(3,5)$ and thus commutes with its neighboring Z stabilizer $Z_{(0,6)}Z_{(3,5)}Z_{(3,7)}Z_{(5,6)}Z_{(0,7)}$. Similarly, the logical Z operators can be found, corresponding to the triangular faces created by the stellation process. Again, these are weight 3, having even support on each X stabilizer. See Figure 1.13 for a visual representation and Table A.1 for a table containing all logical operators.

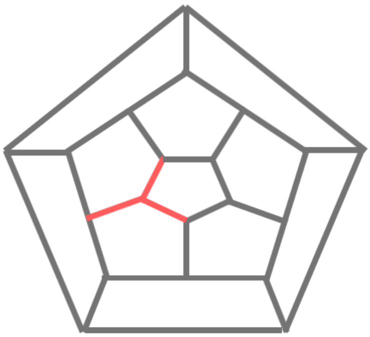
To check the linear dependency of the logical Z operators, observe that one vertex of the small stellated dodecahedron acts in 5 different logical Z operators. This vertex can be seen as the tip of each triangular face. By putting all 5 operators together, only the base of each one remains, as the other sides cancel out due to an even overlap. The 5 bases of the triangular faces however form a Z-check, suggesting that the 5 logical operators are not linearly independent as they together can be reduced to the identity, see Figure 1.14. That is, the 5 logical operators around vertex 0 together form the Z check on plane spanned by qubits $6-7-8-9-10$. Thus, instead of having 20 independent logical



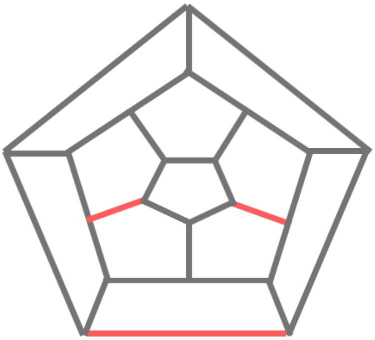
(a) Logical X operator of the SSD shown by the red egdes on a dodecahedron.



(b) Logical Z operator of the SSD shown by the red edges.



(c) Logical X shown in the plane graph representation of the stellated dodecahedron



(d) Logical Z shown in the plane graph representation of the stellated dodecahedron

Figure 1.13: Figure (a) shows a logical X operator in red acting on a vertex of the Dodecahedron. Figure (b) shows a logical Z operator working on a triangular face of the small stellated dodecahedron. Figures (c) and (d) show the same logical X and Z but instead work on the associatedplanar graph.

Z operators, one of the operators has to be removed for each vertex, leading to a total of 20-12 = 8 different logical Z operators. A similar argument can be made about the logical X operators, which also results in a total of 8 independent operators.

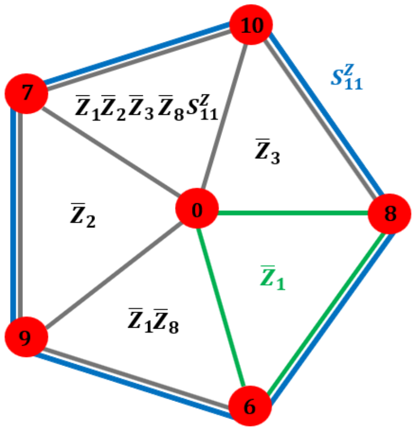


Figure 1.14: The S_{11} Z stabilizer is shown in blue. A single logical operator is shown in green. Combining 5 logical operators around a single central qubit forms a stabilizer, proving they are not linearly independent.

The lowest-weight logical operator has weight 3, thus the code's distance is 3. This implies that all errors affecting at most 2 qubits can be detected and that all single-qubit errors can be corrected. Figure 1.15 shows an example of this. Here 2 X errors have occurred on adjacent qubits, triggering not their common ancilla Z qubit as this commuting error will not result in the stabilizer measurement to detect an error. Instead, two adjacent ancilla qubits get triggered, suggesting a probable wrong correction when using a minimum weight decoder. Single qubit errors correctly trigger its two associated ancilla qubits, which suggest the right correction when using minimum weight perfect matching [21].

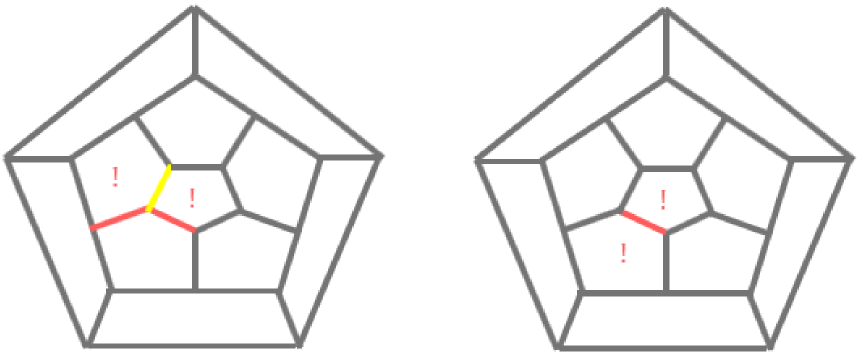


Figure 1.15: Plane graph representation of the small stellated dodecahedron. Left shows two X errors occurring on adjacent qubits. This results in the triggering of two ancilla qubits, which for minimum weight matching ends up with the correction of the wrong qubit. Right shows a single X error, also triggering two ancilla qubits, but this time ending up with the right correction.

1.6.2. MEASURING THE STABILIZERS OF THE SSD CODE

Research has already been done on this code, yielding results showcasing some potential [1], but high logical error rates. Unfortunately, the code seemed to get outperformed by the surface code, undermining possible usage. A probable explanation for the less-than-expected performance can be found in the stabilizer measurement order. The sequential chosen measurement schedule in Table A.6 first performed all X stabilizer measurements before performing Z stabilizer measurements. As each X and Z ancilla has support on 5 qubits, the total amount of CNOT layers needed to perform the whole circuit would be $T = 1 + 10 + 1 = 12$, one layer for initializing all ancilla qubits, 10 CNOT layers for measuring all data qubits and finally 1 layer of to measure the ancilla qubits. As each layer represents a time step, only 12 of the 30 data qubits are measured at each timestep and there is a lot of idling amongst the nonactive qubits, making them susceptible to errors. The main goal of this project is to try and reduce the time steps needed to perform all stabilizer measurements by finding an interleaved schedule where X and Z stabilizer measurements are performed simultaneously. This problem will be tackled in Chapter 2 and Chapter 3 by developing an augmented edge coloring algorithm. The developed algorithm will be tested on different codes in Chapter 4. In Chapter 5, the focus will be on verifying the fault tolerance of the newly found schedules for the SSD. This analysis will look into the possible spreading of single errors. In Chapter 6, the logical error rates associated with these interleaved schedules will be examined, comparing their performance to a sequential schedule and 8 pieces of the distance-3 surface code.

2

Coloring the Small Stellated Dodecahedron

The small stellated dodecahedron code has 30 data qubits on its edges, 12 X ancilla qubits in the middle of each face, representing the X stabilizer acting on the edges going into the respective face, and 12 Z ancilla qubits located on its faces acting on the associated edges. A Tanner graph [22] can also represent this, Figure 2.1, with 24 nodes representing the set of ancilla qubits and 30 nodes representing the data qubits. Each edge between the ancilla qubits and data qubits represents a CNOT between the data qubit and the ancilla qubit. Each Z and X ancilla qubit has support on 5 data qubits and thus the graph has a total of 120 edges.

In this section, we assume that X-checks and Z-checks are measured through the interaction of a single ancilla qubit with a data qubit using a CNOT gate. At any given time, an ancilla qubit can only interact with one data qubit via a CNOT, and similarly, any data qubit can only interact with one ancilla qubit. A key problem addressed in this chapter is finding a schedule for these CNOT gates that minimizes the number of time steps required to measure all parity checks.

This scheduling problem can be formulated as a minimal edge coloring problem on the Tanner graph of the SSD 2.1, where each edge represents a CNOT interaction between an ancilla and a data qubit. The goal is to color the edges such that no two edges incident on the same vertex share the same color, with each color corresponding to a different time step for the interaction [23]. Throughout the subsequent chapters, the qubit labeling from Figure A.2 will be used to identify and distinguish the qubits.

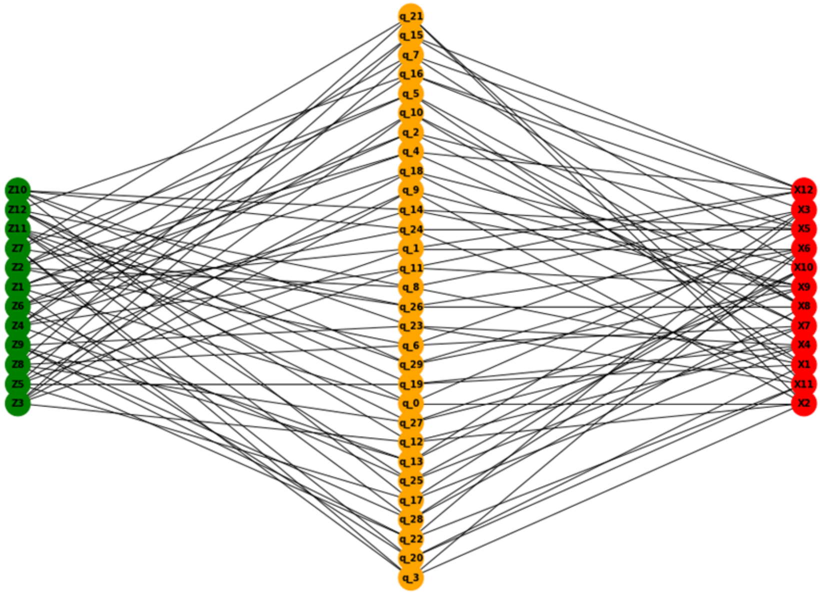


Figure 2.1: A Tanner graph of the small stellated dodecahedron code. On the left, there are 12 Z-ancilla qubits, shown in green. Right, there are 12 X-ancilla qubits, shown in red. Each ancilla qubit has 5 edges going to 5 different of the 30 yellow data qubits. The labeling from Figure A.2 is used.

2.1. LENGTH 5 COLORING

The goal is to find an edge coloring of the SSD Tanner graph.To find a coloring for the bipartite graph of the SSD, König's Line Coloring Theorem [24] says that,

Theorem 1 *For a regular bipartite graph G, where each vertex has degree d, the minimum number of colors $\Delta(G)$ needed to color the edges of G such that no two adjacent edges share the same color is exactly d.*

The Tanner graph G of the small stellated dodecahedron is not regular, as the nodes do not all have the same degree, meaning they are connected to different numbers of edges. The X-checks (and Z-checks) are connected to 5 data qubits, but each data qubit is only connected to 4 checks. In a regular graph, every node must have the same amount of edges, so the procedure described by Erdös in [25] is followed to make a general graph regular, which consists of:

Let n denote the order of the graph G and let d denote its maximum degree. We search a minimal set I of m points such that by adding edges between nodes in I or between nodes in I and G we obtain a regular graph bipartite H. Suppose H has been constructed from G and has order $n+m$. We denote the points in G to be $u_1, u_2, \ldots u_n$ and the points from I to be $v_1, v_2, \ldots v_m$. We let the degree of node v_i be d_i and the deficiency of each node in G is $e_i = d - d_i$. The deficiency of each node is the needed amount of edges to increase its degree to the degree d. At last we take $s = \sum_i e_i$ and $e = \max(e_i)$.

Then for the graph H, there are precisely s lines joining points from I to G. Since every point in I is connected to at most d points in G, it follows that

$$md \geq s \tag{2.1}$$

That is, in order to make our graph G regular, s lines are needed from I to points in G, and surely if every node from I is connected to d, this should be bigger than s.

Clearly, it should also be that,

$$m \geq e \tag{2.2}$$

Finally, the degree of every graph is even as every edge contributes 2 degrees. Thus

$$(m+n)d = 0 \bmod 2 \tag{2.3}$$

Applying the above equation to the Tanner graph of the SSD says that $m \cdot 5 \geq 30$, that $m \geq 1$ and that $(m+54)\cdot 5$ must be even. This boils down to $m = 6$, and thus the graph can be made regular by adding 6 nodes, each with 5 edges going to the qubits, see 2.2. As the Tanner graph of the SSD is now regular, König's theorem states that the graph can be colored using 5 colors.

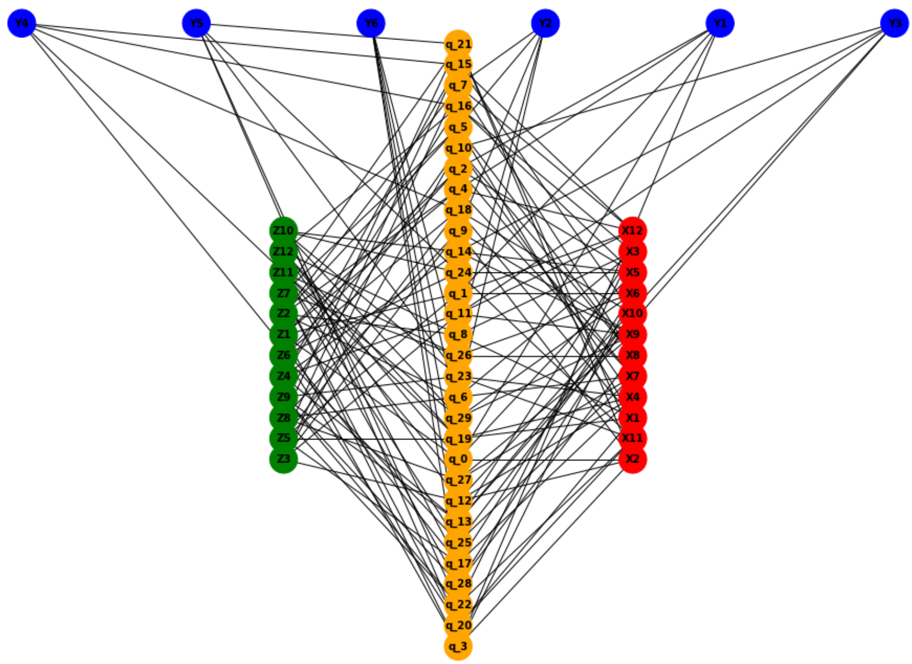


Figure 2.2: A graph of the small stellated dodecahedron. It has been extended with 6 blue points to become regular and bipartite. Each blue point goes to 5 different data qubits.

A numerical solution for the coloring of the small stellated dodecahedron using 5 colors has been found. This numeric solution was found by using a simple coloring algorithm, which checks for each edge if it can be colored, and if so assigns a certain color. The code can be found at the GitHub page B. Figure 2.3 shows the plane graph of the SSD with colors indicating at which timestep each qubit interacts via CNOT with the associated Z ancilla qubit lying in the middle of each face. Figure 2.3 also shows the plane graph of the SSD, this time showing at which time step each qubit interacts with their X-stabilizers. To check that this is a valid coloring, we notice that all Z and X-stabilizers have 5 different colors. By overlapping the graphs for the Z and X-stabilizers, it becomes clear that each qubit also has 4 different colors and that this is indeed a valid coloring. The schedule can be found in Table A.2

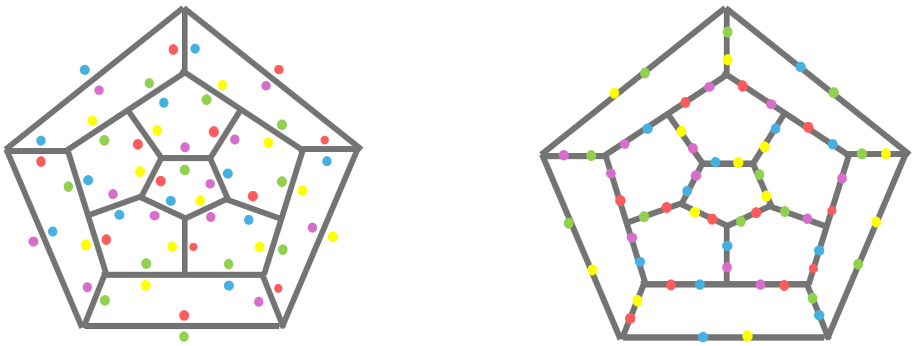


Figure 2.3: A 5-color matching of the SSD. Different colored dots represent the interaction of a qubit with its associated ancilla qubit at a certain time step. Left shows the interaction of the data qubits with the Z ancilla qubits, right shows the interaction with the X ancilla qubits.

2.2. ON PROPERNESS OF SYNDROME EXTRACTION SCHEDULES

Operators and parities of multiple qubits can be measured via schedules given in Chapter 1. However, problems can occur when multiple data qubits are measured in sequence concerning the same ancilla qubit. For this, look back at Figure 1.6, where stabilizer measurements were carried out on two data qubits using 2 ancilla qubits. The order in which each data qubit interacted with the ancilla qubits might have seemed arbitrary, but looking at schedules with different orderings makes it clear that the order does influence the outcome of the measurements. A proper measurement schedule should give the correct measurement outcomes and checking this for the schedule used in Figure 2.4, will show that this is indeed the case:

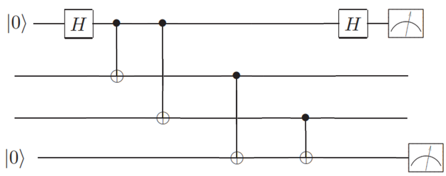


Figure 2.4: Proper measurement schedule for measuring 2 qubits using 2 ancilla qubits. The order is $X_1 - X_2 - Z_1 - Z_2$

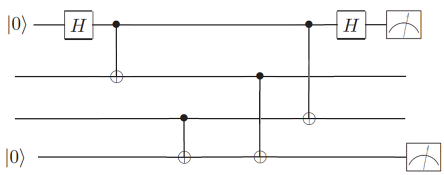


Figure 2.5: Improper measurement schedule for measuring 2 qubits using 2 ancilla qubits. The order is $X_1 - Z_2 - Z_1 - X_2$

$$\begin{aligned} |\psi\rangle &= |0\,a\,b\,0\rangle + |1\,a\,b\,0\rangle \\ &\xrightarrow{CNOT's} |0\,a\,b\,a+b\rangle + |1\,\overline{a}\,\overline{b}\,a+b\rangle \end{aligned} \tag{2.4}$$

But that for the improper circuit in Figure 2.5,

$$\begin{aligned} |\psi\rangle &= |0\,a\,b\,0\rangle + |1\,a\,b\,0\rangle \\ &\xrightarrow{CNOT's} |0\,a\,b\,a+b\rangle + |1\,\overline{a}\,\overline{b}\,\overline{a}+b\rangle \end{aligned} \tag{2.5}$$

Where an overline denotes the negation of qubit a or b. The resultant state for both circuits is nearly the same, except for the last qubit having an $\overline{a}$ instead of a normal a. What has happened is that in the improper circuit, the final X measurement has become dependent on the X measurement of our Z-ancilla qubit, resulting in that for any general state, the measurement outcomes will be random. For example, assuming that the qubits a and b are both prepared in $|0\rangle$. Then the improper circuit will result in

$$\begin{aligned} |\psi\rangle &= |0\,0\,0\,0\rangle + |1\,0\,0\,0\rangle \\ &\rightarrow |0\,0\,0\,0\rangle + |1\,1\,1\,1\rangle \end{aligned} \tag{2.6}$$

If we try to measure the Z-ancilla qubit we notice that we get a random outcome, even though we would expect to measure $|0\rangle$ as both data qubits are prepared in $|0\rangle$. When we look at the proper circuit we see we end up with the state $|\psi\rangle = |0000\rangle + |1110\rangle$ showing that measuring the Z-ancilla will always correctly return $|0\rangle$.

2.2.1. X AND Z PROPAGATION THROUGH CNOT

Looking at the propagation of a Pauli X through a CNOT, it becomes clear why certain schedules are proper or improper. Starting with an arbitrary state $|\psi\rangle = |C\rangle \otimes |T\rangle$, with $|C\rangle$ the control qubit and $|T\rangle$ the target qubit, then by first applying an X-gate to the control qubit, the state changes into $|\psi\rangle = (X \otimes I)(|C\rangle \otimes |T\rangle)$ and after applying a CNOT $|\psi\rangle = \text{CNOT}(X \otimes I)(|C\rangle \otimes |T\rangle)$. However, this state can also be written as

$$\text{CNOT}(X \otimes I) = (X \otimes X)\text{CNOT}, \tag{2.7}$$

(see Section A.2),implying that the Pauli X has propagated 'down' through the CNOT, and now also acts on the target qubit. If the Pauli X initially worked on the target qubit

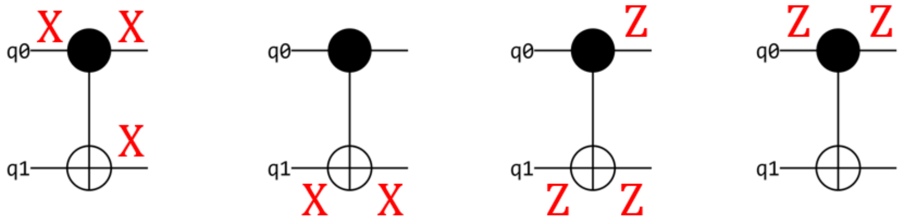


Figure 2.6: The left 2 figures show the propagation of Pauli X through a CNOT. It can propagate downwards through the control qubit, but cannot propagate upwards from the target qubit. The right 2 figures show the propagation for Pauli Z, and the fact it propagates upwards yet not downwards.

instead of the control qubit, we can see that the Pauli X does not propagate upwards through the CNOT. This can also be seen in Figure 2.6.

The same reasoning holds for the propagation of Pauli Z, except the propagation is exactly the opposite. That is, instead of Z errors propagation downwards through a CNOT, they instead propagate upwards, following CNOT$(I \otimes Z) = (Z \otimes Z)$CNOT and CNOT$(Z \otimes I) = (Z \otimes I)$CNOT [26]. To make sure no Pauli X and Z propagate onto the ancilla qubits and randomize the measurement outcomes, the following theorem is used

Theorem 2 *For each pair of overlapping X and Z-checks, $S_j(X), S_k(Z)$, let $Q = \{q_1, q_2, ..., q_{2n}\}$ be the qubits in the overlap. For a qubit q_i which is acted upon by checks $S_j(X)$ and $S_k(Z)$, let $S_j(X) < S_k(Z)$ denote an order where q_i first interacts with the X-ancilla of $S_j(X)$ and afterwards with the Z-ancilla of $S_k(Z)$. In a proper schedule for any pair of overlapping checks $S_j(X), S_k(Z)$, an even number of $q_i \in Q$ must have $S_j(X) < S_k(Z)$.*

To see why this is true, look back at Figure 2.6 where a Pauli X propagates through the control of a CNOT. If an even number of these Paulis come together, they will pairwise annihilate as $X^n = I$, for even n (or $Z^n = I$). If an even number of qubits first interact with the X-ancilla, all the X propagations will cancel, resulting in a proper measurement schedule. The same holds for the Z Pauli on the X ancilla qubit.

2.3. PAIRS OF QUBITS FOR THE SSD

In the case of the small stellated dodecahedron, start by looking at a single Z-check from its plane graph. As each Z stabilizer is associated with a face and each X stabilizer with a vertex, they overlap on either 2 edges (so 2 qubits) or none. See Figure 1.12. Theorem 2 thus reduces to the case where only $q_1, q_2 \in Q$. To have an even number of q_i with $S_j(X) < S_k(Z)$, either both qubits first interact with the X ancilla, and then the Z ancilla, or the reverse where both qubits first interact with Z.

Theorem 2 makes the edge coloring with 5 colors as seen in Figure 2.3, not an actual viable measurement schedule as it does not follow the proper ordering. It can be seen that for overlapping X and Z checks the qubits do not always start by interacting with the same ancilla qubit. It is obvious that a sequential measurement schedule, where first all X stabilizers are measured and subsequently all Z stabilizers, satisfies the properness

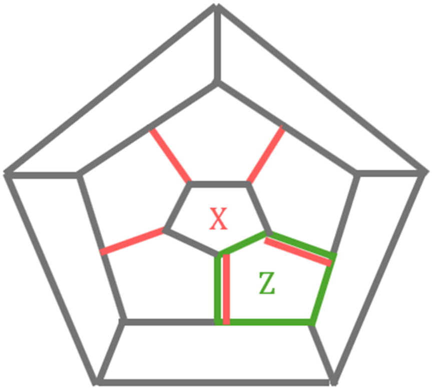


Figure 2.7: A single Z ancilla qubit is shown in green, working on the surrounding 5 edges. An X ancilla qubit works on the edges shown in red, making it so these two stabilizers form an overlapping pair.

condition. However, as described in [1], this takes an unnecessary amount of extra time steps. To find an interleaved measurement schedule, the edge coloring algorithm has to be adapted to take into account the constraints coming from the properness condition.

3

Algorithm for Finding Interleaved Schedules

The goal is to create an algorithm that finds a proper measurement schedule, using the least amount of CNOT layers. One starts by finding all overlapping groups of X and Z-stabilizers. As for the SSD code each q_i interacts with 2 Z ancilla qubits and is in 2 different pairs per Z ancilla qubit, see Figure 2.7, and there are 4 different overlapping groups per qubit. In each of these groups, we get 2 edges corresponding to the X and Z checks of q_i and 2 edges corresponding to the X and Z checks of q_k in the group. That is, an overlapping pair X-Z group always looks like Figure 3.1. Here nodes represent qubits and ancilla qubits, the edges represent the possibility of a qubit to interact with the respective ancilla qubit.

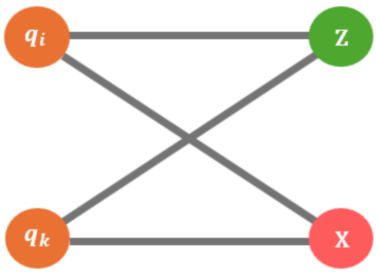


Figure 3.1: An overlapping group containing 2 qubits q_i, q_k, which both interact with the same X and Z ancilla qubit.

In total, there are 60 different groups, see Table A.3 and Table A.4. By going over all the overlapping groups, we count how many edges have been colored. There are in total 5 different cases, corresponding to the coloring of $\{0,1,2,3,4\}$ edges. As each case implies different restrictions on the rest of the pair due to Theorem 2, the goal of this section is to state the restrictions each case puts on the remaining edges and which edges have to temporarily be blocked from being colored. The algorithm works by iteratively applying these restrictions to the graph, performing one single coloring round, removing the edges that got colored, and then updating said restrictions for the newly obtained graph. First, for each amount of colored edges, the restrictions will be worked out, which edges can be colored, and which edges have to be blocked. To conclude, the restrictions on each group put every edge into one of four categories,

1. Normal edge: can be colored by the upcoming coloring round. In the following chapter, it will be represented as a grey edge.

2. Blocked edge: based on the restrictions cannot get colored in the upcoming coloring round and gets temporarily removed from the graph. It only gets made available once its release edge has been colored. It will be represented by a red edge.

3. Release edge: once it gets colored unblocks a blocked edge. It will be represented by a yellow border surrounding an edge.

4. Colored edge: has been colored in a previous round and thus gets permanently removed from the graph in all subsequent rounds. It will be represented by a green edge.

The complete algorithm will take a loop-like structure which can be seen in Figure 3.2. First, the restrictions will be checked and certain edges will get blocked. The graph is then converted into a line graph and a maximal independent set is chosen. This maximal independent set gets assigned a color and gets removed from the graph. Then, based on which edges got colored the restrictions will be updated and new edges will be blocked or released.

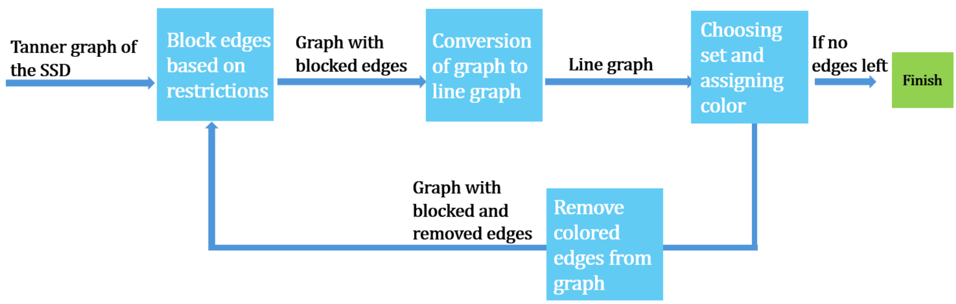


Figure 3.2: A schematic diagram of the algorithm. The algorithm starts with the whole SSD Tanner graph. Based on the restrictions certain edges from this graph will get blocked and temporarily removed from the graph. The graph then gets converted into a linegraph and a maximal independent set gets chosen. This independent set gets assigned a color and gets permanently removed from the graph. This process is performed until there are no edges left in the graph.

3.1. RESTRICTIONS ON THE GROUP BASED ON THE NUMBER OF EDGES COLORED

3.1.1. 0 EDGES HAVE BEEN COLORED

When zero edges in the group have been colored, there are four edges available to be colored. However, due to the constraint that qubits and ancilla qubits can only interact with 1 CNOT at a time, adjacent edges in the group also cannot get the same color. It can also be seen that nonadjacent edges can not get the same color, as in this case both qubits from the pair interact with a different ancilla qubit first. That is, if the edge (q_k, X) gets color 1, then (q_i, Z) cannot get color 1, as this would violate the restriction that both qubits should start interacting with the same ancilla. A group with no colored edges thus results in a group where only 1 edge can be given the upcoming color.

3.1.2. 1 EDGE HAS BEEN COLORED

Once one edge in the group has been colored, the non-adjacent edge to the colored edge should temporarily be blocked to get colored. The interaction between the respective qubit and ancilla qubit can only take place once the qubit has first interacted with the right ancilla qubit. That is, when (q_k, X) gets the first color in the group, (q_i, Z) should be blocked.(q_i, Z) can only be put back once its release edge, (q_i, X) has been colored. This is to sustain that both qubits first interact with the same ancilla qubit. All other edges stay the same and, unless blocked, can be colored.

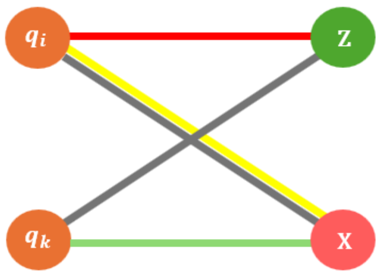


Figure 3.3: A pair of qubits, if the green edge gets colored then the red edge has to be removed. The yellow highlighted edge is the release of the red edge.

3.1.3. 2 EDGES HAVE BEEN COLORED

There are two different scenarios in which two edges of a group have been colored. Suppose (q_k, X) has been colored first, and (q_k, Z) has been colored second. Now one qubit q_k has performed both interactions with both ancilla qubits, however the other qubit q_i has still to interact with X first, leading us to put no new restrictions on the group. It can however also be the case that (q_k, X) has been colored first, and (q_i, X) has been colored second. In this case, both qubits have interacted with the same ancilla qubit, enabling them both to also interact with the other ancilla qubit. All edges from the group which were made blocked from being colored, can now potentially be made available to get colored again. However, each edge is in two groups, group 1 could indicate that the edge can be made available again, whilst group 2 does not allow the edge to be put back. For every blocked edge that could be put back according to group 1, it also needs to be checked whether it can be put back according to group 2. Suppose the edge (q_i, Z) can be put back due to group 1. Then there are 4 cases of colored edges in group 2 such that it is allowed to make (q_i, Z) available again.

1. Group 2 has only 1 colored edge which is not the non-adjacent edge of (q_i, Z).
2. Group 2 has 2 colored edges, both interacting with the same ancilla qubit.
3. Group 2 has 2 colored edges. The edge which got colored first is not the non-adjacent edge of (q_i, Z).
4. Group 2 has 3 colored edges

3.1.4. 3 OR MORE EDGES HAVE BEEN COLORED

Once 3 colors have been colored there are 2 possible scenarios left. Either the last remaining edge has been blocked, or the edge has not been blocked and can be colored. There is nothing to do for this overlapping pair group but wait until the last remaining edge gets put back by the other group it is in, following the same 4 criteria as in Subsection 3.1.3.

If 4 edges of a group are colored, there is nothing left to do.

3.2. LINE GRAPH CONVERSION AND COLORING THE EDGES

Every group has been checked and it has been determined for its 4 edges in which category they fall (normal, blocked, release, removed). Now the second step of the algorithm is to change each of the groups into a line graph [27], where each edge becomes a node and each node becomes an edge. In Subsection 3.1.1 it has been explained that for a pair group where no edges have been (temporarily) removed, only 1 edge can be colored with the upcoming color. To capture this constraint on groups with 4 available edges, 2 extra edges are added between non-connected nodes, see Figure 3.4.

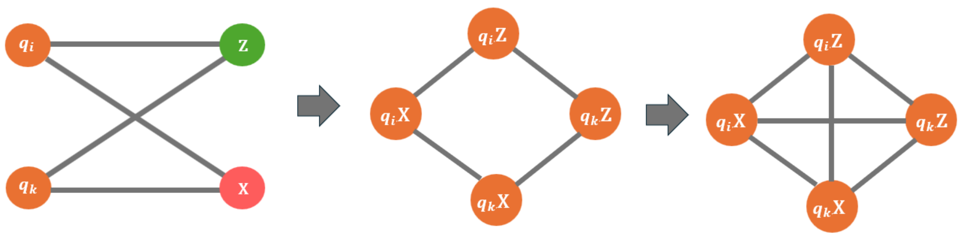


Figure 3.4: Conversion of a qubit pair into a linegraph. Nodes become edges and edges become nodes. 2 additional edges are added to capture the restriction from 3.1.1

It is however also a possibility that no edge has been matched in a group, but edges have been blocked, and thus have temporarily been removed, thanks to their non-adjacent edges in their other group getting colored. In this case, the pair has less than 4 edges, and the resulting line graph has less than 4 nodes. The algorithm in these cases detects that the converted pair group has less than 4 nodes, and adds no extra edges, see Figure 3.5 showcasing an example.

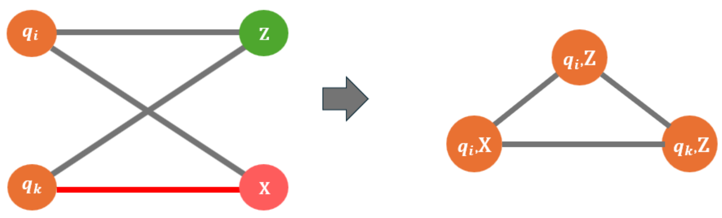


Figure 3.5: If one edge of the qubit pair has been removed, the algorithm converts the pair into a linegraph taking into account that one edge is missing.

Once every overlapping pair group has been changed into a line graph, a maximal independent set algorithm used from NetworkX [28] finds a maximal independent set based on a random seed. A set S of vertices is called independent when it is chosen in a way such that for every two vertices in S, there is no edge connecting the two. Such a set S is called a maximal independent set when there are no vertices out of S that may join it [29]. In the worst case, the used maximal independent set algorithm returns a set of size $O(\frac{|V|}{(log|V|)^2})$ for a graph with $|V|$ vertices. The small stellated dodecahedron graph has 54 nodes and 120 edges, by converting it into a line graph we thus end up with $V = 120$, and in a worst-case scenario a coloring matching of size $\frac{120}{log(120)^2} = 6$. In the best case, the found maximal set is of size 24, as then every X and Z-check is matched to 1 qubit. Because the used graph is highly symmetric, color matchings of such low size do luckily not occur. Running the algorithm for 5 minutes returns 10.260 possible first matches, averaging a length of 21 vertices, although returning merely 37 cases of a length 24.

After a maximal independent set has been chosen, the edges from this set get removed from the original graph and assigned their associated color. For the new resulting graph, the restrictions again are checked as described before, blocking edges when needed. The new updated graph again gets converted to a line graph and a new maximal independent gets chosen and assigned a new color. This process gets iteratively repeated until there a no edges left in the graph. The algorithm can be found at the GitHub page B.

3.3. IMPROVING THE ALGORITHM

A problem with the described algorithm is that it terminates almost half of the time because the eventually found parity check schedule cannot add any more edges. As every qubit interacts with 2 Z-stabilizers and 2 X-stabilizers, they can locally be viewed as lying on a square lattice, with their corresponding pair qubits lying on the corners, see Figure 3.6.

Section 3.1.2 showed that properness imposes that the first colored edge in a pair

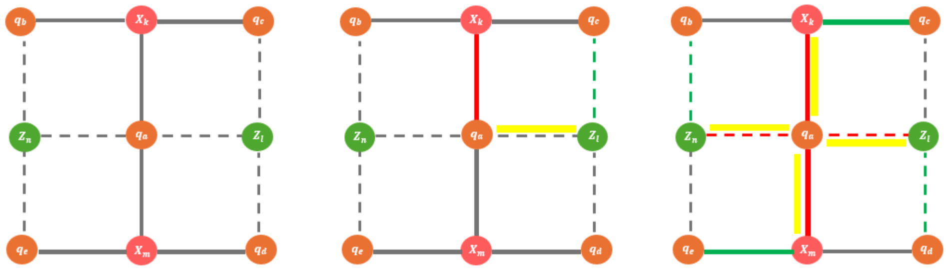


Figure 3.6: Each qubit is in 4 pair groups and can thus be shown as laying on a square grid (Left). A single edge (q_j, Z_j) has been colored (green), resulting in the removal of a single other edge (q_i, X_j). This removed edge can get put back once the edge (q_i, Z_j) has been colored (Middle). A cycle of removed edges that cannot be broken up as each removed edge forms the release for another (Right).

results in the non-adjacent edge being blocked. This temporarily removed edge can be released and assigned a color once its release edge has been colored. An example of a blocked edge can be seen in Figure 3.3. Here the edge (q_j, Z_j) is the first edge to be colored in the pair between q_j and q_i, and as a result the edge (q_i, X_j) has to be temporarily removed. Now the (q_i, X_j) can only be added back once (q_i, Z_j) has been colored, as this way both qubits in the pair $[q_i, q_j]$ have both first been matched with Z, and subsequently with X.

Problems occur however when the edge(s), which form the release of blocked edges, are removed in a loop-like structure, see Figure 3.6. In this case, four edges get removed from one qubit, by the coloring of one nonadjacent edge from each pair. Explicitly, the temporarily removed edge (q_i, X_j) can be released when (q_i, Z_j) has been colored. (q_i, Z_j) itself however has also been removed and can be put back once (q_i, X_i) has been colored. Following this line of removed edges which form the release for a former edge, we eventually end up at the first removed edge (q_i, Z_j) forming the release for the edge (q_i, X_j). A loop structure thus results in a blocking set of edges that can never be released and results in the failure of the algorithm. Bigger loops, where multiple qubits get surrounded by a loop of first colored edges in pairs, pose no problem for the qubits as there will always be an edge that has not been temporarily removed for each qubit.

Thus after each coloring round the algorithm checks if no such blocking sets have occurred in the coloring. This is done by finding all the 4 pair groups associated with a single qubit q_i. Once each group has at least 1 edge colored, it is checked whether these edges do not act on the center qubit q_i and none of the edges share the same X or Z stabilizer. If these requirements are met, the algorithm will stop and start a new attempt to find another schedule. The updated algorithm either terminates due to a loop forming or due to the found schedule being longer than the desired length. As the algorithm uses a random seed the run times tend to fluctuate heavily. The improved algorithm seems to be 3 times faster, having a runtime of $(7.7 \pm 5.4)10^2$s versus the runtime of the old algorithm $(2.2 \pm 2.0)10^3$s.

3

3.4. LENGTH 6 PROPER COLORING

The improved algorithm was run multiple times, returning a total of 18 seemingly different measurement schedules. No conclusive answer can be given as to whether these 18 schedules are all possible interleaved schedules. It is also not known whether new schedules could have better performance when tested as in Chapter 5. All 18 found schedules were of length 6, having 1 color more than the normal coloring of the Tanner graph as in Figure 2.1. An example of the found measurement schedules can be seen in Figure 3.7 which is based on the schedule in Table A.7. It can be verified that the qubits from each qubit pair either first interact with their X stabilizer or their Z stabilizer.

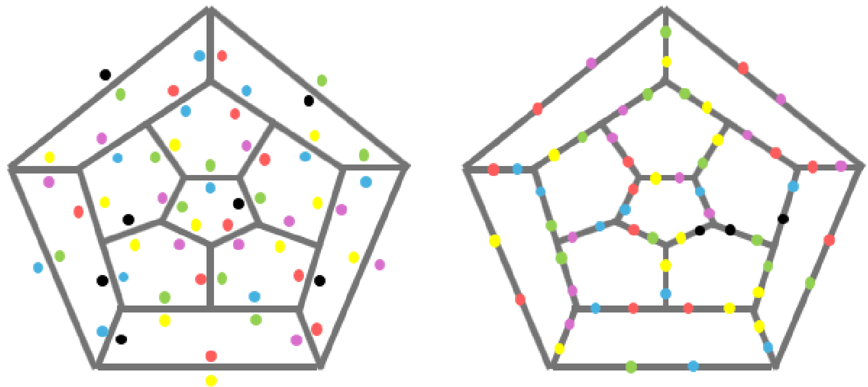


Figure 3.7: Interleaved proper schedule of length 6, shown in a planar graph of the SSD. The left figure shows the Z stabilizers, and the right figure the X stabilizers. The order of colors is Red-Green-Yellow-Blue-Purple-Black. Overlapping of both figures shows that no qubit has the same color twice, but now also that qubits that form a pair also follow the properness condition.

In order to understand the resultant schedules, some work was put into finding a pattern within the interleaved schedules. This was done to get a grasp of why the interleaved schedules work, possibly giving insight into how to construct a length 5 schedule or how to manually construct new length 6 schedules. Although there seemed to be some periodicity, where multiple Z stabilizers would take the same pattern in measuring the 5 qubits, no conclusive pattern could be found.

3.5. LENGTH 5 PROPER COLORING?

The fact that the normal bipartite graph of the dodecahedron could be colored in 5 colors raises the question of whether this would also be possible when taking the properness condition into account. The numerical results found by running the algorithm do not give immediate insight if this could be possible. To this end, the algorithm was run for a substantial amount of time, visiting approximately 2592000 different graph colorings. Yet no 5-length coloring was found. Due to the permutations of possible colorings, it is hard to rule out the possible existence of a length 5 coloring, although a sophisticated brute force method could be applied. Instead, attempts were made to try and find a proof or sketch of proof for the existence of a length 5 coloring. Several steps have been made, yet no conclusive answer has been found.

3.5.1. THE CREATION OF 6TH COLOR

The normal coloring of the Tanner graph of the SSD used 5 colors. A 6th color occurs once two qubits in a pair get the colors 1 and 5 with respect to the same measurement (both X or both Z). See Figure 3.8, as 1 is the lowest color, q_i must have a color >1 with X, and q_k must have a color >5 with X and thus a 6th color is needed.

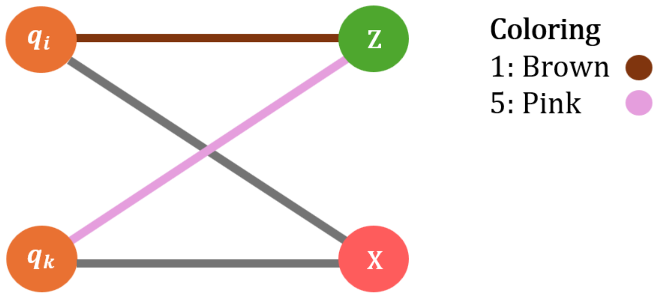


Figure 3.8: A qubit pair. If the edges connected to the Z (or X) ancilla qubit get colored 1 and 5, then a 6th round will be made.

In the representation of the plane graph of the dodecahedron, we get that for each face, corresponding to a Z-check, the edges with colors 1 and 5 must be adjacent to one another. Otherwise, the edges would be in the same pair, resulting in needing a 6th color, see Figure 3.8 For the X stabilizers, the edges going into the face should be non-adjacent for colors 1 and 5.

In order to try and rule out the possibility of a proper 5-coloring of the SSD, a brute force method was used to find whether a coloring with every color 1 and color 5 following Figure 3.9. which can be found on the GitHub B. All edges were grouped per Z and X check, and one Z or X check was chosen as the starting edge. This edge got assigned a permutation of (1,2,3,4,5), and based on this starting point other permutations were chosen for the remaining checks. Due to the scale of the problem, the estimated total runtime was $O(120^{24})$, but a solution was found after merely 50s, which can be seen in Table A.5.

3.5.2. REDUCING THE GRAPH

Multiple different reductions have been made to the graph to try and see which restrictions or combinations of restrictions lead to a length-6 coloring. Reductions include

1. Taking only 1 Z stabilizer with 5 X stabilizers acting on its edges

2. Taking a central Z stabilizer, surrounding it with 5 Z stabilizers

3. Leaving out a random number of stabilizers

yet every reduction resulted in the development of a length-5 schedule, showing no hints as to why the 6th color was created. Leaving out the 12th X and Z stabilizers, resulted in a length 6 coloring, following the fact that they are to some degree redundant and pose no new restrictions on the graph. It seems that only the whole collection of restrictions leads to the creation of a length-6 schedule, and reducing the graph does not give more insight into why this happens.

3.5.3. ORDERING FOR THE SSD

In the literature [16], different codes such as the surface code, utilize a specific ordering for measuring X and Z checks. This ordering is determined by the orientation of the surface, where a vector orthogonal to the surface is used and the right-hand rule is applied to establish the sequence of measurements (e.g., North-West-East-South). However, this ordering does not directly translate to hyperbolic surface codes due to the non-trivial parallel transport of a vector around a closed curve, which prevents it from returning to its original configuration [30]. If we attempt to impose a periodic ordering on the SSD, we could begin by coloring one Z-check. However, due to the different returning paths along the dodecahedron, we would end up with inconsistencies in the orientation of the coloring on the initial face. This suggests that achieving a ordered coloring of length 5 may not be feasible. Nonetheless, the question of whether the entire graph can be properly colored using 5 colors remains open.

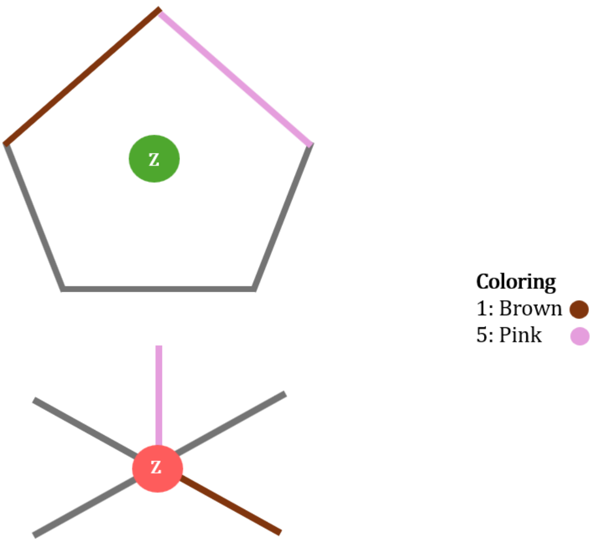


Figure 3.9: If the edges with color 1 and 5 are next to one another in a Z stabilizer, a 6th color will not be created. The edges must be non adjacent for a X stabilizer.

4

Applying the algorithm to other codes

The developed algorithm can produce interleaved schedules for the SSD whilst taking the properness condition into account. The algorithm was built on the overlapping groups existing out of 4 edges, see Figure 3.1, and the fact that each edge was present in a total of 2 overlapping groups. These characteristics are such an intricate part of the algorithm, that the algorithm cannot be used for codes having bigger overlapping groups. This chapter explores the application of this algorithm to various quantum error correction codes that share the same overlapping group structure as the SSD. We will begin by examining the distance-3 surface code (see Section 1.5), followed by the introduction of a code based on the Tetrahemihexahedron polyhedron. All Tanner graphs of the codes can be found at the GitHub B.

4.1. Distance-3 surface code

The distance-3 surface code uses a square tiling, putting 9 qubits on each of the vertices, see Figure 4.1. X and Z stabilizers are plaquettes working on the surrounding qubits, and logical X and Z operators are respectively vertical and horizontal strings of Pauli's. Just as for the SSD, the goal is to find an interleaved proper schedule by coloring the Tanner graph. The overlapping pair groups can be seen to again have size 4, and each edge is present in at most 2 overlapping groups, allowing the algorithm described in Chapter 2 to be used.

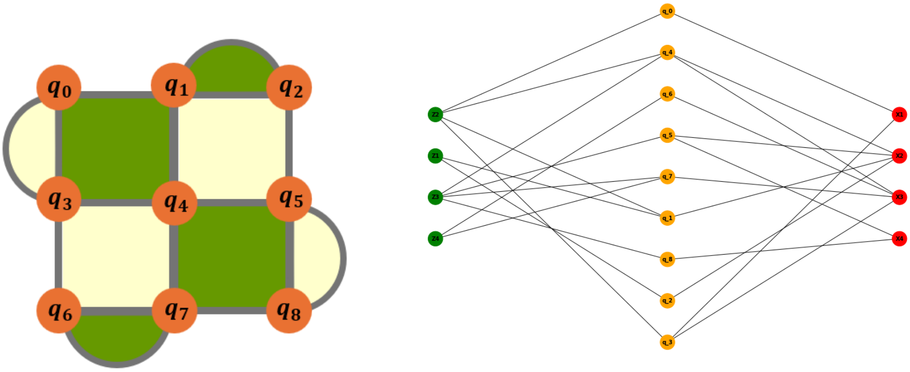


Figure 4.1: Left figure shows a distance-3 surface code, see Section 1.5. The right figure shows the Tanner graph of the surface code. Green nodes represent X-checks, orange nodes represent data qubits and the red nodes represent Z-checks.

The algorithm will return an interleaved schedule of length 4, see Table A.10, in correspondence to the schedule found in [11]. It can be seen that the schedule takes a certain ordering, where each X and Z stabilizer takes a Z-shape. To check that the found schedule is proper, take two qubits with overlapping X and Z stabilizers, i.e. q_1 and q_4. Then both qubits first interact with the X stabilizer before interacting with the Z stabilizer. This measurement schedule will be used to benchmark the SSD code.

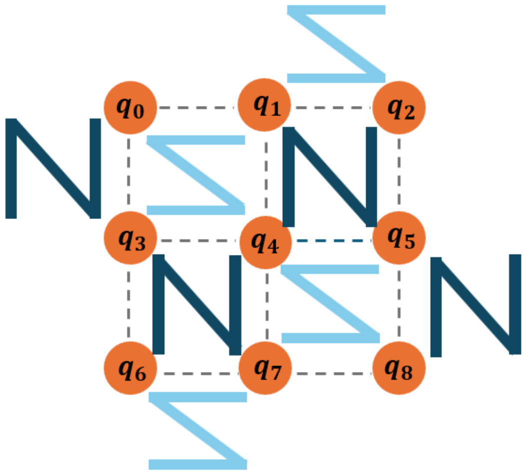


Figure 4.2: A proper and interleaved matching of the distance-3 surface code. Each stabilizer is measured following a Z-shape, starting at the top right qubit and ending at the bottom left.

4.2. TETRAHEMIHEXAHEDRON CODE

The Tetrahemihexahedron, a polyhedron, consists of 7 faces: 3 square planes and 8 triangular faces. It has 12 edges and 6 vertices, which makes it an interesting simple candidate for quantum error correction codes due to its geometric properties [1]. Qubits can be placed at the edges of the Tetrahemihexahedron. Each edge represents a data qubit, while the faces of the polyhedron correspond to stabilizers, see Figure 4.3. Here, the red triangular faces are associated with weight 3 Z stabilizers. The yellow square planes are weight 4 Z stabilizers. X stabilizers are chosen to act on the edges surrounding a vertex, which adds 6 weight 4 X stabilizers. As combining all X (or Z) stabilizers leads to the identity, the total encoded logical qubits are $12-13+2=1$. The qubits are labeled as in Figure A.3.

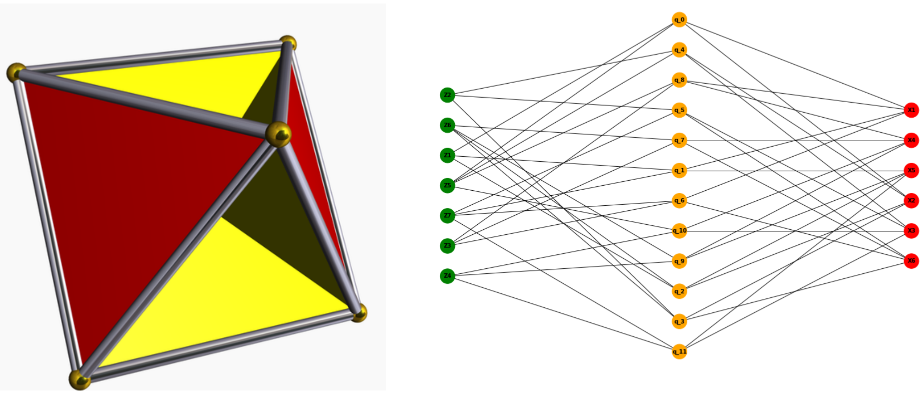


Figure 4.3: Left figure shows a Tetrahemihexahedron. The right figure shows the Tanner graph of the Tetrahemihexahedron. Green nodes represent X-checks, orange nodes represent data qubits and the red nodes represent Z-checks. [31]

Even though the code provides no way to correct errors, it is a good test to see whether the algorithm can also find interleaved schedules for different weight X and Z stabilizers. Running the algorithm on the Tanner graph seen in Figure 4.3, returns an interleaved schedule of length 6. The schedule can be seen in Table A.11. The schedules show no distinct pattern, even when comparing multiple schedules. Again, taking qubits from the same overlapping pair group shows that both first interact with X- or with Z-, showing that the schedule is proper. Also, the found interleaved schedule for the Tetrahemihexahedron seems to be 2 longer than the coloring found by performing a normal edge coloring.

4.3. PERFORMANCE OF IMPROVED ALGORITHM

Section 3.3 introduces an improvement to the algorithm by detecting cycles of removed edges, as illustrated in Figure 3.6. This enhanced algorithm produced interleaved schedules approximately three times faster for the small stellated dodecahedron (SSD) code by prematurely detecting blocking cycles and restarting the algorithm when necessary.

To evaluate the performance gains, both the basic and improved versions of the algorithm were tested on the distance-3 surface code and the Tetrahemihexahedron code. The results are summarized in Table 4.1.

The improvement made to the algorithm did not significantly influence the runtime for the surface code and only a bit for the Tetrahemihexahedron code. Prematurely stopping the algorithm upon detecting a blocking cycle offers no significant advantage, as finding a new schedule takes under 0.1 seconds due to the low amount of edges in their respective Tanner graphs. The additional runtime gained by checking for blocking cycles effectively nullifies any benefit from identifying them. The standard deviation for all 3 codes is high, being a result of each coloring round being chosen based on a random seed.

	Basic algorithm	Improved algorithm
SSD	$(2.2 \pm 2.0)10^3$	$(7.7 \pm 5.4)10^2$
Surface code	(0.43 ± 0.1)	(0.28 ± 0.1)
Tetrahemihexahedron	(0.13 ± 0.1)	(0.12 ± 0.1)

Table 4.1: Table containg the runtime in seconds of the basic and improved algorithm for the SSD, the surface code and the Tetrahemihexahedron code.

5

VERIFICATION OF FAULT TOLERANCE

A proper measurement schedule has been found, enabling us to correctly measure the ancilla qubits in the case where no errors have yet to occur. Our noiseless syndrome extraction circuit now takes the following form;

1. In the first time step we prepare 12 ancilla qubits in the $|+\rangle$ state, and 12 ancilla qubits in $|0\rangle$.
2. A measurement schedule found by the algorithm is performed, putting our data qubits in a quiescent stabilized state.
3. The ancilla qubits get measured in their respective basis and reset into their original state and one EC cycle has been completed.
4. After two cycles, detectors are added to the ancilla qubits, comparing the measurement outcomes of the current cycle with the outcomes of the previous cycle. If the two measurements differ, the detector gets triggered, indicating that one of the data qubits has been corrupted by an error.

The Stim package [32] is used to simulate the circuit from the found measurement schedule. One cycle can be seen in Figure 5.1. The visual representation enables us to quickly check whether the support of each ancilla qubit is correctly defined and gives a sense of scale. The red dotted lines denote the beginning of a new time step. It can be seen that the total amount of time steps $T = 1+6+1 = 8$, having 4 CNOT's layers less than the sequential measurement schedule. [1].

5.1. DEPOLARIZING NOISE MODEL

We consider the standard circuit-level depolarizing noise model [33], described by:

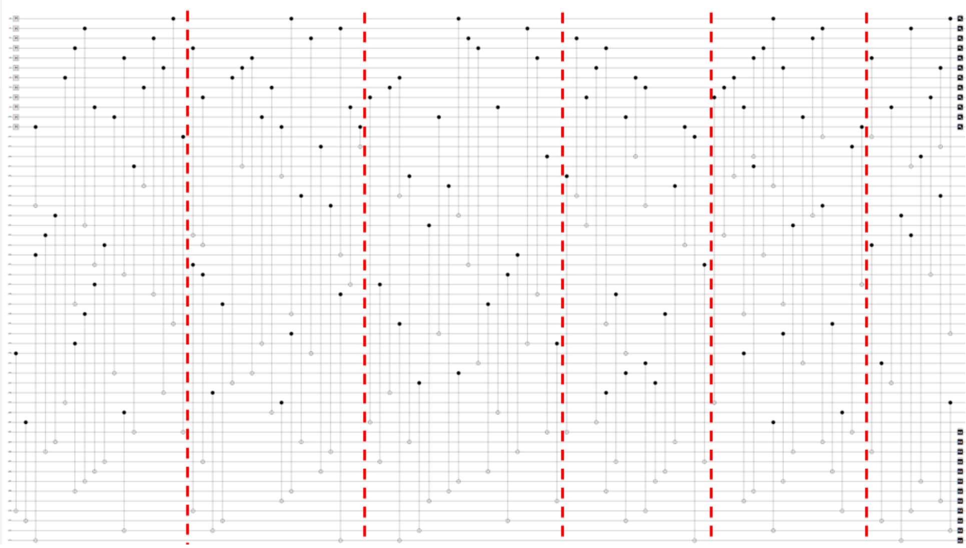


Figure 5.1: Measurement schedule found by the algorithm described in Chapter 2. The top 12 qubits are the X ancilla qubits, the 30 following qubits are the data qubits and the last 12 qubits are the Z ancilla qubits. The red dotted lines mark the beginning of a new timestep.

1. Each CNOT gate is followed by a two-qubit Pauli error drawn from $\{I, X, Z, Y\}^{\otimes 2} \setminus \{I \otimes I\}$ with probability p.

2. Each qubit initialized in $|0\rangle$ gets replaced by $X|0\rangle = |1\rangle$ with probability $\frac{2}{3}p$, . Similarly, a qubit initiliazed in $|+\rangle$ turns into $Z|+\rangle = |-\rangle$ with probability $\frac{2}{3}p$.

3. A measurement's outcome gets flipped with probability $\frac{2}{3}p$.

4. Each idling qubit is followed by an error from $\{X, Z, Y\}$ with probability p.

A visualization of the circuit including noise can be found on GitHub B.

5.2. ERROR PROPAGATION

In Section 2.2 it was shown how Pauli X and Z could propagate through a CNOT and affect the measurement outcome of an ancilla qubit. To fix this problem Theorem 2 was used to make sure an even number of Pauli's ended up at the ancilla qubits. However, with the added noise in the circuit, it now also becomes possible that Pauli X and Z propagate from the ancilla qubits onto the data qubits. Since the SSD code requires only three data qubits to complete a logical error chain, the next round of stabilizer measurements could incorrectly diagnose an error on the third qubit, possibly leading to a logical error chain. See Figure 1.15, where a minimum weight decoder would correct the yellow qubit by applying a Pauli X, but by doing this performs a logical operator.

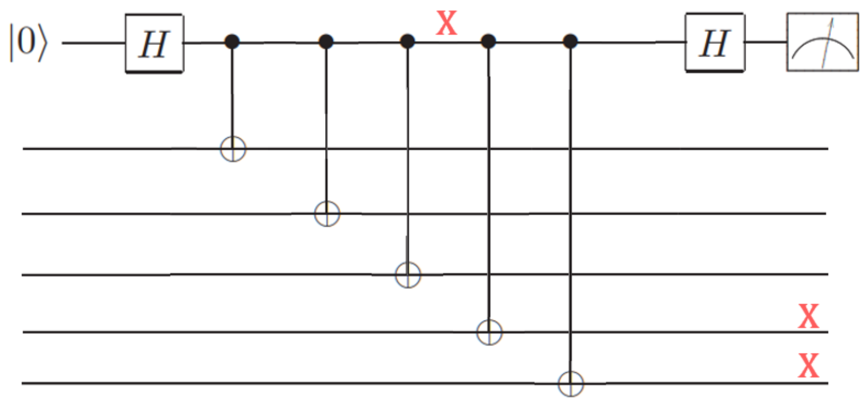


Figure 5.2: A single error on the ancilla qubit spreads into multiple errors on the data qubits.

5.3. FAULT-TOLERANCE OF FOUND SYNDROME EXTRACTION CIRCUITS

To check whether a single error does not harmfully spread into multiple errors, the circuit gets simulated again using STIM, this time including the depolarizing noise. In STIM the logical operators are defined as observables, see Table A.1. The stabilizers of the code are defined by measuring the data qubits and applying a detector between these results and the associated ancilla qubit. It is possible to measure the data qubits directly since maintaining the superposition or entanglement of the qubits is no longer a concern. The focus shifts from maintaining coherence to detecting any missed errors. These final measurements allow it to directly define the stabilizers using the data qubit outcomes, and to construct detectors that reveal any errors that may have gone unnoticed during the earlier QEC cycles.

The integrated function shortest graph-like error[32] is used to determine the smallest number of errors needed to form a logical error. Now as the distance of our code is 3, we expect the minimal amount of errors needed to form a logical error also to be 3. If the circuit could result in a logical error coming from just two errors, it would indicate that at least one of the errors spread in a harmful way. The algorithm described in Chapter 2 was used to find 18 possibly different measurement schedules. Out of the 18 schedules, just 5 proved to be fault-tolerant. For all other schedules a logical X or Z error could occur with only 2 errors occurring in the circuit, indicating harmful spreading [11].

6

Logical performance of the circuits

The previous chapter verified that the found circuits are fault-tolerant. We will now look into finding a pseudo-threshold for the SSD code, which tells us at what error rate p our encoded logical qubits are more susceptible to errors than if we were not to encode our qubits at all. The code will be beneficial for all p lower than the pseudo-threshold. The logical error rate p_L of the SSD code scales as $p_L = cp^2 + O(p^3)$. The dependence on p^2 is justified by the fact the code has distance 3, meaning it can correct a single error, but two errors can still lead to a logical error. The logical error rate is based on a single logical error occurring on one of the 8 logical qubits. As there are 8 logical qubits, the pseudo-threshold is chosen to be the intersection of the logical error rate with the line $p_L(p) = 1-(1-p)^8 \approx 8p$ for small p, such that the intersection denotes the highest error probability p at which the probability of an error occurring on at least one of the 8 unencoded qubits is higher than a logical error occurring on one of the logical qubits.

To sample the logical error probability, 6 rounds of noisy stabilizer measurements are performed and afterward decoded with minimum weight perfect matching from Pymatching [15]. The decoder determines whether a logical error has occurred based on the detectors, which then gets compared to the actual logical observable outcomes of the circuit. Any mismatches show that an undetected logical error has occurred at the end of the 6 rounds.

First, the pseudo-threshold for all found schedules of the SSD code will be determined, comparing possible differences. Then, the best result will be compared against the pseudo-threshold for the sequential measurement from [1]. At last, the best-performing schedule will be compared to the performance of eight copies of distance-3 surface codes.

6.1. INTERLEAVED SCHEDULE FOR THE SSD

A total of 18 different interleaved measurement schedules have been found for the SSD code, 5 of which were fault-tolerant. For these schedules, the circuit noise model described in Chapter 5 was used. As the SSD code encodes 8 logical qubits, we define that a logical error has occurred when a logical error has occurred on at least 1 logical qubit.

An interval of error probability p between 10^{-5} and 10^{-3} was used, in correspondence to actual error probabilities of qubits [9]. In total $2 \cdot 10^7$ samples are taken of the circuit, to then calculate the average logical error rate and deviation in the results. The logical errors are estimated by bootstrap resampling, a technique where the original dataset is repeatedly sampled with replacement to generate multiple simulated datasets [34]. These samples are used to calculate the distribution of the error rates, and from this, the standard deviation serves as an estimation of the variability in the results. The logical X and logical Z error rates are determined separately, through separate X and Z memory experiments, resulting in a logical error rate for both X and Z errors.

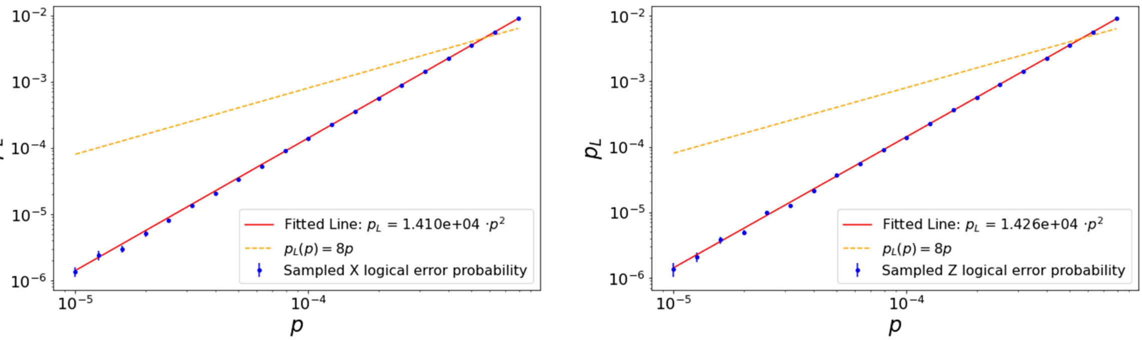


Figure 6.1: Logical X (left) and Z (right) error rate for the best-performing measurement schedule. Blue dots represent the sampled data. The red line is fitted on the data, with $c = 1.410 \cdot 10^4$ and $c = 1.426 \cdot 10^4$, and the line $p_L(p) = 8p$ is dotted yellow.

	Best-performing schedule	Worst-performing schedule
Fitted constant c	$1.410 \cdot 10^4$ $1.426 \cdot 10^4$	$1.470 \cdot 10^4$ $1.452 \cdot 10^4$
Pseudo-threshold	$5.676 \cdot 10^{-4}$ $5.611 \cdot 10^{-4}$	$5.440 \cdot 10^{-4}$ $5.511 \cdot 10^{-4}$

Table 6.1: Performance comparison between best-performing and worst-performing schedules. The top values in each cell correspond to parameters associated with the logical X error rate, while the bottom values correspond to those associated with the logical Z error rate.

Figure 6.1 shows the fitted $p_L = cp^2$ on the best-performing fault-tolerant circuit, see Table A.7. That is, the circuit with the lowest fitted constant c. The figure also shows the line $p_L(p) = 8p$, whose intersection with the fitted error rate determines the pseudo-threshold. The worst fault-tolerant circuit A.8 is shown in Figure A.4. Table 6.1 shows the fitted constant c and the pseudo-thresholds for the best- and worst-performing schedules. The best-performing circuit has a constant c 4.3% lower for X errors and 2% lower for Z errors. These are not huge margins, and the algorithm described in Chapter 2 thus seems to return schedules with about the same performance, provided they are fault-tolerant.

The above procedure was also performed for one non-fault-tolerant circuit to see the difference between fault-tolerant and non-fault-tolerant circuits. The chosen circuit was not X fault-tolerant. The graph seen in Figure 6.2 shows that the logical X error rate p_L instead scales with $p_L = cp$ at lower p, since now single errors can also lead to logical errors. The logical Z error rate is the same as for other fault-tolerant circuits, as the schedule is fault-tolerant against logical Z errors, the graph can be seen in Figure A.5.

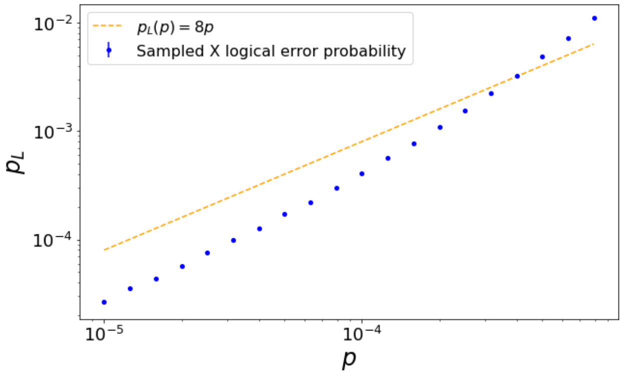


Figure 6.2: Logical X error rate for a non-fault-tolerant schedule, see Table A.9. Blue dots represent the sampled data. It can be seen that for lower error rates p the data seems to follow more the $p_L(p) = 8p$ line at low p.

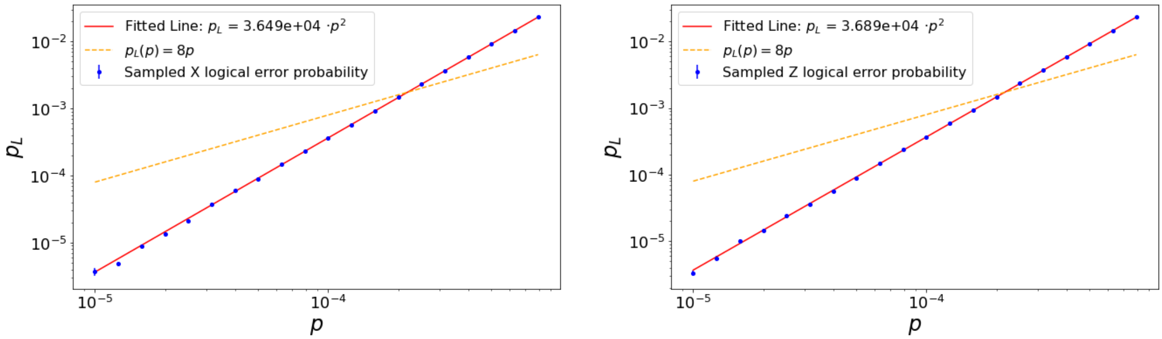


Figure 6.3: Logical X (left) and Z (right) error rate for a sequential measurement schedule, see Table A.6. Blue dots represent the sampled data. The red line is fitted on the data, with $c = 3.649 \cdot 10^4$ and $c = 3.689 \cdot 10^4$, and the line $p_L(p) = 8p$ is dotted yellow.

6.2. SEQUENTIAL SCHEDULE FOR THE SSD

One of the main questions to be answered was whether by finding an interleaved schedule the performance of the SSD code could be improved. A fault-tolerant circuit with sequential measurement of X and Z stabilizers was used from [1], where only the 12th X and Z stabilizers were added in a way that the schedule still proved to be fault-tolerant. Table A.6 shows the sequential measurement schedule. In Figure 6.3 the logical error rate of a sequential measurement schedule is shown.

The interleaved schedule significantly outperforms a sequential measurement schedule. For the logical X and Z error rate of the sequential schedule, the fitted constants are respectively $c = 3.649 \cdot 10^4$ and $c = 3.689 \cdot 10^4$, which is about 2.6 times higher than for the best-interleaved schedule. A big contribution to the worse performance of the sequential measurement circuit is the idling of the qubits. On average there are 10 idling qubits at each time step for the interleaved schedule, while there are 18 idling qubits for the sequential measurement schedule. Reducing the probability p of an error occurring on idling qubit to $\frac{p}{10}$, the sequential and interleaved schedules perform more similarly. This is expected because reducing this idling error probability decreases the difference between both circuits, and they start to act more the same. Consequently, completely removing the idling noise results in the same performance. When taking $\frac{p}{10}$, the sequential circuit only performs 1.23 times worse compared to the interleaved schedule. Of course, the pseudo-thresholds scale the same way.

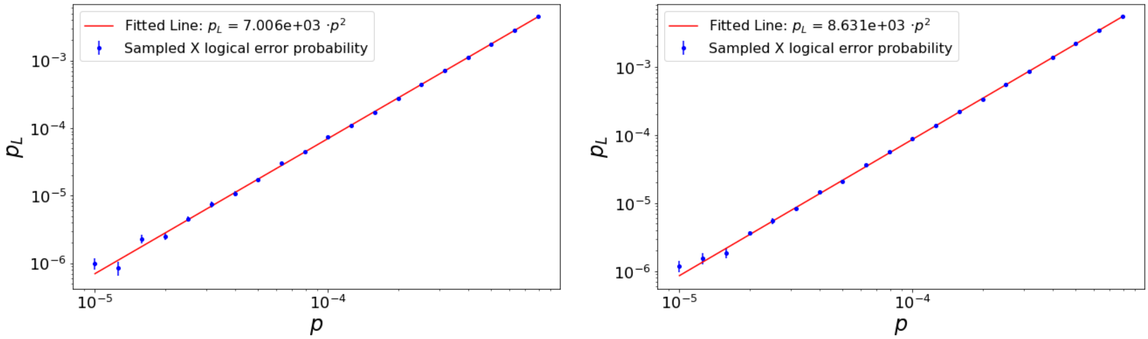


Figure 6.4: Logical X error rate for the best interleaved (left) and sequential (right) measurement schedule. The idling probability has been set to $\frac{p}{10}$ Blue dots represent the sampled data. The red line is fitted on the data, with $c = 7.006 \cdot 10^3$ and $c = 8.631 \cdot 10^3$. The circuits start to act more the same for this lower idling probability.

6.3. 8 COPIES OF THE 17-QUBIT SURFACE CODE

The procedure described in Section 6.1 is also performed for the distance-3 surface code. As the SSD code encodes 8 logical qubits and the 17-qubit surface code only one, 8 pieces of the surface code are used to make a fair comparison against the SSD. Again, a logical error occurs when an error occurs on at least 1 of the logical qubits. Suppose $p_L^{(1)}$ is the logical error rate of one copy of the surface code. In that case, the total logical error rate of 8 copies is $p_L(p) = 1 - (1 - p_L(p))^8 \approx 8p_L^{(1)}(p)$, because errors in different copies are independent between each other. The used measurement schedule can be found in

Table A.10.

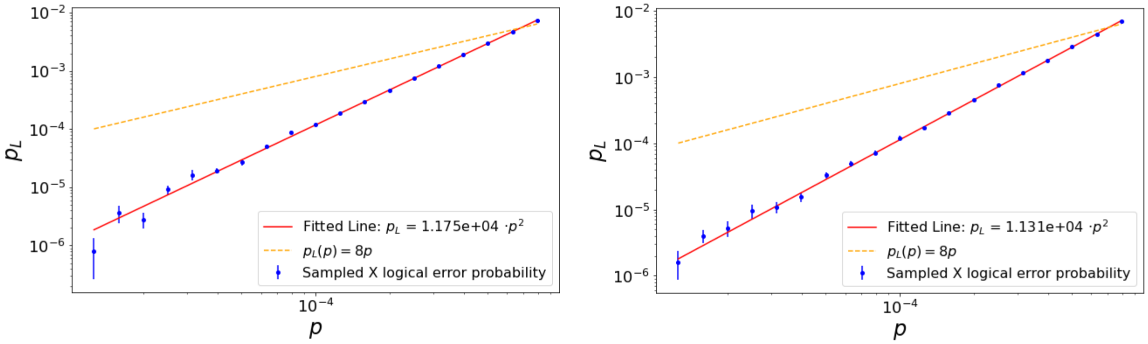


Figure 6.5: The logical X (left) and Z (right) error rates for 8 independent pieces of the surface code. Data was sampled using $2 \cdot 10^7$ samples, and the mean and deviation were found using bootstrapping. Data points are shown in blue, the fitted $p_L = cp^2 + O(p^3)$ is shown in red, and the line $p_L(p) = 8p$ is dotted yellow.

6

It can be seen that the fitted constant c of the logical error rate for the found interleaved schedule is only a factor 1.26 higher than for the logical error rate of 8 pieces of the surface code. The surface code still seems to outperform the SSD code, even though the interleaved schedules perform better than the sequential schedule by a factor of 2.6. However, the SSD code only uses a total of 54 qubits, whereas 8 pieces of the distance-3 surface code use a total of $8 * 17 = 136$. Thus even though the SSD code has a higher logical error rate, it uses only 0.4 times as many qubits. The SSD code can thus form a very promising alternative for future research, particularly in situations where qubit resources are limited. While the surface code remains a strong performer in terms of minimizing logical error rates, the significantly lower qubit overhead of the SSD code suggests that it may offer a good solution in constrained environments.

Moreover, the SSD code can serve as a strong candidate for testing transversal Clifford gates [18], which help ensure that logical operations can be performed without compromising the code's ability to correct errors. They are designed to minimize the creation of logical errors while allowing for logical operations to be performed. The low amount of qubits in the SSD code, combined with its interleaved schedule performance, provides a more compact framework for exploring the performance of transversal Clifford gates.

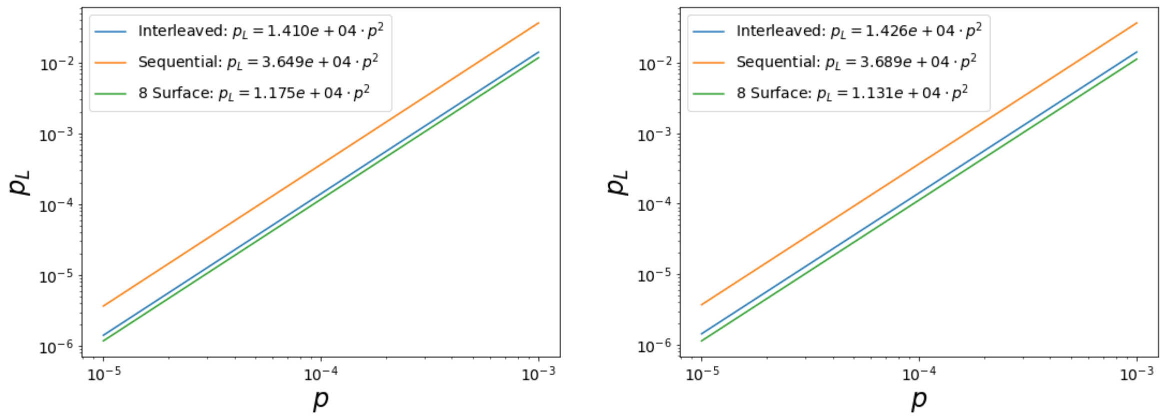


Figure 6.6: The fitted logical error rates for 8 independent pieces of the surface code (green), the sequential schedule (orange) and the interleaved measurement schedule (blue). The performance of the interleaved schedule is approximately 2.6 better than the sequential schedule. However, 8 pieces of the distance-3 surface code still perform 1.26 better than the interleaved schedules.

7

DISCUSSION AND FURTHER IDEAS

The heuristic algorithm described in Chapter 3 proved to find interleaved schedules with 6 CNOT layers, achieving performance close to the performance of the distance-3 surface code. Efforts have been made to find a proper schedule with 5 CNOT layers, yet no conclusive answer could be given as to whether such a circuit exists. It has been shown that such length 5 schedule could not follow a certain ordering, where a vector could be transported along the stabilizers, but if it were to exist, it would have to take a non-ordered form [30]. Determining the existence of a length-5 schedule remains an open question, which a brute-force approach could perhaps answer.

Using the algorithm, 18 interleaved schedules were found, of which five were fault-tolerant. Within these five schedules, variations in the measurement count at the sixth-time step were observed: two schedules had 13 measurements, two had 11, and only one schedule had 10 measurements. Notably, the circuit with the fewest measurements at its final layer showed the best overall performance. This trend could suggest that reducing the number of measurements at the last time step could perhaps lead to better-performing circuits. Further investigation into last round's measurement count and the total length of the schedules could reveal methods for optimizing the development of interleaved schedules.

Significant time was put into finding a pattern within the interleaved schedules. The found schedules for the SSD showed some hints of a pattern, where certain stabilizers had consistent qubit measurement orders across schedules, hinting at an underlying structure that may explain why certain interleaved schedules succeed. However, due to the scale and complexity of the graph, no definitive pattern could be found. Finding a pattern could give more insight into the workings of the schedules, possibly showing how to construct a length 5 schedule or how to manually design new length 6 schedules. Recognizing patterns within the interleaved schedules could serve as a valuable foundation for future research, potentially guiding the development of even better schedules.

The algorithm now detects blocking cycles once they have already occurred, and restarts once one is found. A better approach would be to integrate a way for the algorithm to prematurely disable these blocking sets from forming, instead of only detecting them once formed. An idea could be to not allow any alternating patterns of edges (see Figure 3.6) to exist, and if 3 edges have been colored in an alternating way, to block the 4th edge, even though its own overlapping pair group does not specifically imply this restriction.

The current algorithm produces interleaved scheduling at an acceptable rate. To determine its efficiency, benchmarking it against a standard graph coloring algorithm would be valuable. Comparisons between both methods could indicate how much faster the developed algorithm is than random approaches and whether adjustments could further enhance performance. Benchmarking results would be particularly valuable in assessing the potential applicability of the algorithm for other (larger) codes.

Taking the maximal independent set is based on a random seed. A more deterministic or structured approach could ensure that a proper schedule could be produced every single time, independent of seed randomness. Additionally, the algorithm does not incorporate fault tolerance as a constraint, meaning fault-tolerant schedules are found more by chance rather than by design. By adapting the algorithm to include fault tolerance as a constraint, the likelihood of finding fault-tolerant circuits could be improved.

For the simulations, a depolarizing noise model was used, which applies noise evenly to all qubits and gates [33]. However, exploring other noise models could give a better understanding of how well the algorithm performs in real-world scenarios. For instance, testing models that simulate biased noise or uneven error rates could help us assess the algorithm's reliability and its relevance for quantum systems that experience different types of noise.

7

The developed algorithm works for any code where the overlapping pair groups exist out of 4 edges, and where each edge is in 2 different overlapping groups. In Chapter 4 it was shown that the algorithm also correctly determines proper interleaved schedules for the surface code and a code based on the Tetrahemihexahedron, whose qubits satisfy these constraints. The algorithm is intrinsically built on the fact that the overlapping pair groups exist out of 4 edges, and adapting it to codes where the overlapping pair groups are bigger would require a different approach. However, the basic algorithm can also be improved to work in the case where each single edge is in more than 2 overlapping groups. For this, only the verification step described in Subsection 3.1.3 should be expanded to account for all relevant groups beyond the second (e.g., third, fourth). Extending the improved algorithm, which detects blocking cycles (see Figure 3.6), would be more complex, as the existing cycle-detection strategy may not apply to more intricate codes. Further research on cycle detection in highly intertwined codes could support adaptation for a wider range of quantum codes.

The performance of the SSD showed to come close to the performance of the surface code, despite using only 40% of the amount of qubits. However, the physical implementation for the SSD might be difficult. For example, ion qubit traps [4] and superconducting qubits [7] are often held in a linear or planar trap, where qubits prefer interaction with their nearest neighbors. Adapting the SSD into a linear array or plane of qubits will face some problems. The planar graph, Figure 1.12, can suggest a layout where each Z-check can work on its associated face, however the X-checks pose a problem by not working on neighboring qubits. The surface code has an advantage, having easier connectivity requirements. It is unknown how much this influences the performance of both codes, but it could form a basis for further research to determine the exact shortcomings of the SSD.

8

CONCLUSION

Previous research [1] on the Small Stellated Dodecahedron (SSD) code used measurement circuits that sequentially measured all X stabilizers first, followed by all Z stabilizers. These sequential measurement circuits required 10 CNOT layers to measure all stabilizers but were outperformed by the surface code. The research left an open question regarding the potential existence of interleaved schedules, where both X and Z stabilizers could be measured simultaneously. Deriving an interleaved measurement schedule for the SSD was equivalent to solving an edge coloring problem on its corresponding Tanner graph. An edge coloring algorithm yielded an interleaved schedule with only 5 CNOT layers. However, propagating Pauli X and Z operators through the CNOT gates revealed that this schedule did not correctly measure each stabilizer. For a valid measurement schedule, any pair of qubits interacting with the same stabilizers must both first interact with either the Z or X stabilizer consistently.

To enforce this properness constraint, a new edge coloring algorithm was developed. The objective was to find valid interleaved schedules using fewer time steps than the sequential schedules. A loop-based structure was employed, where each iteration checked qubit pairs for the properness condition and blocked certain edges accordingly. The remaining graph was transformed into a line graph, and a maximal independent set was selected and assigned a color.

The heuristic algorithm discovered 18 interleaved schedules, each utilizing 6 CNOT layers. Although this is an improvement over the 10 layers used in sequential schedules, it remains greater than the 5 layers found in the non-proper schedule. To rule out the existence of a valid 5-layer schedule, further research was conducted on the creation of the 6th color. This additional color emerged when two adjacent edges in a Z stabilizer or two non-adjacent edges in an X stabilizer were assigned colors 1 and 5. A brute-force approach was used to determine whether such a 5-layer schedule could exist, disregarding the properness constraint, and a solution was found. However, attempts to directly derive a proper 5-layer circuit have yet to yield a conclusive answer. It has been demonstrated that this schedule cannot follow a specific ordering—where a vector could be transported along the stabilizers—but if it exists, it must assume a random configuration. The primary unresolved question is whether such a random 5-layer exists.

The algorithm was tested on 2 additional codes, the distance-3 surface code and the Tetrahemihexahedron code. In both cases, the algorithm proved to develop proper interleaved schedules and can thus function as a good heuristic tool for finding interleaved schedules. However, multiple improvements can still be made. For example, implementing a more structured way of choosing independent sets, or verifying fault tolerance whilst the schedule is developed. The algorithm also only works for overlapping groups of size 4, but a more general algorithm could be developed following nearly the same heuristic method.

All 18 interleaved schedules were simulated with Stim, using a circuit-level depolarizing noise model. To maintain the requirement that single errors do not propagate, the integrated function of the shortest graph-like function was employed to detect harmful error spreading. Only 5 schedules proved to be fault-tolerant. The identified interleaved schedules showed performance variability, with error thresholds differing by up to 4.3%. However, the interleaved schedules outperformed the sequential schedules by a factor 2.6. These new interleaved schedules also nearly match the performance of 8 independent distance-3 surface code instances, with the surface code pseudo-threshold being only 1.26 times higher. Considering that the SSD encodes 8 logical qubits using 54 physical qubits, while the surface code requires 136 qubits, the SSD code presents a promising alternative for near-term quantum experiments, particularly in scenarios with limited qubit availability and for the implementation of transversal Clifford gates on the logical qubits.

Bibliography

[1] J. Conrad et al. *The Small Stellated Dodecahedron Code and Friends*. 2018. TuDelft: https://pure.tudelft.nl/ws/portalfiles/portal/51491958/45787026_rsta.2017.0323.pdf.

[2] Alan M. Turing. *On Computable Numbers, with an Application to the Entscheidungsproblem*. URL: https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf.

[3] Richard P. Feynman. *Simulating Physics with Computers*. URL: https://s2.smu.edu/~mitch/class/5395/papers/feynman-quantum-1981.pdf.

[4] J. I. Cirac and P. Zoller. *Quantum Computations with Cold Trapped Ions*. Technical report. URL: https://iontrap.umd.edu/wp-content/uploads/2013/10/Quantum-computations-with-cold-trapped-ions.pdf.

[5] Neil Gershenfeld and Isaac L. Chuang. *Bulk Spin-Resonance Quantum Computation*. Technical report. URL: https://qudev.phys.ethz.ch/static/content/courses/QSIT12/pdfs/Gershenfeld1997.pdf.

[6] Q. A. Turchette et al. *Measurement of Conditional Phase Shifts for Quantum Logic*. URL: https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.75.4710.

[7] Beni Yoshida and Isaac H. Kim. *Topological Quantum Error Correction Codes from Hyperbolic and Other Regular Tessellations*. URL: https://arxiv.org/pdf/2209.06841.

[8] Aleksandar Lebl et al. *Selected Papers of the Unitech 2019 International Scientific Conference*. URL: https://unitech-selectedpapers.tugab.bg/images/papers/2019/s3/s3_p202.pdf.

[9] Chunxiao Xiong et al. *Quantum computational advantage using photons*. URL: https://www.nature.com/articles/s41586-019-1666-5.

[10] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. arXiv: quant-ph/9705052 [quant-ph].

[11] A. Kitaev and J. Preskill. *Surface Codes: Towards Practical Large-Scale Quantum Computation*. arXiv: 1208.0928 [quant-ph].

[12] David J. Griffiths. *Introduction to Quantum Mechanics*. URL: https://www.fisica.net/mecanica-quantica/Griffiths%20-%20Introduction%20to%20quantum%20mechanics.pdf.

[13] Peter W. Shor. *Scheme for Reducing Decoherence in Quantum Computer Memory*. URL: https://doi.org/10.1103/PhysRevA.52.R2493.

[14] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. URL: https://profmcruz.wordpress.com/wp-content/uploads/2017/08/quantum-computation-and-quantum-information-nielsen-chuang.pdf.

[15] Oscar Higgott. *PyMatching: A Python package for decoding quantum error-correcting codes*. URL: https://github.com/oscarhiggott/PyMatching.

[16] Yu Tomita and Krysta M. Svore. *Low-Distance Surface Codes under Realistic Quantum Noise*. arXiv: 1404.3747 [quant-ph].

[17] Nikolas P. Breuckmann et al. *Hyperbolic and Semi-Hyperbolic Surface Codes for Quantum Storage*. arXiv: 1703.00590 [quant-ph].

[18] Nikolas P. Breuckmann and A. R. Burton. *Pymatching: A fast implementation of a matching-based decoder for quantum error-correcting codes*. URL: https://arxiv.org/pdf/2202.06647.

[19] Polyhedr.com. *Dodecahedron Geometry and Symmetry*. URL: https://polyhedr.com/dodecahedron2.html.

[20] Wolfram Research. *Dodecahedral Graph*. URL: https://mathworld.wolfram.com/DodecahedralGraph.html.

[21] Amanda Raymond. *Assignment for Math 324*. Washington University. URL: https://sites.math.washington.edu/~raymonda/assignment.pdf.

[22] Wikipedia. *Tanner graph Wikipedia*. URL: https://en.wikipedia.org/wiki/Tanner_graph.

[23] Ajai Kapoor and Romeo Rizzi. *Edge-Coloring Bipartite Graphs*. URL: https://pdf.sciencedirectassets.com/272497/1-s2.0-S0196677400X00156/1-s2.0-S0196677499910581/main.pdf.

[24] Robert Green. *Vizing's Theorem and Edge-Chromatic Graph Theory*. REU Paper. URL: https://math.uchicago.edu/~may/REU2015/REUPapers/Green.pdf.

[25] Paul Erdős and Vera T. Sós. *On the Coloring of Graphs*. URL: https://users.renyi.hu/~p_erdos/1967-26.pdf.

[26] Vedika Saravanan and Samah Mohamed Saeed. *Pauli Error Propagation-Based Gate Rescheduling for Quantum Circuit Error Mitigation*. URL: https://arxiv.org/pdf/2201.12946.

[27] Wikipedia. *Line graph — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Line_graph.

[28] NetworkX Developers. *maximum_independent_set — NetworkX 3.0 documentation*. URL: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.clique.maximum_independent_set.html.

[29] Wikipedia. *Maximal Independent Set*. URL: https://en.wikipedia.org/wiki/Maximal_independent_set.

[30] Laila K. Khayat and James Oxley. *An Introduction to Extremal Graph Theory*. URL: https://www.math.lsu.edu/system/files/final%20draft.pdf.

8

[31] Wikipedia contributors. *Tetrahemihexahedron*. URL: https://en.wikipedia.org/wiki/Tetrahemihexahedron.

[32] Quantum Computing Group. *Stim Documentation*. URL: https://github.com/quantumlib/Stim/tree/main/doc.

[33] Daniel Gottesman and John Preskill. *Fault-tolerant quantum computation with constant overhead*. URL: 10.1103/PhysRevA.78.012309.

[34] A. C. Davison and D. V. Hinkley. *Bootstrap Methods: A Guide for Practitioners and Researchers*. URL: https://www.hms.harvard.edu/bss/neuro/bornlab/nb204/statistics/bootstrap.pdf.

A

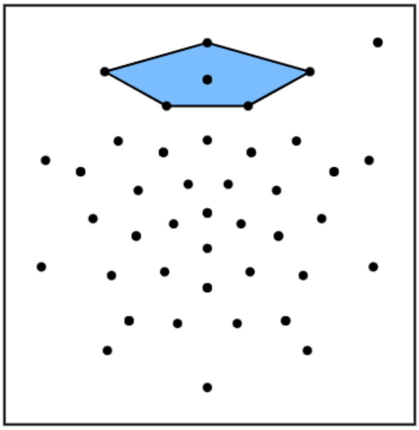
APPENDIX

A.1. VERIFICATION OF RESULTS

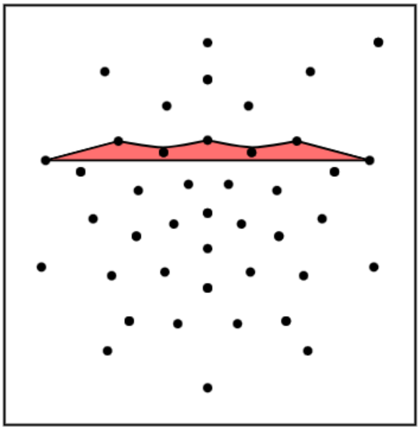
Now that the schedules have been made into actual circuits using Stim, it is wise to take a step back and verify that the results are correct. To verify that our schedules are indeed proper, we initialize all our qubits in $|0\rangle$. For a proper and noiseless circuit, the stabilizer Z measurements should all return the eigenvalue $+1$, as $|0\rangle$ is the $+1$ eigenstate. We would expect random outcomes for the X stabilizers. The same should hold if we initialize all qubits in $|+\rangle$ and measure the X stabilizers. When using the schedules that should be proper, indeed all measurement outcomes return $+1$, and for non-proper schedules, the measurements return random outcomes.

To verify that all stabilizers and logical operators are correctly defined, a detector slice is created, illustrating the detector configurations at each time step. Qubits with measurement outcomes that contribute to the detector are interconnected, forming a detector region. In order to see the logical operators, we replace their logical observable property with a detector in our circuit. A detector slice requires the coordinates of each qubit to make a graph. To find the coordinates of each qubit, the graph seen in 1.12 was used. Each qubit gets placed in the middle of an edge, an ancilla in the middle of each face. STIM determines whether each detector represents a X or Z stabilizer, and colors them respectively blue or red. In Figure A.1 2 stabilizers and 2 logical operators are given, it has been verified for all other stabilizers and logical operators that they are correctly defined too.

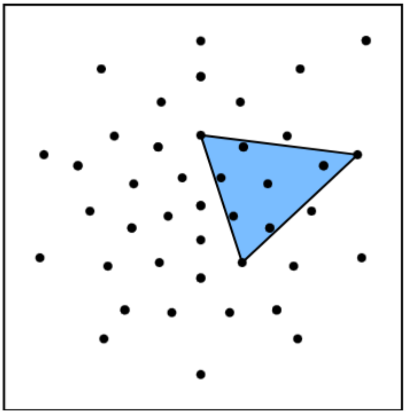
A



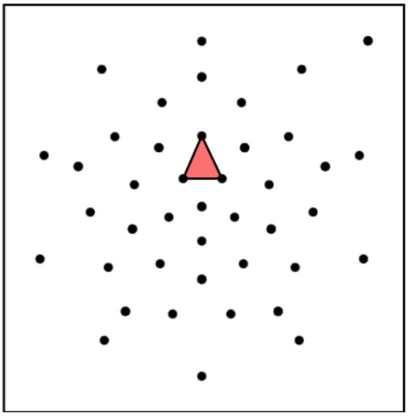
(a) Z1 stabilizer represented as a detector region



(b) X1 stabilizer represented as a detector region



(c) Logical $\overline{Z}_1$ shown as detector region



(d) Logical $\overline{X}_1$ shown as detector region

Figure A.1: Four images of the detector slice showing the detector region of various X and Z operators. Z-type detector region are shown in blue, X-type detector regions in red. Qubits are represented by the black dots.

A.2. X AND Z PAULI PROPAGATION

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad X \otimes I = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\text{CNOT}(X \otimes I) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

But we can also see that

$$(X \otimes X) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

And that we get

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} = (X \otimes X)CNOT$$

Thus

$$\mathrm{CNOT}(X \otimes I) = (X \otimes X)\mathrm{CNOT}, \tag{A.1}$$

A.3. FIGURES

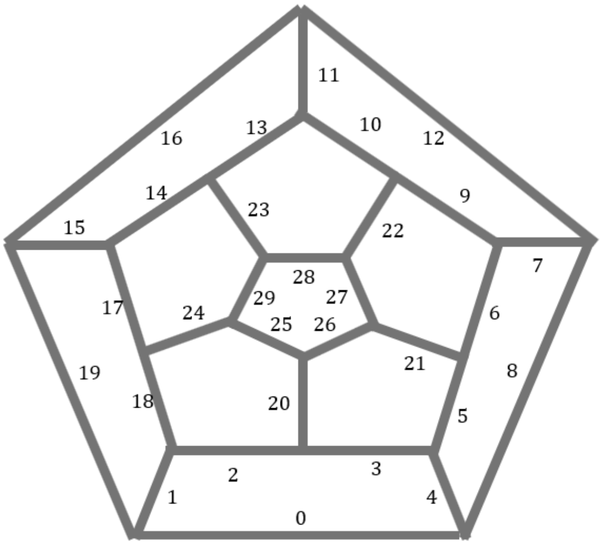


Figure A.2: Labeling of the qubits, showed in a plane graph of the dodecahedron

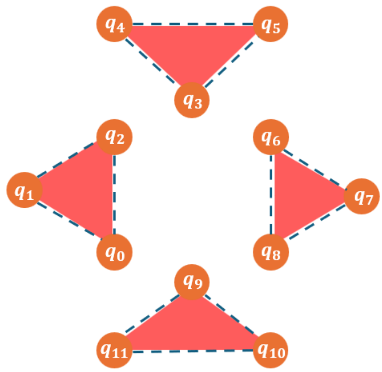


Figure A.3: Labeling of the qubits, showed in a plane graph of the Tetrehemihexahedron. Red faces are shown to denote the weight 3 Z stabilizers.

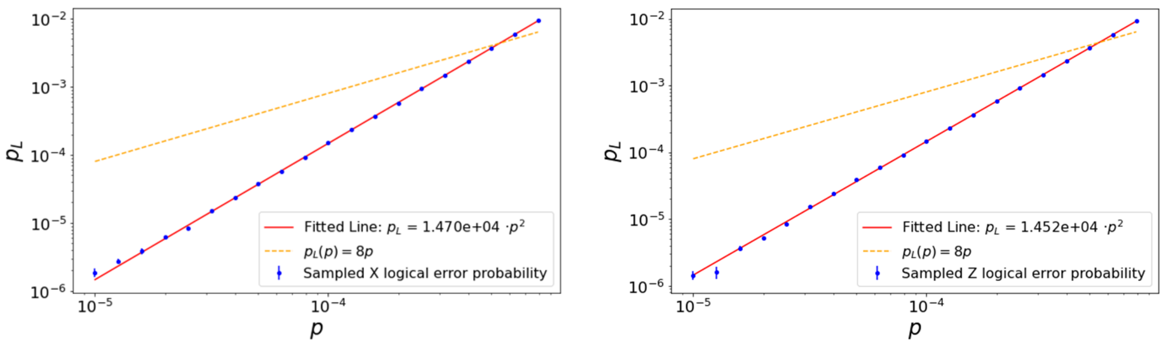


Figure A.4: Logical X (left) and Z (right) error rate for the worst performing measurement schedule. Blue dots represent the sampled data. The red line is fitted on the data, with $c = 1.378 \cdot 10^4$.

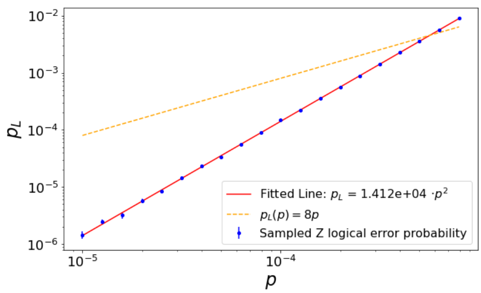


Figure A.5: Logical Z error rate for a non X fault-tolerant circuit, see Table A.9. Blue dots represent the sampled data. The red line is fitted on the data, with $c = 1.412 \cdot 10^4$.

A

A.4. TABLES

Table A.1: Logical $\bar{Z}$'s and $\bar{X}$'s Table. In total 16 different logical operators are shown, using the notation of Figure 1.11 to denote the locations of the qubits it has support on.

Index	**Logical** $\overline{Z}$	**Logical** $\overline{X}$
1	$Z_{(0,6)}Z_{(0,8)}Z_{(6,8)}$	$X_{(0,6)}X_{(2,4)}X_{(3,5)}$
2	$Z_{(0,7)}Z_{(0,9)}Z_{(7,9)}$	$X_{(0,7)}X_{(1,4)}X_{(3,5)}$
3	$Z_{(0,8)}Z_{(0,10)}Z_{(8,10)}$	$X_{(0,10)}X_{(1,3)}X_{(2,4)}$
4	$Z_{(1,7)}Z_{(1,10)}Z_{(7,10)}$	$X_{(1,7)}X_{(4,8)}X_{(5,11)}$
5	$Z_{(2,5)}Z_{(2,6)}Z_{(5,6)}$	$X_{(2,6)}X_{(3,11)}X_{(4,10)}$
6	$Z_{(3,7)}Z_{(3,9)}Z_{(7,9)}$	$X_{(2,4)}X_{(3,5)}X_{(3,7)}X_{(4,10)}$
7	$Z_{(5,6)}Z_{(5,9)}Z_{(6,9)}$	$X_{(1,11)}X_{(2,8)}X_{(5,9)}$
8	$Z_{(0,8)}Z_{(0,9)}Z_{(6,8)}Z_{(6,9)}$	$X_{(0,6)}X_{(0,10)}X_{(1,3)}X_{(1,11)}X_{(3,5)}X_{(5,11)}X_{(6,8)}X_{(8,10)}$

Edge + Color	Edge + Color	Edge + Color	Edge + Color	Edge + Color
(Z_1, q_0) 1	(Z_1, q_1) 2	(Z_1, q_2) 3	(Z_1, q_3) 4	(Z_1, q_4) 5
(Z_2, q_4) 1	(Z_2, q_5) 2	(Z_2, q_6) 3	(Z_2, q_7) 4	(Z_2, q_8) 5
(Z_3, q_7) 1	(Z_3, q_9) 2	(Z_3, q_{10}) 3	(Z_3, q_{11}) 4	(Z_3, q_{12}) 5
(Z_4, q_{11}) 1	(Z_4, q_{13}) 2	(Z_4, q_{14}) 3	(Z_4, q_{15}) 4	(Z_4, q_{16}) 5
(Z_5, q_{15}) 1	(Z_5, q_{17}) 2	(Z_5, q_{18}) 3	(Z_5, q_{19}) 4	(Z_5, q_1) 5
(Z_6, q_{18}) 1	(Z_6, q_2) 2	(Z_6, q_{20}) 3	(Z_6, q_{24}) 4	(Z_6, q_{25}) 5
(Z_7, q_{20}) 1	(Z_7, q_3) 2	(Z_7, q_5) 3	(Z_7, q_{21}) 4	(Z_7, q_{26}) 5
(Z_8, q_{21}) 1	(Z_8, q_6) 2	(Z_8, q_9) 3	(Z_8, q_{27}) 4	(Z_8, q_{22}) 5
(Z_9, q_{22}) 1	(Z_9, q_{10}) 2	(Z_9, q_{13}) 3	(Z_9, q_{23}) 4	(Z_9, q_{28}) 5
(Z_{10}, q_{23}) 1	(Z_{10}, q_{14}) 2	(Z_{10}, q_{29}) 3	(Z_{10}, q_{24}) 5	(Z_{10}, q_{17}) 4
(Z_{11}, q_{29}) 1	(Z_{11}, q_{28}) 2	(Z_{11}, q_{27}) 5	(Z_{11}, q_{26}) 3	(Z_{11}, q_{25}) 4
(Z_{12}, q_0) 2	(Z_{12}, q_8) 1	(Z_{12}, q_{12}) 3	(Z_{12}, q_{16}) 4	(Z_{12}, q_{19}) 5
(X_1, q_5) 1	(X_1, q_8) 2	(X_1, q_{18}) 4	(X_1, q_{19}) 3	(X_1, q_{20}) 5
(X_2, q_3) 1	(X_2, q_9) 4	(X_2, q_{12}) 2	(X_2, q_0) 3	(X_2, q_{21}) 5
(X_3, q_6) 1	(X_3, q_8) 3	(X_3, q_{13}) 5	(X_3, q_{16}) 2	(X_3, q_{22}) 4
(X_4, q_{10}) 1	(X_4, q_{12}) 4	(X_4, q_{17}) 5	(X_4, q_{19}) 2	(X_4, q_{23}) 3
(X_5, q_0) 4	(X_5, q_2) 1	(X_5, q_{14}) 5	(X_5, q_{16}) 3	(X_5, q_{24}) 2
(X_6, q_1) 3	(X_6, q_3) 5	(X_6, q_{17}) 1	(X_6, q_{26}) 2	(X_6, q_{29}) 4
(X_7, q_2) 4	(X_7, q_4) 2	(X_7, q_6) 5	(X_7, q_{25}) 1	(X_7, q_{27}) 3
(X_8, q_5) 4	(X_8, q_7) 2	(X_8, q_{10}) 5	(X_8, q_{26}) 1	(X_8, q_{28}) 3
(X_9, q_9) 1	(X_9, q_{11}) 3	(X_9, q_{14}) 4	(X_9, q_{27}) 2	(X_9, q_{29}) 5
(X_{10}, q_{13}) 1	(X_{10}, q_{15}) 2	(X_{10}, q_{18}) 5	(X_{10}, q_{25}) 3	(X_{10}, q_{28}) 4
(X_{11}, q_{20}) 4	(X_{11}, q_{21}) 2	(X_{11}, q_{22}) 3	(X_{11}, q_{23}) 5	(X_{11}, q_{24}) 1
(X_{12}, q_1) 1	(X_{12}, q_4) 4	(X_{12}, q_7) 3	(X_{12}, q_{11}) 2	(X_{12}, q_{15}) 5

Table A.2: Edges of the SSD Tanner graph and their associated colors. In total 5 colors are used.

A

CNOT 1	CNOT 2	CNOT 3	CNOT 4
$(X1, q_5)$	$(X1, q_8)$	$(Z2, q_5)$	$(Z2, q_8)$
$(X1, q_{19})$	$(X1, q_{18})$	$(Z5, q_{19})$	$(Z5, q_{18})$
$(X1, q_{20})$	$(X1, q_{18})$	$(Z6, q_{20})$	$(Z6, q_{18})$
$(X1, q_{20})$	$(X1, q_5)$	$(Z7, q_{20})$	$(Z7, q_5)$
$(X1, q_{19})$	$(X1, q_8)$	$(Z12, q_{19})$	$(Z12, q_8)$
$(X2, q_0)$	$(X2, q_3)$	$(Z1, q_0)$	$(Z1, q_3)$
$(X2, q_9)$	$(X2, q_{12})$	$(Z3, q_9)$	$(Z3, q_{12})$
$(X2, q_3)$	$(X2, q_{21})$	$(Z7, q_3)$	$(Z7, q_{21})$
$(X2, q_9)$	$(X2, q_{21})$	$(Z8, q_9)$	$(Z8, q_{21})$
$(X2, q_0)$	$(X2, q_{12})$	$(Z12, q_0)$	$(Z12, q_{12})$
$(X3, q_6)$	$(X3, q_8)$	$(Z2, q_6)$	$(Z2, q_8)$
$(X3, q_{16})$	$(X3, q_{13})$	$(Z4, q_{16})$	$(Z4, q_{13})$
$(X3, q_6)$	$(X3, q_{22})$	$(Z8, q_6)$	$(Z8, q_{22})$
$(X3, q_{22})$	$(X3, q_{13})$	$(Z9, q_{22})$	$(Z9, q_{13})$
$(X3, q_{16})$	$(X3, q_8)$	$(Z12, q_{16})$	$(Z12, q_8)$
$(X4, q_{10})$	$(X4, q_{12})$	$(Z3, q_{10})$	$(Z3, q_{12})$
$(X4, q_{19})$	$(X4, q_{17})$	$(Z5, q_{19})$	$(Z5, q_{17})$
$(X4, q_{10})$	$(X4, q_{23})$	$(Z9, q_{10})$	$(Z9, q_{23})$
$(X4, q_{17})$	$(X4, q_{23})$	$(Z10, q_{17})$	$(Z10, q_{23})$
$(X4, q_{19})$	$(X4, q_{12})$	$(Z12, q_{19})$	$(Z12, q_{12})$
$(X5, q_0)$	$(X5, q_2)$	$(Z1, q_0)$	$(Z1, q_2)$
$(X5, q_{16})$	$(X5, q_{14})$	$(Z4, q_{16})$	$(Z4, q_{14})$
$(X5, q_{24})$	$(X5, q_2)$	$(Z6, q_{24})$	$(Z6, q_2)$
$(X5, q_{14})$	$(X5, q_{24})$	$(Z10, q_{14})$	$(Z10, q_{24})$
$(X5, q_{16})$	$(X5, q_0)$	$(Z12, q_{16})$	$(Z12, q_0)$
$(X6, q_1)$	$(X6, q_3)$	$(Z1, q_1)$	$(Z1, q_3)$
$(X6, q_1)$	$(X6, q_{17})$	$(Z5, q_1)$	$(Z5, q_{17})$
$(X6, q_{26})$	$(X6, q_3)$	$(Z7, q_{26})$	$(Z7, q_3)$
$(X6, q_{17})$	$(X6, q_{29})$	$(Z10, q_{17})$	$(Z10, q_{29})$
$(X6, q_{26})$	$(X6, q_{29})$	$(Z11, q_{26})$	$(Z11, q_{29})$

Table A.3: Pairs of qubits with stabilizers

A

CNOT 1	CNOT 2	CNOT 3	CNOT 4
$(X7, q_4)$	$(X7, q_2)$	$(Z1, q_4)$	$(Z1, q_2)$
$(X7, q_6)$	$(X7, q_4)$	$(Z2, q_6)$	$(Z2, q_4)$
$(X7, q_2)$	$(X7, q_{25})$	$(Z6, q_2)$	$(Z6, q_{25})$
$(X7, q_{27})$	$(X7, q_6)$	$(Z8, q_{27})$	$(Z8, q_6)$
$(X7, q_{27})$	$(X7, q_{25})$	$(Z11, q_{27})$	$(Z11, q_{25})$
$(X8, q_5)$	$(X8, q_7)$	$(Z2, q_5)$	$(Z2, q_7)$
$(X8, q_{10})$	$(X8, q_7)$	$(Z3, q_{10})$	$(Z3, q_7)$
$(X8, q_{26})$	$(X8, q_5)$	$(Z7, q_{26})$	$(Z7, q_5)$
$(X8, q_{10})$	$(X8, q_{28})$	$(Z9, q_{10})$	$(Z9, q_{28})$
$(X8, q_{26})$	$(X8, q_{28})$	$(Z11, q_{26})$	$(Z11, q_{28})$
$(X9, q_9)$	$(X9, q_{11})$	$(Z3, q_9)$	$(Z3, q_{11})$
$(X9, q_{14})$	$(X9, q_{11})$	$(Z4, q_{14})$	$(Z4, q_{11})$
$(X9, q_{27})$	$(X9, q_9)$	$(Z8, q_{27})$	$(Z8, q_9)$
$(X9, q_{14})$	$(X9, q_{29})$	$(Z10, q_{14})$	$(Z10, q_{29})$
$(X9, q_{27})$	$(X9, q_{29})$	$(Z11, q_{27})$	$(Z11, q_{29})$
$(X10, q_{13})$	$(X10, q_{15})$	$(Z4, q_{13})$	$(Z4, q_{15})$
$(X10, q_{15})$	$(X10, q_{18})$	$(Z5, q_{15})$	$(Z5, q_{18})$
$(X10, q_{25})$	$(X10, q_{18})$	$(Z6, q_{25})$	$(Z6, q_{18})$
$(X10, q_{13})$	$(X10, q_{28})$	$(Z9, q_{13})$	$(Z9, q_{28})$
$(X10, q_{28})$	$(X10, q_{25})$	$(Z11, q_{28})$	$(Z11, q_{25})$
$(X11, q_{20})$	$(X11, q_{24})$	$(Z6, q_{20})$	$(Z6, q_{24})$
$(X11, q_{20})$	$(X11, q_{21})$	$(Z7, q_{20})$	$(Z7, q_{21})$
$(X11, q_{22})$	$(X11, q_{21})$	$(Z8, q_{22})$	$(Z8, q_{21})$
$(X11, q_{22})$	$(X11, q_{23})$	$(Z9, q_{22})$	$(Z9, q_{23})$
$(X11, q_{24})$	$(X11, q_{23})$	$(Z10, q_{24})$	$(Z10, q_{23})$
$(X12, q_1)$	$(X12, q_4)$	$(Z1, q_1)$	$(Z1, q_4)$
$(X12, q_4)$	$(X12, q_7)$	$(Z2, q_4)$	$(Z2, q_7)$
$(X12, q_7)$	$(X12, q_{11})$	$(Z3, q_7)$	$(Z3, q_{11})$
$(X12, q_{15})$	$(X12, q_{11})$	$(Z4, q_{15})$	$(Z4, q_{11})$
$(X12, q_1)$	$(X12, q_{15})$	$(Z5, q_1)$	$(Z5, q_{15})$

Table A.4: Pairs of qubits with stabilizers. Each row represents a pair.

Stabilizer	Qubits (Cycle)
$Z1$	$q_0(1), q_1(2), q_2(3), q_3(4), q_4(5)$
Z2	$q_4(1), q_5(2), q_6(3), q_7(4), q_8(5)$
Z3	$q_7(1), q_9(2), q_{10}(3), q_{11}(4), q_{12}(5)$
Z4	$q_{11}(1), q_{13}(2), q_{14}(3), q_{15}(4), q_{16}(5)$
Z5	$q_{15}(1), q_{17}(2), q_{18}(3), q_{19}(4), q_1(5)$
Z6	$q_{18}(1), q_2(2), q_{20}(3), q_{24}(4), q_{25}(5)$
Z7	$q_{20}(1), q_3(2), q_5(3), q_{21}(4), q_{26}(5)$
Z8	$q_{21}(1), q_6(2), q_9(3), q_{27}(4), q_{22}(5)$
Z9	$q_{22}(1), q_{10}(2), q_{13}(3), q_{23}(4), q_{28}(5)$
Z10	$q_{23}(1), q_{14}(2), q_{29}(3), q_{17}(4), q_{24}(5)$
Z11	$q_{29}(1), q_{28}(2), q_{27}(5), q_{26}(3), q_{25}(4)$
Z12	$q_{12}(1), q_0(2), q_{19}(5), q_4(3), q_{16}(4)$
X1	$q_{20}(2), q_8(1), q_5(4), q_{19}(3), q_{18}(5)$
X2	$q_0(3), q_9(1), q_{12}(2), q_{21}(5), q_6(4)$
X3	$q_8(2), q_6(1), q_{13}(5), q_{16}(3), q_{22}(4)$
X4	$q_{12}(3), q_{17}(5), q_{10}(4), q_{19}(1), q_{23}(2)$
X5	$q_2(1), q_0(4), q_{14}(5), q_{24}(3), q_{16}(2)$
X6	$q_1(3), q_{26}(2), q_{29}(4), q_{17}(1), q_3(5)$
X7	$q_2(4), q_6(5), q_{25}(1), q_{27}(3), q_4(2)$
X8	$q_5(1), q_7(2), q_{26}(4), q_{10}(5), q_{28}(3)$
X9	$q_9(4), q_{27}(2), q_{29}(5), q_{14}(1), q_{11}(3)$
X10	$q_{13}(1), q_{18}(2), q_{25}(3), q_{15}(5), q_{28}(4)$
X11	$q_{23}(5), q_{24}(1), q_{20}(4), q_{21}(2), q_{22}(3)$
X12	$q_1(1), q_7(3), q_{11}(5), q_{15}(2), q_4(4)$

Table A.5: Qubit measurement per cycle for each ancilla qubit. Every qubit with cycle 1 does not form a pair with the qubit with cycle 5.

A

Timestep	Measurement
0	(X3, q_{16}), (X6, q_{26}), (X10, q_{25}), (X8, q_5), (X1, q_{20}), (X11, q_{21}), (X5, q_0), (X7, q_{27}) (X2, q_9), (X4, q_{10}), (X9, q_{29}), (X12, q_4)
1	(X3, q_{22}), (X6, q_{29}), (X10, q_{13}), (X8, q_{28}), (X1, q_{18}), (X11, q_{20}), (X5, q_{14}), (X7, q_2) (X2, q_{21}), (X4, q_{23}), (X9, q_{27}), (X12, q_{15})
2	(X3, q_6), (X6, q_3), (X10, q_{18}), (X8, q_{26}), (X1, q_8), (X11, q_{22}), (X5, q_{24}), (X7, q_{25}) (X2, q_0), (X4, q_{19}), (X9, q_{14}), (X12, q_7)
3	(X3, q_{13}), (X6, q_1), (X10, q_{28}), (X8, q_{10}), (X1, q_5), (X11, q_{24}), (X5, q_2), (X7, q_6) (X2, q_{12}), (X4, q_{17}), (X9, q_9), (X12, q_{11})
4	(X3, q_8), (X6, q_{17}), (X10, q_{15}), (X8, q_7), (X1, q_{19}), (X11, q_{23}), (X5, q_{16}), (X7, q_4) (X2, q_3), (X4, q_{12}), (X9, q_{11}), (X12, q_1)
5	(Z11, q_{28}), (Z9, q_{10}), (Z10, q_{29}), (Z6, q_{18}), (Z7, q_{20}), (Z8, q_9), (Z1, q_3), (Z2, q_5) (Z3, q_7), (Z4, q_{14}), (Z5, q_{15}), (Z12, q_8)
6	(Z11, q_{29}), (Z9, q_{22}), (Z10, q_{23}), (Z6, q_{25}), (Z7, q_{26}), (Z8, q_{21}), (Z1, q_1), (Z2, q_6) (Z3, q_{10}), (Z4, q_{11}), (Z5, q_{17}), (Z12, q_{16})
7	(Z11, q_{26}), (Z9, q_{13}), (Z10, q_{14}), (Z6, q_{24}), (Z7, q_5), (Z8, q_{27}), (Z1, q_2), (Z2, q_7) (Z3, q_9), (Z4, q_{15}), (Z5, q_1), (Z12, q_{12})
8	(Z11, q_{27}), (Z9, q_{23}), (Z10, q_{24}), (Z6, q_2), (Z7, q_3), (Z8, q_6), (Z1, q_0), (Z2, q_4) (Z3, q_{11}), (Z4, q_{13}), (Z5, q_{18}), (Z12, q_{19})
9	(Z11, q_{25}), (Z9, q_{28}), (Z10, q_{17}), (Z6, q_{20}), (Z7, q_{21}), (Z8, q_{22}), (Z1, q_4), (Z2, q_8) (Z3, q_{12}), (Z4, q_{16}), (Z5, q_{19}), (Z12, q_0)

Table A.6: Sequential measurement for the stellated dodecahedron. It is the same as seen in [1], except for the 12th Z and X stabilizers, which have been added in a way such that the schedule still is fault-tolerant.

Timestep	Measurement
0	(Z11, q_{26}), $(X3, q_8)$, $(X7, q_2)$, $(X5, q_{16})$, $(X1, q_{19})$, $(Z7, q_5)$, $(X12, q_{15})$, $(Z1, q_0)$ (X4, q_{12}), $(Z9, q_{10})$, $(X6, q_3)$, $(X10, q_{25})$, $(X11, q_{23})$, $(X2, q_9)$, $(Z2, q_4)$, $(X9, q_{29})$ (Z3, q_{11}), $(Z8, q_{22})$, $(Z4, q_{13})$, $(Z6, q_{20})$, $(X8, q_7)$, $(Z5, q_{17})$
1	(Z4, q_{16}), $(Z6, q_2)$, $(Z3, q_7)$, $(Z10, q_{23})$, $(X5, q_0)$, $(Z8, q_{27})$, $(X3, q_{13})$, $(X4, q_{10})$ (X1, q_8), $(X2, q_{21})$, $(X12, q_{11})$, $(Z9, q_{28})$, $(Z11, q_{29})$, $(X10, q_{18})$, $(X7, q_{25})$, $(X11, q_{22})$ (Z7, q_{20}), $(Z5, q_{19})$, $(X9, q_{14})$, $(X6, q_{17})$, $(Z12, q_{12})$, $(X8, q_5)$, $(Z1, q_3)$
2	(Z6, q_{24}), $(Z2, q_8)$, $(X10, q_{28})$, $(X1, q_5)$, $(X3, q_{22})$, $(X2, q_3)$, $(Z11, q_{25})$, $(Z7, q_{21})$ (Z9, q_{23}), $(X7, q_4)$, $(X12, q_1)$, $(Z1, q_2)$, $(X9, q_{11})$, $(Z8, q_6)$, $(Z3, q_9)$, $(Z4, q_{15})$ (X5, q_{14}), $(X6, q_{26})$, $(X11, q_{20})$, $(X8, q_{10})$, $(Z12, q_0)$, $(Z10, q_{17})$, $(X4, q_{19})$
3	(X3, q_6), $(Z9, q_{13})$, $(Z2, q_7)$, $(Z4, q_{11})$, $(X11, q_{24})$, $(X9, q_{27})$, $(Z7, q_3)$, $(Z11, q_{28})$ (X10, q_{15}), $(Z10, q_{14})$, $(X1, q_{20})$, $(X4, q_{17})$, $(X2, q_0)$, $(Z12, q_{19})$, $(Z3, q_{10})$, $(Z5, q_1)$ (Z8, q_9), $(Z6, q_{18})$, $(X6, q_{29})$, $(X5, q_2)$, $(X12, q_4)$
4	(X1, q_{18}), $(Z2, q_6)$, $(Z8, q_{21})$, $(X2, q_{12})$, $(X5, q_{24})$, $(X8, q_{28})$, $(X7, q_{27})$, $(X3, q_{16})$ (Z10, q_{29}), $(Z9, q_{22})$, $(Z4, q_{14})$, $(X4, q_{23})$, $(X9, q_9)$, $(Z7, q_{26})$, $(X10, q_{13})$, $(Z5, q_{15})$ (Z6, q_{25}), $(X6, q_1)$, $(X12, q_7)$, $(Z12, q_8)$, $(Z1, q_4)$
5	(Z2, q_5), $(Z1, q_1)$, $(X11, q_{21})$, $(Z3, q_{12})$, $(X8, q_{26})$, $(Z10, q_{24})$, $(Z11, q_{27})$, $(Z12, q_{16})$ (Z5, q_{18}), $(X7, q_6)$

Table A.7: Best performing measurement schedule for the stellated dodecahedron

Timestep	Measurement
0	$(Z12, q_{12}), (X2, q_3), (X3, q_8), (X7, q_6), (Z6, q_2), (Z7, q_5), (Z2, q_7), (Z10, q_{24})$ $(X1, q_{19}), (Z11, q_{26}), (X10, q_{13}), (X11, q_{21}), (X5, q_0), (Z5, q_{15}), (X9, q_{14}), (Z9, q_{22})$ $(X6, q_{17}), (X4, q_{10}), (Z1, q_1), (Z8, q_9), (Z3, q_{11}), (Z4, q_{16})$
1	$(Z2, q_8), (Z3, q_9), (Z1, q_3), (X11, q_{24}), (Z10, q_{14}), (X5, q_{16}), (Z6, q_{25}), (X7, q_{27})$ $(X6, q_{26}), (Z11, q_{28}), (Z5, q_{19}), (X4, q_{23}), (Z8, q_6), (X12, q_{11}), (Z9, q_{10}), (X10, q_{15})$ $(Z12, q_0), (X9, q_{29}), (X1, q_{18}), (X8, q_7), (Z4, q_{13})$
2	$(Z8, q_{21}), (Z11, q_{29}), (Z10, q_{23}), (X9, q_9), (X2, q_0), (Z4, q_{14}), (X5, q_2), (Z9, q_{13})$ $(X4, q_{12}), (X6, q_1), (Z1, q_4), (X7, q_{25}), (Z7, q_{26}), (X10, q_{28}), (X8, q_{10}), (Z2, q_5)$ $(Z6, q_{18}), (X3, q_6), (Z3, q_7), (X12, q_{15}), (Z12, q_{19}), (Z5, q_{17}), (X1, q_8), (X11, q_{20})$
3	$(Z12, q_8), (X4, q_{17}), (Z1, q_0), (X9, q_{11}), (Z2, q_6), (X2, q_{21}), (X11, q_{22}), (X1, q_5)$ $(Z3, q_{10}), (Z6, q_{24}), (X3, q_{16}), (X5, q_{14}), (Z5, q_1), (X7, q_4), (Z11, q_{25}), (Z4, q_{15})$ $(X8, q_{26}), (Z7, q_{20}), (X6, q_3), (Z9, q_{28}), (X12, q_7), (Z8, q_{27})$
4	$(X6, q_{29}), (X8, q_{28}), (X9, q_{27}), (X3, q_{13}), (Z7, q_{21}), (X12, q_1), (Z9, q_{23}), (Z8, q_{22})$ $(Z10, q_{17}), (X4, q_{19}), (Z3, q_{12}), (X1, q_{20}), (Z5, q_{18}), (X10, q_{25}), (Z1, q_2), (Z2, q_4)$ $(X2, q_9), (Z4, q_{11}), (X5, q_{24}), (Z12, q_{16})$
5	$(Z10, q_{29}), (Z7, q_3), (X8, q_5), (X11, q_{23}), (X2, q_{12}), (X7, q_2), (X12, q_4), (X3, q_{22})$ $(Z6, q_{20}), (X10, q_{18}), (Z11, q_{27})$

Table A.8: Worst performing measurement schedule for the stellated dodecahedron

Timestep	Measurement
0	$(Z2, q_6), (X12, q_{15}), (X9, q_{29}), (X6, q_1), (Z7, q_{20}), (X7, q_{25}), (Z6, q_{24}), (Z3, q_9)$ $(X4, q_{10}), (X8, q_5), (X3, q_{13}), (X1, q_8), (Z1, q_4), (X10, q_{18}), (X2, q_0), (Z11, q_{26})$ $(Z8, q_{27}), (X11, q_{22}), (Z4, q_{14}), (Z5, q_{19}), (Z12, q_{16}), (Z10, q_{17}), (Z9, q_{28})$
1	$(Z11, q_{28}), (Z4, q_{15}), (X2, q_9), (X4, q_{12}), (X3, q_{16}), (Z3, q_{11}), (X8, q_{26}), (Z12, q_0)$ $(X1, q_{19}), (Z1, q_2), (X11, q_{23}), (Z6, q_{18}), (Z9, q_{10}), (X7, q_{27}), (Z2, q_4), (X6, q_3)$ $(Z10, q_{29}), (Z8, q_6), (X9, q_{14}), (X5, q_{24}), (Z7, q_{21})$
2	$(X12, q_{11}), (X9, q_{27}), (Z9, q_{13}), (Z2, q_8), (Z8, q_{22}), (Z11, q_{25}), (X10, q_{15}), (X7, q_6)$ $(X8, q_7), (Z1, q_1), (X6, q_{26}), (X11, q_{24}), (Z3, q_{12}), (X2, q_{21}), (Z5, q_{18}), (Z7, q_5)$ $(Z10, q_{23}), (Z4, q_{16}), (X5, q_{14}), (X4, q_{19}), (Z6, q_{20})$
3	$(X7, q_2), (X6, q_{17}), (X10, q_{25}), (X1, q_{20}), (X12, q_4), (Z12, q_8), (X3, q_{22}), (Z3, q_7)$ $(Z10, q_{24}), (Z1, q_0), (Z7, q_3), (Z11, q_{29}), (X8, q_{28}), (Z8, q_9), (X5, q_{16}), (Z4, q_{11})$ $(Z5, q_{15}), (X2, q_{12}), (X4, q_{23}), (X11, q_{21})$
4	$(Z9, q_{23}), (Z8, q_{21}), (Z12, q_{19}), (X6, q_{29}), (Z6, q_2), (Z10, q_{14}), (Z4, q_{13}), (X1, q_5)$ $(X11, q_{20}), (X9, q_{11}), (X12, q_1), (X2, q_3), (X8, q_{10}), (Z7, q_{26}), (Z11, q_{27}), (X7, q_4)$ $(X3, q_8), (X5, q_0), (X10, q_{28}), (Z2, q_7), (Z5, q_{17})$
5	$(X9, q_9), (X1, q_{18}), (X3, q_6), (X4, q_{17}), (Z2, q_5), (Z6, q_{25}), (Z1, q_3), (X5, q_2)$ $(Z12, q_{12}), (Z9, q_{22}), (X10, q_{13}), (Z3, q_{10}), (X12, q_7), (Z5, q_1)$

Table A.9: Non X fault-tolerant schedule for the stellated dodecahedron.

Timestep	Measurement
0	(Z1, q_0), $(Z2, q_2)$, $(X2, q_1)$, $(X3, q_5)$, $(Z3, q_4)$, $(X4, q_7)$
1	(X2, q_0), $(X3, q_4)$, $(X4, q_6)$, $(Z1, q_3)$, $(Z2, q_5)$, $(Z3, q_7)$
2	(X1, q_2), $(X2, q_4)$, $(X3, q_8)$, $(Z2, q_1)$, $(Z3, q_3)$, $(Z4, q_5)$
3	(X1, q_1), $(X2, q_3)$, $(X3, q_7)$, $(Z2, q_4)$, $(Z3, q_6)$, $(Z4, q_8)$

Table A.10: Measurement schedule 17-qubit Surface Code

Timestep	Measurement
0	(Z6, q_9), (X1, q_{11}), (X2, q_8), (X3, q_6), (X5, q_{10}), (Z2, q_4), (X4, q_1), (Z3, q_7), (Z1, q_2)
1	(Z6, q_{10}), (X4, q_0), (X5, q_9), (X1, q_3), (Z2, q_5), (Z3, q_6), (X2, q_4), (Z4, q_{11}), (Z5, q_8), (X3, q_1)
2	(X4, q_6), (Z5, q_1), (Z1, q_0), (X1, q_{10}), (Z6, q_7), (Z2, q_3), (X3, q_5), (Z3, q_8), (Z4, q_9)
3	(X2, q_0), (Z1, q_1), (X5, q_7), (X1, q_2), (Z4, q_{10}), (Z5, q_6), (X4, q_8)
4	(Z5, q_0), (X5, q_8), (X1, q_7)
5	(Z6, q_8), (X1, q_9)

Table A.11: Measurement schedule for Tetrahemihexahedron

B

GitHub

All of the relevant Python scripts and visualisations of the circuits can be found at the GitHub page:

https://github.com/MarcSerraPeralta/ssd-scheduler/tree/dev