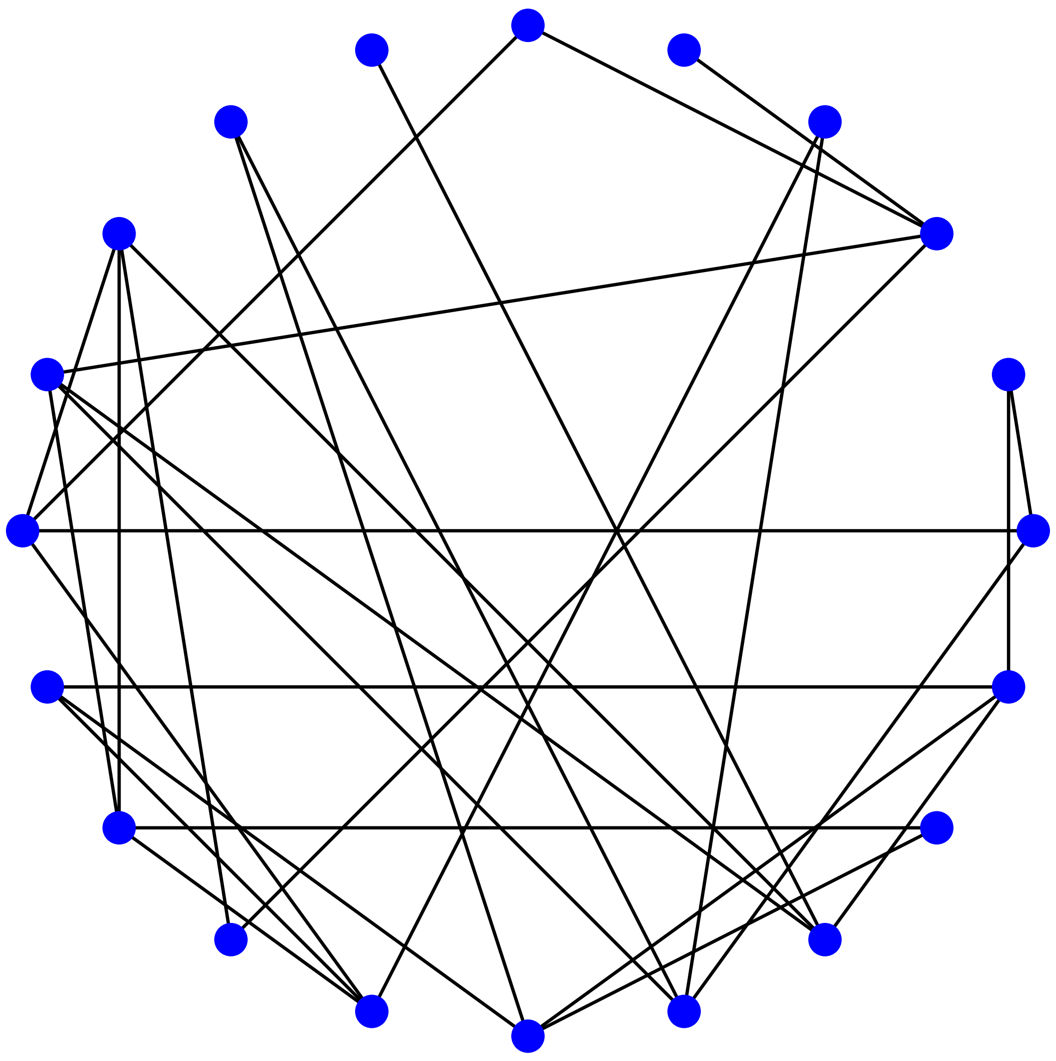
Improved fermion-to-qubit mappings for local and sparse fermionic Hamiltonians

A way to map fermionic Hamiltonians onto qubit Hamiltonians while preserving locality and sparsity and minimizing the number of qubits

Koen Eggen



Improved fermion-to-qubit mappings for local and sparse fermionic Hamiltonians

A way to map fermionic Hamiltonians onto qubit Hamiltonians while preserving locality and sparsity and minimizing the number of qubits

by

Koen Eggen

Delft University of Technology,
to be defended publicly on Monday June 2, 2025.

Student number:	5641276	
Project duration:	February, 2025 – June, 2025	
Thesis committee:	Prof. dr. B.M. Terhal,	TU Delft, supervisor
	Dr. E. Greplová,	TU Delft, supervisor
	Dr. M. Blaauboer,	TU Delft
	Dr. B. Janssens,	TU Delft
Daily supervisor:	MSc. ir. M. Stroeks,	TU Delft
Faculty:	Faculty of Applied Physics and Applied Mathematics, TU Delft	

Abstract

In order to perform simulations of fermionic systems on a quantum computer, there is an encoding needed which maps the fermionic Hamiltonian onto a qubit Hamiltonian. In this thesis, a fermionic encoding is constructed which is less constraining in terms of the required operations that realize this mapping compared to other known mappings.

In the constructed mapping, the Bravyi-Kitaev superfast encoding is used to place qubits at the edges of a graph corresponding to a physical system of n fermionic modes. From this encoding certain stabilizer constraints arise which ensure that the correct fermionic operators are being encoded throughout the entire simulation, and which are used to detect the occurrence of any errors during the simulation.

In combination with the superfast encoding, a randomized algorithm developed by Freedman-Hastings is combined with a stacking- and sewing procedure of the graph and a Vertex Coloring algorithm to obtain a cycle basis which governs the properties of these stabilizer constraints.

This constructed mapping obtains a locality and sparsity of the qubit Hamiltonian terms and stabilizer constraints of $\mathcal{O}(1)$, while keeping the total number of qubits $\mathcal{O}(npoly(\log(n)))$. Here $poly(\log(n))$ means some polynomial in $\log(n)$. It is also numerically shown that the number of qubits can be further reduced, while keeping the locality of the stabilizer constraints $\mathcal{O}(1)$, and the locality and sparsity of the qubit Hamiltonian terms and the sparsity of the stabilizer constraints $\mathcal{O}(poly(\log(n)))$.

Is is furthermore shown how to use this encoding to perform simulations of two fermionic systems, namely the Fermi-Hubbard model on sparse hopping graphs and the sparse SYK model.

Contents

1 Introduction

Everything that is present in the world around us is made of fundamental particles, and in particular fermions play a crucial role [1]. This is the reason that we are interested in performing simulations of fermionic systems to get more insight in their properties and behaviour. One could try to use a normal modern day computer to perform these simulations, but doing very precise simulations quickly becomes too computationally heavy for a normal computer. This is mainly caused by the exponential blow-up of number of calculations needed, since fermions are quantum particles [2].

To perform these simulations, one will need a quantum computer made up of quantum particles called qubits [3]. To be able to perform the simulations on a quantum computer, one first needs to map the fermionic Hamiltonians to qubit Hamiltonians, so that a quantum computer is able to read the information provided. This mapping is called a fermionic encoding [4].

An aspect of a fermionic encoding is to represent the n fermionic modes by a graph for which certain edge and vertex operators can be defined [5]. These operators can be used to represent the interaction terms in the fermionic Hamiltonian, and they will be mapped onto qubit operators. In this way the fermionic Hamiltonian will be mapped onto a qubit Hamiltonian.

Most of the interesting physical systems are described by so called local and sparse fermionic Hamiltonians [6]. A local Hamiltonian means that each term in the Hamiltonian involves only a constant number of modes while the number of fermionic modes n can grow, and a Hamiltonian is sparse if each mode only occurs in a constant number of terms.

The fermionic encoding should preserve both the locality and sparsity of the Hamiltonian terms. This is because performing quantum simulations with constant weight qubit operators ensures that the quantum simulation requires only a limited amount of resources. [2]. Such a mapping might require a number of qubits that exceeds the number of fermionic modes. However, we want to keep the total number of qubits as low as possible. This is because current quantum hardware consists of only a small number of qubits, and so most quantum hardware does not have the resources to perform simulations requiring a lot of qubits [4].

So, the goal of a fermionic encoding is to rewrite the fermionic operators such that their fermionic nature is preserved, and that the resulting qubit Hamiltonian is local and sparse. To furthermore bound the amount of resources needed, this should all be done while keeping the total number of qubits as low as possible.

There are already several known fermionic encodings. The most intuitive one is the Jordan-Wigner transformation where qubits are placed at each vertex in the graph onto which the encoding is applied [4]. Another more complex transformation is the Bravyi-Kitaev superfast encoding, where the qubits are placed at every edge of the graph [5], [7].

However, the Jordan-Wigner transformation does not preserve the locality of the Hamiltonian. In contrast, the Bravyi-Kitaev superfast encoding does provide a local and sparse qubit Hamiltonian, but with the need of some extra stabilizer constraints [5]. These stabilizer constraints define a subspace that corresponds to the space occupied by the fermionic modes to be encoded. Furthermore, these constraints make sure that we stay within this subspace throughout the entire simulation [7] and are used to detect the occurrence of errors during the simulation [2].

The locality and sparsity of these stabilizer constraints are governed by the properties of a cycle basis of the graph onto which the encoding is applied [6]. The locality is determined by the length of the cycles in the basis, and the sparsity is the number of cycles in which a given edge participates. The main disadvantage of the Bravyi-Kitaev superfast encoding is that these stabilizer constraints are not necessarily local and sparse, but we aim to achieve this to minimize the strain on the quantum computer during simulations [2], [8], [9].

In Ref. [10] and [11], another type of fermionic encoding is constructed, which provides a compact mapping which preserves operator locality and sparsity and limits the required qubit to mode ratio to at most 2.5. However, this mapping can only be applied to regular tilings with maximum degree at most 4 and cubic lattices. Since we consider more general sparse graphs in this thesis, we will not discuss this mapping in more detail.

In this thesis, we consider two known algorithms to modify the graph onto which the Bravyi-Kitaev superfast encoding is applied, and to construct a cycle basis for this graph. The first algorithm uses a stacking procedure of the graph and sews a cycle in each layer to obtain small cycles [6]. This algorithm provides a cycle basis where the locality and sparsity of the stabilizer constraints are both $\mathcal{O}(1)$, but on the other hand uses $\mathcal{O}(n^2)$ qubits to obtain this. The other algorithm is developed in the Decongestion Lemma of Freedman-Hastings, which provides a cycle basis with sparsity $\mathcal{O}(poly(\log(n))$ and only uses $\mathcal{O}(n)$ qubits [12]. However, the length of the cycles in this cycle basis could be as big as $\mathcal{O}(n)$.

So for general sparse graphs onto which the superfast encoding is applied, there currently does not exist an algorithm which modifies the graph and constructs a cycle basis for this new graph, such that it exhibits all the desired properties mentioned above. This sets the goal of this thesis, formally formulated in the research question:

How to obtain a fermionic encoding from a local and sparse fermionic Hamiltonian, to a local and sparse qubit Hamiltonian, while minimizing the required number of qubits?

In this thesis, we use the Bravyi-Kitaev superfast encoding in combination with a new algorithm to modify the graph and construct a cycle basis for it. This algorithm makes use of a certain Vertex Coloring algorithm, which is used to obtain an improved stacking- and sewing procedure of the graph and the cycles in its cycle basis. In this way we construct a fermionic encoding which does exhibit all desired properties.

We consider two different scenarios throughout this thesis. In the first scenario we obtain locality and sparsity of the qubit Hamiltonian terms and of the stabilizer constraints of $\mathcal{O}(1)$, while keeping the required number of qubits $\mathcal{O}(npoly(\log(n))$. In the second scenario, we are more interested in reducing the number of qubits. We have numerically obtained a total number of qubits of $\mathcal{O}(npoly(\log(n))$ with a reduced pre-factor compared to the first scenario. To achieve this, we have increased the locality and sparsity of the qubit Hamiltonian and the sparsity of the stabilizer constraints to $\mathcal{O}(poly(\log(n))$, while keeping the locality of the stabilizer constraints $\mathcal{O}(1)$.

In chapter 2 we first discuss the necessary theory on fermions and qubits needed to understand the rest of this thesis. We move on by laying the mathematical foundation on graphs and cycle bases in chapter 3 and discuss a special type of random graph called a Erdos-Renyi graph. In chapter 4 we describe the goal and properties of a general fermionic encoding in more detail, after which we will discuss several known fermionic encodings in chapter 5. We apply these known transformations to two specific physical systems, and then discuss the results of numerically implementing this in chapter 6. In chapter 7 we discuss the two scenarios in more detail and we construct the improved fermionic encoding. We again apply this improved encoding to a physical system and we discuss the results following from the numerical implementation of this in chapter 8. Finally, in chapter 9 we formally prove the properties of this improved fermionic encoding. We end the thesis with a discussion were we look back at the research question and provide some ideas for further research.

2 Fermions and qubits

Everything that is present in the world around us is made of fundamental particles. They are the smallest building blocks of matter. According to the Standard Model [1], we can divide the fundamental particles into two subgroups: fermions and bosons. What both groups have in common, is that they are indistinguishable particles [13]. To see the consequences of this, consider a two particle wavefunction $\Psi(x_1, x_2)$. The probability of finding particle 1 at position x_1 and particle 2 at position x_2 is given by the absolute square of this wavefunction: $|\Psi(x_1, x_2)|^2$. Since both particles are indistinguishable, this probability should remain the same if we switch the labels 1 and 2:

$$|\Psi(x_1, x_2)|^2 = |\Psi(x_2, x_1)|^2. \tag{2.1}$$

This means that we must have:

$$\Psi(x_1, x_2) = e^{i\phi}\Psi(x_2, x_1). \tag{2.2}$$

Here ϕ is some phase factor, and the symmetry postulate for fermions then states that $e^{i\phi} = -1$.

Or in words: the wavefunction describing fermions should be antisymmetric under particle exchange. This property of the wavefunction is one way to divide the fundamental particles into fermions and bosons. Fermions are described by antisymmetric wavefunctions, and bosons by symmetric wavefunctions. An equivalent definition of these subgroups is the fact that bosons have integer spin and fermions have half integer spin, but we will not discuss this property in much detail.

The group of fermions includes among others the electron, which plays a central role in all chemical processes. This is mainly the reason why we are in particular interested in discovering and describing all properties of fermions. Hence in this thesis, we are only concerned with fermions.

Before we dive further into the world of fermions, we will first develop the necessary language and knowledge of basic quantum mechanics that we will need throughout this thesis. We start by introducing some important concepts from linear algebra.

The theory presented in the rest of this chapter is largely based on the book written by M. A. Nielsen and I. L. Chuang given in ref. [2].

2.1. Important concepts from linear algebra

The first concept that we discuss is the notation that we will use for vectors. The standard quantum mechanical notation for a vector is the bra-ket notation developed by Paul Dirac: a ket is written as $|\psi\rangle$, and a bra as $\langle\psi|$. Sometimes we will write the vectors in column notation: $|\psi\rangle = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$.

We now introduce the concept of a vector space.

Definition 2.1.1. *A* complex vector space *is a set V, whose elements are called* vectors *denoted by* $z \in \mathbb{C}$*, which includes the following operations:*

1. Addition: $\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} + \begin{bmatrix} z'_1 \\ \vdots \\ z'_n \end{bmatrix} \equiv \begin{bmatrix} z_1 + z'_1 \\ \vdots \\ z_n + z'_n \end{bmatrix}$.
2. Scalar multiplication: *for any* $c \in \mathbb{C}$, $c \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} \equiv \begin{bmatrix} cz_1 \\ \vdots \\ cz_n \end{bmatrix}$.

In quantum mechanics, one special type of vector space plays a central role. This vector space is called the Hilbert space, which is a vector space equipped with an inner product. We present the general definition of an inner product:

Definition 2.1.2. *A function* $(\cdot, \cdot)$ *from* V *x* V *to* $\mathbb{C}$ *is an* inner product *if it satisfies the following:*

1. $(\cdot, \cdot)$ *is linear in the second argument, meaning that* $(|v\rangle, \sum_i \lambda_i |w_i\rangle) = \sum_i \lambda_i(|v\rangle, |w_i\rangle)$.
2. $(|v\rangle, |w\rangle) = (|w\rangle, |v\rangle)^*$, *where* $*$ *means taking the complex conjugate.*
3. $(|v\rangle, |v\rangle) \geq 0$, *with equality if and only if* $|v\rangle = 0$.

For example, the vector space $\mathbb{C}^n$ has an inner product defined by

$$((y_1, \ldots, y_n), (z_1, \ldots, z_n)) \equiv \sum_{i=1}^{n} y_i^* z_i = [y_1^*, \ldots, y_n^*] \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}. \quad (2.3)$$

Using this knowledge, we now discuss the definitions of a linear span and the basis of a vector space:

Definition 2.1.3.

1. *If* $(|\psi_1\rangle, \ldots, |\psi_n\rangle)$ *is a system of vectors, then the* linear span *of these vectors, written as* Span$(|\psi_1\rangle, \ldots, |\psi_n\rangle)$, *is the set of all linear combinations* $\sum_i a_i |\psi_i\rangle$ *of the vectors* $(|\psi_1\rangle, \ldots, |\psi_n\rangle)$.
2. *If* V *is a* $n-$*dimensional vector space, then the system* $\mathcal{B} = (|\phi_1\rangle, \ldots, |\phi_k\rangle)$ *of vectors in* V *is called a basis for* V *if:*

   (a) *The vectors in* $\mathcal{B}$ *are orthogonal, meaning that the inner product* $(|\phi_i\rangle, |\phi_j\rangle) = 0$ *for any* $i, j \in \{, 1, \ldots, k\}, i \neq j$.
   (b) Span$(\mathcal{B}) = V$.
   (c) *The system* $\mathcal{B}$ *is linear independent, which means that no vector in* $\mathcal{B}$ *can be written as a linear combination of other vectors in* $\mathcal{B}$.

We continue by introducing another important concept used in quantum mechanics, namely the concept of a linear operator.

Definition 2.1.4. *A* linear operator *between vector spaces V and W is defined to be any function* $A : V \to W$ *which is linear in its inputs:* $A\left(\sum_i a_i |\psi_i\rangle\right) = \sum_i a_i A(|\psi_i\rangle)$.

The most convenient way to represent linear operators is in terms of their matrix representations.

Definition 2.1.5. *Suppose* $A : V \to W$ *is a linear operator, and suppose that* $|v_1\rangle, \ldots, |v_m\rangle$ *is a basis for V, and* $|w_1\rangle, \ldots, |w_n\rangle$ *is a basis for W. Then for all* $j \in \{1, \ldots, m\}$, *there exist complex numbers* $A_{1j}, \ldots, A_{nj}$ *such that* $A|v_j\rangle = \sum_i A_{ij} |w_i\rangle$.
The matrix with entries A_{ij} *is then the matrix representation of the operator A.*

In the matrix representation of any operator A, the matrix $A \in C^{n \times m}$ is a linear operator sending vectors in the vector space C^n to the vector space C^m by a matrix multiplication of the matrix A by a vector in C^n.

Throughout this thesis we will need several special operators and matrices, satisfying certain properties:

Definition 2.1.6.

1. *A* Hermitian operator*, given in its matrix representation satisfies the following property:* $A^\dagger \equiv (A^*)^T = A$*, implying that* $(|v\rangle, A|w\rangle) = (A^\dagger |v\rangle, |w\rangle)$*.*
2. *A* normal matrix *satisfies* $AA^\dagger = A^\dagger A$*.*
3. *A* unitary matrix *satisfies* $U^\dagger U = I$*.*

The operators that we discuss throughout this thesis can furthermore satisfy certain important properties relative to each other, namely they can commute or anti-commute.

Definition 2.1.7.

1. *The* commutator *of two operators A and B is defined as* $[A, B] = AB - BA$*.*
2. *The* anti-commutator *is defined as* $\{A, B\} = AB + BA$*.*

Two operators A and B are said to commute if $[A, B] = 0$, and anti-commute if $\{A, B\} = 0$.

Now, if we are given a set of vector spaces, we can construct a larger vector space using the tensor product. The tensor product between two vector spaces V and W is written as $V \otimes W$, and satisfies the following properties:

1. For an arbitrary scalar $z \in \mathbb{C}$ and elements $|v\rangle \in V$ and $|w\rangle \in W$, $z(|v\rangle \otimes |w\rangle) = (z|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z|w\rangle)$.
2. For arbitrary $|v_1\rangle, |v_2\rangle \in V$ and $|w\rangle \in W$, $(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$.
3. For arbitrary $|v\rangle \in V$ and $|w_1\rangle, |w_2\rangle \in W$, $|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$.

The working of the tensor product can be made more clear by writing it in a convenient matrix representation known as the Kronecker product. Suppose $A \in \mathbb{C}^{m\times n}$ and $B \in \mathbb{C}^{p\times q}$. Then we get the following matrix representation for the tensor product:

$$A \otimes B \equiv \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1n}B \\ A_{21}B & A_{22}B & \cdots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1}B & A_{m2}B & \cdots & A_{mn}B \end{bmatrix}. \tag{2.4}$$

which has dimensions $nq \times mp$.
For example, the tensor product of vectors (1, 2) and (3, 4) is the vector:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1\times 3 \\ 1\times 4 \\ 2\times 3 \\ 2\times 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 8 \end{bmatrix}. \tag{2.5}$$

Finally, we mention the notation of taking the tensor product of a vector with itself: $|\psi\rangle^{\otimes 2} = |\psi\rangle \otimes |\psi\rangle$. Tensoring a vector with itself k times is then written as $|\psi\rangle^{\otimes k}$.

Now that we have refreshed our knowledge about linear algebra, it is time to discuss the properties of fermions in more detail.

2.2. Fermions

As said before, the wavefunction describing fermions should be antisymmetric. Up until now we have only discussed what this means if we consider a system of only two particles, but we can generalize this to N particles. An arbitrary wavefunction consisting of N fermions $|\Psi(x_1, \ldots, x_N)\rangle$ can be made antisymmetric via the operation [4]:

$$\mathcal{A}[|\Psi(x_1, \ldots, x_N)\rangle] \equiv \sum_{p\in P} \mathrm{sgn}(p)\, |\Psi(x_{p(1)}, \ldots, x_{p(N)})\rangle\,, \tag{2.6}$$

where P is the set of all unique permutations of the N particles and sgn(p) is the sign of the permutation p. Every time that two particles swap positions we multiply the state with a minus sign to satisfy the symmetry postulate. We let s_p denote the number of swaps needed to obtain a certain permutation $p \in P$ from the original state $|\Psi(x_1, \ldots, x_N)\rangle$. So we must multiply the obtained permutation $|\Psi(x_{p(1)}, \ldots, x_{p(N)})\rangle$ with a factor $(-1)^{s_p}$, hence sgn(p) $= (-1)^{s_p}$. From this we see that the parity of the wavefunction changes if we have an odd number of swaps s_p, and it remains unchanged if the number of swaps is even.

Wavefunctions should always be normalised, since the probability of finding a particle somewhere should always be equal to one, but the normalisation factor is omitted here for clarity. This normalisation factor depends on the overlaps between the permuted wavefunctions, but does not influence the symmetry of the wavefunction.

Now that we have generalised the antisymmetric wavefunction to a system of N fermions, we want to find a way to describe a general fermionic state. We consider a system consisting of n fermionic modes in which N particles are present.

The description of fermionic systems using wavefunctions like $\Psi(x_1, \ldots, x_N)$ becomes very complicated as the system size grows. The solution for this is the *second quantised notation* used in quantum mechanics. This comes down to indicating for each mode whether or not this mode is occupied by a fermionic particle, which automatically ensures the anti-symmetry of the wavefunction.

The *second quantised notation* makes use of special types of fermionic operators called the *creation* and *annihilation* operators. The creation operator applied on mode j is written as $a_j^\dagger$ and this creates a fermion at this particular mode. Taking the Hermitian conjugate of the creation operator, we obtain a_j. This is the annihilation operator for mode j, and it removes a fermion at that mode.

This mode label j could be a position label x or it could for example indicate spin. If j is a position label, the state $|\psi(x_1, x_2)\rangle$ then becomes $a_{x_1}^\dagger a_{x_2}^\dagger |\Omega\rangle$. Here, $|\Omega\rangle$ is the vacuum state, defined as:

$$a_j |\Omega\rangle = 0 \quad \forall j \in [n]. \tag{2.7}$$

Or in words: the vacuum state is defined as the fermionic state in which no particles are present. Using this state, we can mathematically express what it means for the fermionic states to be normalised. Namely, this property can be expressed as $\langle\Omega|\Omega\rangle = 1$.

The creation and annihilation operators satisfy the following anti-commutation relations:

$$\begin{gathered}
\{a_j, a_k^\dagger\} = I\delta_{jk}, \\
\{a_j, a_k\} = 0 \rightarrow a_j a_k = -a_k a_j, \\
\{a_j^\dagger, a_k^\dagger\} = 0, \\
a_j^2 = (a_j^\dagger)^2 = 0.
\end{gathered} \tag{2.8}$$

Using the last relation, we note that we cannot create two fermions at the same mode j. This is known as the Pauli exclusion principle [14].

With the creation and annihilation operators, we can define another type of fermionic operator, called the number operator:

$$n_j = a_j^\dagger a_j. \tag{2.9}$$

This operator counts whether there is a fermion present at mode j. Summing over all modes gives the total number of particles in the system.

Now using all of this, we can define a general description of a fermionic state consisting of n modes and N particles:

$$|\psi\rangle = \sum_{S \subseteq [n]} \alpha_S a_S^\dagger |\Omega\rangle . \tag{2.10}$$

where S is some subset of the collection of n modes, α_S is some normalisation factor, $a_S^\dagger$ is the (ordered) creation of all modes in the subset S and $|\Omega\rangle$ is the vacuum state. Note that if the state has N particles, we have $|S| = N$.

We can calculate the number of particles present in this state as:

$$N = \sum_{j \in [n]} n_j. \tag{2.11}$$

As will become clear in chapter 5, we will need one last type of fermionic operators, namely the Majorana operators c_{2j-1} and c_{2j}. They can be expressed in terms of the creation and annihilation operators:

$$c_{2j-1} = a_j + a_j^\dagger, \quad c_{2j} = \frac{a_j - a_j^\dagger}{i}. \tag{2.12}$$

For the Majorana operators we also present the anti-commutation relations:

$$\begin{aligned} c_j^\dagger &= c_j \text{ for j} \in [2n] \text{ (Hermitian)}, \\ \{c_j, c_k\} &= 2I\delta_{jk} \to c_j^2 = I \text{ for j} \in [2n]. \end{aligned} \tag{2.13}$$

In Appendix A.1, we have worked out an example using these anti-commutation relations for the Majorana operators.

After completing the theory on fermions, we will now discuss the theory on qubits that we will need throughout the rest of this thesis.

2.3. Qubits

In classical computation and information processing, the bit is a fundamental concept. It takes value 0 or 1 and is used to store and transfer information. The quantum counterpart of the bit is the quantum bit, or qubit. In contrast to the distinct values a classical bit can take, the qubit state is a linear combination of zeros and ones:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle. \tag{2.14}$$

Here the entire state is normalised, meaning that $|\alpha|^2 + |\beta|^2 = 1$. This phenomenon is called the principle of superposition, and it thus implicates that a qubit can be in a combination of the states $|0\rangle$ and $|1\rangle$, a purely quantum mechanical phenomenon. The states $|0\rangle$ and $|1\rangle$ form a basis for the Hilbert space in which the qubits live. When we measure a qubit, we find it to be in the state $|0\rangle$ with probability $|\alpha|^2$, and in state $|1\rangle$ with probability $|\beta|^2$. It is because of this that we have the normalisation condition, since all probabilities must some to one. An example of a qubit state is $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, where there is a fifty percent chance of measuring the state $|0\rangle$, and a fifty percent chance of measuring the state $|1\rangle$.

There are different ways to realize a physical qubit, examples of such realizations are:

1. The two different polarizations of a photon, where vertically polarized corresponds to the state $|0\rangle$ and horizontally polarized to $|1\rangle$.
2. The alignment of a nuclear spin in a uniform magnetic field.
3. Two states of an electron orbiting a single atom.

We do not further discuss this, since this is beyond the scope of this research.

One way to visualize the superposition states of qubits is to use the Bloch sphere. We can rewrite eq. 2.14 as:

$$|\psi\rangle = e^{i\gamma}\left[\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right], \tag{2.15}$$

where $\theta, \phi, \gamma \in \mathbb{R}$. This is because for this representation we have $|\alpha|^2 + |\beta|^2 = \left|e^{i\gamma}\cos\left(\frac{\theta}{2}\right)\right|^2 + \left|e^{i\phi}\sin\left(\frac{\theta}{2}\right)\right|^2 = \cos^2\left(\frac{\theta}{2}\right) + \sin^2\left(\frac{\theta}{2}\right) = 1$, so we obtain a valid normalised qubit state.

The phase factor $e^{i\gamma}$ in front can be ignored, because it has no observable effects, so we can write:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle. \tag{2.16}$$

This means that the qubit state $|\psi\rangle$ can be written as a point on a three-dimensional sphere with radius 1, called the Bloch sphere. This sphere is shown in figure 2.1 [2]. So every point on the Bloch sphere corresponds to a qubit state, but if we measure the state in the z-basis, we find that it collapses to either of the two states $|0\rangle$ or $|1\rangle$ [1].

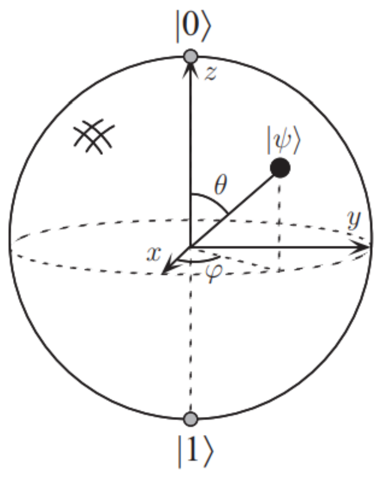


Figure 2.1: A qubit state $|\psi\rangle$ visualized on a Bloch sphere.

The most important operators that we will need to describe qubits are the Pauli X, Y and Z operators:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.17}$$

From these definitions we can see that if we for example apply the Pauli Z-operator twice on the same qubit, we then get: $Z^2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$.

We impose the commutation relations of these Pauli operators (we denote the Pauli operators with P_i here):

$$\begin{aligned} P_i P_j &= -P_j P_i, \text{ where } i, j \in \{X, Y, Z\},\ i \neq j \\ P_i^2 &= I \end{aligned} \tag{2.18}$$

An important remark is that Pauli operators applied on different qubits always commute. Using these Pauli operators, we can define some other operators which can be seen as the Pauli analogues of the fermionic creation and annihilation operators, called the qubit raising and lowering operators. These are respectively defined by:

$$\begin{aligned} \sigma_+ &= \frac{1}{2}(X + iY), \\ \sigma_- &= \frac{1}{2}(X - iY). \end{aligned} \tag{2.19}$$

And they behave in a similar way as the creation and annihilation operators:

$$\begin{aligned} \sigma_+ |0\rangle = |1\rangle, &\quad \sigma_+ |1\rangle = 0, \\ \sigma_- |0\rangle = 0, &\quad \sigma_- |1\rangle = |0\rangle. \end{aligned} \tag{2.20}$$

To conclude the theory on qubits, we note that an n-qubit state lives in the n-fold tensor product of a single-qubit Hilbert space: $\mathcal{H}_0^{\otimes n}$.

This theory about fermions and qubit provides the necessary foundation for understanding the rest of this thesis. We will now continue with some theory about graphs and cycle bases.

[1] One can find more information about quantum measurements and the basis in which a measurement is performed in ref. [2]

3 Preliminaries on graphs and cycle bases

In this chapter, we discuss the mathematical theory on graphs and cycle bases that we will need throughout this thesis. We will represent a fermionic system by a graph G containing V vertices and E edges, representing the operators needed to map a given fermionic Hamiltonian.

As will become clear in chapter 5, we will also need some knowledge about the cycle basis of a graph. In chapters 5 - 8, we will apply the fermionic encodings to two specific types of graphs, namely a graph corresponding to so called sparse SYK Hamiltonians, which are described in sec. 5.5.2, and to a certain type of random graph corresponding to the Fermi-Hubbard model, which is described in sec. 3.2.

3.1. Graphs and cycles bases

As said before, we have a graph $G = (V, E)$ containing V vertices and E edges. We first define the concept of a sparse graph:

Definition 3.1.1. *A graph $G = (V, E)$ is* sparse *if the degree of each vertex is $\mathcal{O}(1)$.*

As a consequence of this, a sparse graph will have a number of edges which is linear in terms of the number of vertices: $|E| = \mathcal{O}(|V|)$. This can be shown using the Handshaking lemma [15], which says that the sum over the degrees of the vertices in the graph is equal to twice the number of edges:

$$\sum_{v \in V} d_v = 2|E|, \tag{3.1}$$

where d_v is the degree of vertex v. Since we know that d_v is bounded by some $d = \mathcal{O}(1)$ for all $v \in V$, we can write: $\sum_{v \in V} d_v \leq d \cdot |V|$, to obtain: $2|E| \leq d \cdot |V|$, or $|E| = \mathcal{O}(|V|)$.

Throughout this thesis, we will only consider sparse graphs G onto which we will apply the fermionic encodings. We explain the reason behind this in chapter 4.

As will become clear in chapter 5, we will need to construct a cycle basis of the graph G:

Definition 3.1.2. *[16]*

1. *A* cycle *is defined as a closed path between vertices in G where no edge is repeated.*
2. *Two cycles in G are* independent *if both cycles have an edge that is not present in the other one.*
3. *A* cycle basis *is a set of independent cycles in G that can be used to generate all other cycles in the graph, by taking the symmetric difference of cycles in the cycle basis*[1]*.*

The number of cycles in a cycle basis is given by the *cyclomatic number* of the graph: $\beta = m - n + c$, where $m = |E|$, $n = |V|$ and c is the number of connected components ($c = 1$ for a connected graph). From this we can for example see that a sparse graph will have a cyclomatic number $\mathcal{O}(|V|)$.

Throughout this thesis, we furthermore consider one specific type of cycle basis, namely a weakly fundamental cycle basis:

Definition 3.1.3. *A set $B = \{C_1, \dots, C_n\}$ of cycles is a* weakly fundamental cycle basis *[17] of a graph G if there exists some permutation σ such that*

$$C_{\sigma(i)} \setminus \left(C_{\sigma(1)} \cup \dots \cup C_{\sigma(i-1)}\right) \neq \emptyset, \forall i = 2, \dots, n. \tag{3.2}$$

Here $C_{\sigma(i)}$ is the collection of edges in the cycle C_i and $C_{\sigma(i)} \setminus C_{\sigma(j)}$ denotes all edges in C_i except those in C_j.

This is the formal definition, but it comes down to the fact that a weakly fundamental cycle basis has the property that its cycles can be placed into a linear ordering such that each cycle includes at least one edge that is not included in any earlier cycle.

3.1.1. Basic algorithms for constructing a cycle basis

The most intuitive algorithm to construct a cycle basis of some graph G, is to use a spanning tree of G. For simplicity, we assume here that G is connected. A *spanning tree* is a subgraph of G that includes all vertices of G, is connected and contains no cycles. One can then use the following steps to construct a cycle basis using a spanning tree:

1. *Find a spanning tree:* choose a spanning tree of the graph, which contains n−1 edges.
2. *Identify the remaining edges:* these are the *fundamental edges*. These form fundamental cycles when added back to the spanning tree.
3. *Construct the cycle basis:* each fundamental edge creates a cycle when added to the spanning tree. Furthermore, since each fundamental edge only occurs in one cycle, these fundamental cycles are independent and we see that they form a cycle basis.

Finding the spanning tree in step 1 can be done using a Depth-First Search (DFS) algorithm which for connected graphs has a running time of $\mathcal{O}(|E|)$, since in the worst case we traverse every edge once [18], [19]. The other steps can also be done in $\mathcal{O}(|E|)$ time, since we also traverse all edges in the worst case. So we conclude that the Spanning Tree algorithm has a time complexity of $\mathcal{O}(|E|)$.

A commonly used algorithm to construct a minimum weight cycle basis is *Horton's algorithm*. The weight of a cycle basis is the sum of the weights of its cycles, which for each cycle can be determined by adding the weight of the edges constructing the cycle.

Horton's algorithm uses the following steps [20]:

1. For every vertex $v \in V$ and edge $(x, y) \in E$, find the shortest paths $P(v, x)$ and $P(v, y)$ between respectively v and x and v and y, using for example Dijkstra's algorithm. After this, create the cycle $C(v, x, y) = P(v, x) + P(v, y) + (x, y)$.
2. Order the cycles by weight.
3. Consider the cycles as rows of a 0-1 matrix, where the columns correspond to the edges of the graph and the rows are the incidence vectors of the cycles. Now use Gaussian elimination using elementary row operations to find all independent cycles. Instead of processing one column at a time, process each row in order of the weights of the cycles. In that way one obtains a minimum weight cycle basis.

This algorithm uses Dijkstra's algorithm to find a shortest path between two vertices, which is explained in Appendix A.3.1. In Appendix A.3.2 an example is given where Horton's algorithm is used.

The total time complexity of Horton's algorithm is $\mathcal{O}(|V|^4)$ [20], and thus this algorithm is slower than the Spanning Tree algorithm. However, Horton's algorithm can be used to find a minimum weight cycle basis as is shown in ref. [20], which is not necessarily the case for the Spanning Tree algorithm.

[1]The symmetric difference between two sets A and B is defined as follows: $A \triangle B = A \setminus B \cup B \setminus A$.

3.2. Random graphs

A specific type of graph that we consider onto which we will apply the fermionic encoding, is the Erdos-Renyi $G(n,p)$ random graph. This graph is constructed such that between every pair of vertices $u, v \in V$ an edge is drawn with probability p [21].

We need to determine some bound or value for this probability p. By construction, every vertex gets in expectation degree $p \cdot (n-1)$ since there are $(n-1)$ vertices with which a given vertex can form a connected pair. Since we want to obtain a graph with constant expected degree to obtain a sparse graph, we then note that we must have $p \cdot (n-1) = c$ for some constant c, or $p \propto \frac{1}{n}$.

By constructing the graph in the way described above, we could obtain a graph which has a vertex with a degree higher than $\mathcal{O}(1)$, since we could add up to $\mathcal{O}(n)$ edges to each vertex. To overcome this issue, we artificially modify the graph such that the degree of each vertex will be $\mathcal{O}(1)$. We do this by considering all vertices who have a degree higher than some threshold, for which we will remove incident edges uniformly at random until the degree of the given vertex is below the threshold value. In this way the degree of each vertex in the graph will be $\mathcal{O}(1)$, and thus the resulting graph will be sparse by def. 3.1.1. Throughout the rest of this thesis we will set this threshold for the degree equal to 4.

The reason that we consider this type of random graph, is that we can show that this graph is equal to one spin-layer of the fermionic interaction graph representing the terms in the Fermi-Hubbard Hamiltonian, which is a widely used physical model in condensed matter physics. We will describe how to represent this model by the random graph discussed above in sec. 5.5.1.

As we will see in chapter 5 we will need to construct a cycle basis of this random graph. As explained earlier in this chapter, this can be done using different types of algorithms as for example the Spanning Tree algorithm of Horton's algorithm. The most important property of this cycle basis that we will need is that there is always a cycle with non-constant length present in the basis, since in that case we need to find a way to reduce the length of this cycle as will be explained in chapter 5.

After numerically implementing different types of algorithms for constructing a cycle basis, including the Spanning Tree algorithm and Horton's algorithm, we have always found a cycle with length $\Omega(\log(n))$ in the basis of this random graph. These results are shown in Appendix B.1.

However, we have not formally proven that such a cycle always exists for this type of graph. If one is interested in a graph which always has a cycle with non-constant length, one can for example investigate the Margulis expander graphs constructed in ref. [22], for which it is shown that the length of their shortest cycle is $\Omega(\log(n))$.

4 Preliminaries on fermionic encodings

Now that we have discussed the theory of fermions and qubits, we move on to the part on fermionic encodings. To perform simulations of fermionic systems on a quantum computer, we need to transform the fermionic Hamiltonian to a qubit Hamiltonian. We achieve this by transforming the fermionic creation and annihilation operators to the qubit Pauli operators. This can be done in several different ways, each with its own advantages and disadvantages. In this chapter, we will discuss the general idea of a fermionic encoding and the properties that need to be investigated when setting up such an encoding.

4.1. General idea of a fermionic encoding

The general idea of a fermionic encoding is to transform the fermionic creation and annihilation operators, a_j and $a_j^\dagger$, to Pauli expressions. The most important property that this encoding must exhibit, is that it preserves the anti-commutation relation of the fermionic operators: $\{a_j, a_k^\dagger\} = I\delta_{jk}$. If we construct the encoding in the most trivial way by just mapping a creation/annihilation operator directly onto a Pauli operator, we do not retain this anti-commutation relation since Pauli operators mutually commute if they act on different qubits. Furthermore, Pauli operators are Hermitian while the creation/annihilation operators are not. So we have to come up with a more complex way to transform the fermionic operators.

There are several already existing fermionic encodings, of which we will discuss a few in chapter 5. When defining more complex fermionic encodings than just directly mapping a fermionic operator onto a Pauli operator, we introduce new kinds of properties of these encodings that we need to investigate to assess the effectiveness and suitability of the mapping.

The main property of a fermionic encoding that we will investigate, is the preservation of the locality and sparsity of the Hamiltonian. Throughout this thesis, we will only consider sparse and local fermionic Hamiltonians, who are formally defined as follows [6]:

Definition 4.1.1. $H = \sum_{i=1}^{m} H_i$ *is a* local sparse n-fermion (resp. n-qubit) Hamiltonian *when the maximum number of Majorana operators (resp. Pauli operators) in each term* H_i *is* $\mathcal{O}(1)$ *(local) and the maximum number of terms involving any fermion (resp. qubit) is* $\mathcal{O}(1)$ *(sparse).*

An informal definition of both terms is that locality means that there are only a constant number of fermions (resp. qubit) present in each term of the Hamiltonian, and sparsity means that each fermion (resp. qubit) is present in only a constant number of terms of the Hamiltonian.

The reason that we consider such Hamiltonians, is that they are ubiquitous in physics. Most of the condensed matter systems of interest are described by local and sparse Hamiltonians. This is because the forces that are of most influence, are on-site repulsion forces and nearest neighbour repulsion and attraction forces. Even if we consider next-nearest neighbour forces, they are still described by a local and sparse Hamiltonian.

Besides retaining the anti-commutation relations of the fermionic operators, we want the fermionic encoding to return a local and sparse qubit Hamiltonian. The reason behind this, is quantum simulation-based. When performing a quantum simulation, we are interested in determining the state $|\psi(t)\rangle$, which is equal to:

$$|\psi(t)\rangle = e^{-i\tilde{H}t} |\psi\rangle , \tag{4.1}$$

where $|\psi\rangle$ is the time-independent qubit state and $\tilde{H}$ is the qubit Hamiltonian. In general, the Hamiltonian $\tilde{H}$ is exponentially large, even when it is sparse, making the direct computation of the matrix exponential computationally too hard. Therefore, we seek approximations of this exponential. One way to do this, is to write the Hamiltonian as:

$$\tilde{H} = \sum_i \tilde{H}_i, \tag{4.2}$$

and to perform short time evolutions under these separate Hamiltonian terms $\tilde{H}_i$. However, since in general $\tilde{H}_j$ and $\tilde{H}_k$ do not commute, we cannot write $e^{-i\tilde{H}t} = \prod_i e^{-i\tilde{H}_i t}$. Instead, we use something known as the Trotter approximation [2]:

$$\lim_{n\to\infty} \left(e^{iAt/n} e^{iBt/n}\right)^n = e^{i(A+B)t}, \tag{4.3}$$

where A and B are Hermitian operators.

As we will show, in order to perform the simulation using a limited amount of resources, we want these $\tilde{H}_i$ Pauli terms to have a constant weight, i.e. they must be local.

Given the local Pauli evolution terms, one has to decompose them into elementary gates in order to run the approximate time evolution on a quantum computer. One way of doing this is by diagonalizing the Pauli term by a single layer of qubit gates and then implement the diagonal evolution term by a decomposition into elementary gates [2]. This process can best be explained by using an example, so assume that we have a term $U_\theta = e^{-i\theta X_1 Z_2 X_3}$, where θ is some coefficient arising from the Trotterization.

During the simulation, we perform the calculations on one qubit and apply elementary gates to entangle it with the other qubits in the term. These elementary gates are CNOT gates, which have the key feature that they apply a conditional operation to the target qubits based on the state of the control qubit, enabling entanglement between them [2].

The first step is to diagonalize U_θ. For this example, this can be done using a Hadamard gate H, which is defined as [2]:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{4.4}$$

Applying Hadamard gates to qubits 1 and 3 transforms the X-operators to Z-operators, giving:

$$\begin{aligned} (H_1 I_2 H_3)\, U_\theta\, (H_1 I_2 H_3) &= \exp\left(-i\theta H_1 X_1 H_1 \otimes Z_2 \otimes H_3 X_3 H_3\right) \\ &= \exp(-i\theta Z_1 Z_2 Z_3), \end{aligned} \tag{4.5}$$

as one can calculate directly using $HXH = Z$.

After having diagonalized U_θ, you can apply a gate sequence that involves a rotation along the z-axis on one of the qubits to implement the diagonalized U_θ. This rotation is given by $R_z(\theta) = e^{-i\theta Z_3}$, and we apply it to the third qubit in the Pauli term. By entangling qubits 1, 2, and 3 with CNOT gates, this operation effectively simulates the full exponential $e^{i\theta Z_1 Z_2 Z_3}$. To obtain the simulated version of the original operator U_θ, we then reverse the basis change and disentangle the qubits again using CNOTs and Hadamard gates. This process can be summarized in the circuit given in fig. 4.1.

The depth of this circuit is determined by the number of two-qubit gates needed, which is equal to the number of CNOT gates as one can see in fig. 4.1. Since each non-identity Pauli operator in the term requires a CNOT gate for the entanglement and disentanglement with the third qubit, the number of CNOTs scales linearly with the weight of the Pauli terms, when performing the gate decomposition in this way. This is the main reason that we want to obtain local Hamiltonian terms $\tilde{H}_i$: if the Pauli terms in the Hamiltonian have constant weight, they can be simulated using a circuit of constant depth.

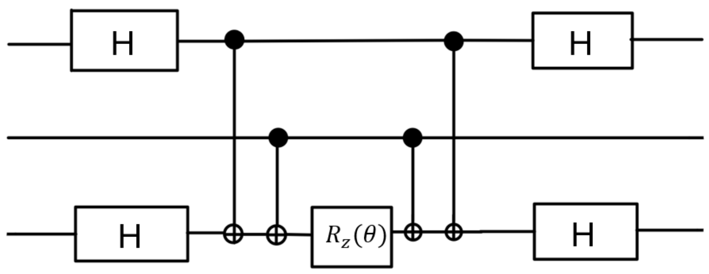


Figure 4.1: Circuit needed for the simulation of a Pauli term. The blocks with an H represent the Hadamard gate, the vertical lines between a black dot and a dot with a $+$ sign in it represent the CNOT gates, and the block with $R_z(\theta)$ represents the rotation along the z-axis. Furthermore, the horizontal black lines represent the qubits, ordered from top to bottom with the first qubit in the term at the top and the last at the bottom.

So with this in mind, we can formulate the research question of this thesis once again:

How to obtain a fermionic encoding from a local and sparse fermionic Hamiltonian, to a local and sparse qubit Hamiltonian, while minimizing the required number of qubits?

Now that we know what a fermionic encoding must satisfy, we discuss the working and properties of several already existing fermion-to-qubit mappings in the next chapter.

5 Known fermionic encodings

In this chapter we discuss a few already known mappings from fermionic modes to qubits. We start with one of the most intuitive ones, known as the Jordan-Wigner transformation, and we will see that this encoding does not retain the locality of the Hamiltonian terms. After this we move on to the more complex Bravyi-Kitaev superfast encoding. We will discover that in this encoding, certain stabilizer constraints arise which are not necessarily local nor sparse. The properties of these stabilizer constraints are governed by modifying the graph onto which we apply the encoding, and constructing a cycle basis for this newly obtained graph G'.

The first algorithm for constructing a cycle basis is a stacking- and sewing procedure, which results in local and sparse stabilizer constraints but needs $\mathcal{O}(n^2)$ qubits to achieve this. Another algorithm is developed in the Decongestion Lemma of Freedman-Hastings, which provides a cycle basis with sparsity $\mathcal{O}(poly\log(n))$ and an ordering of cycles in the basis such that each cycle overlaps with only a polylogarithmic number of cycles later in the basis.

5.1. The Jordan-Wigner transformation

As said, one of the most intuitive transformation is the Jordan-Wigner transformation, which is an example of an n-to-n encoding [4]: it maps n fermionic modes onto n qubits.

The idea of this transformation is to assign an occupied fermionic mode to a $|1\rangle$, and an unoccupied mode to a $|0\rangle$. As discussed in chapter 3, we represent the operators needed to map a given fermionic Hamiltonian by a graph $G = (V, E)$. The main idea of the Jordan-Wigner transformation is to let each vertex in this graph correspond to a qubit. This is visualized for a 2D square lattice in fig. 5.1.

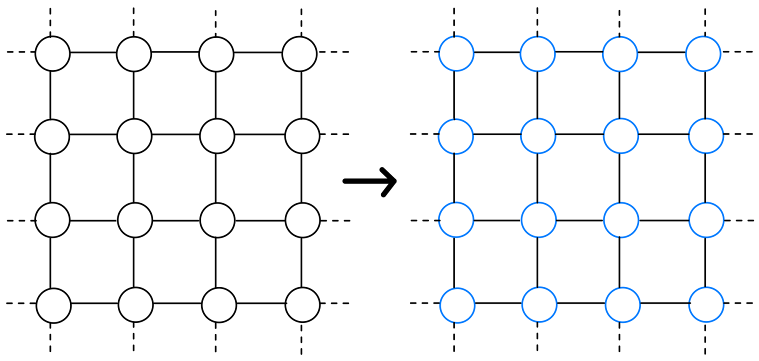


Figure 5.1: Visualization of the Jordan-Wigner transform for a 2D-square lattice. Here it can be seen that the qubits (blue) are placed at the same positions in the grid as the fermionic modes (black).

We will use creation $a_j^\dagger$ and annihilation a_j operators together with the vacuum state $|\Omega\rangle$ to denote a fermionic mode as in 2.10.

The Jordan-Wigner transformation assumes that there is some implicit ordering of the fermionic modes $(1, 2, \dots, n)$. Using this ordering, the transformation can be mathematically formulated as follows [4]:

$$\begin{aligned} a_j &\to \left(\prod_{k<j} Z_k\right)\sigma_j^-, \\ a_j^\dagger &\to \left(\prod_{k<j} Z_k\right)\sigma_j^+. \end{aligned} \tag{5.1}$$

So the Jordan-Wigner transformation maps the fermionic creation and annihilation operators of fermionic mode j onto the qubit raising and lowering operators, where each mapping gets additional Pauli Z-operators applied to the modes earlier in the ordering than the given mode j. These Z-operators are also called Jordan-Wigner strings or Z-strings.

In sec. 2.2 we have seen that the fermionic creation and annihilation operators anti-commute, and any fermionic encoding must preserve this anti-commutation relation. In the Jordan-Wigner transformation, this is ensured by adding the Z-strings to the mapping of the creation and annihilation operators onto the raising and lowering operators. We show this by directly calculating the anti-commutator $\{a_j, a_k^\dagger\}$. We first assume that $j < k$:

$$\begin{aligned} \{a_j, a_k^\dagger\} &= a_j a_k^\dagger + a_k^\dagger a_j \\ &= \left(\prod_{i=1}^{j-1} Z_i\right)\sigma_j^- \cdot \left(\prod_{l=1}^{k-1} Z_l\right)\sigma_k^+ + \left(\prod_{l=1}^{k-1} Z_l\right)\sigma_k^+ \cdot \left(\prod_{i=1}^{j-1} Z_i\right)\sigma_j^- \\ &= \left(\prod_{i=1}^{j-1} Z_i\right)\left(\prod_{l=1}^{j-1} Z_l\right) Z_j\sigma_j^- Z_j \left(\prod_{l=j+1}^{k-1} Z_l\right)\sigma_k^+ + \left(\prod_{l=j+1}^{k-1} Z_l\right) Z_j \left(\prod_{l=1}^{j-1} Z_l\right)\sigma_k^+ \left(\prod_{i=1}^{j-1} Z_i\right)\sigma_j^- \\ &= \left(\prod_{l=1}^{j-1} Z_l\right)^2 Z_j \cdot -Z_j\sigma_j^- \left(\prod_{l=j+1}^{k-1} Z_l\right)\sigma_k^+ + \left(\prod_{l=j+1}^{k-1} Z_l\right)\left(\prod_{l=1}^{j-1} Z_l\right)^2 Z_j\sigma_k^+ Z_j\sigma_j^- \\ &= -\sigma_j^- \left(\prod_{l=j+1}^{k-1} Z_l\right)\sigma_k^+ + \left(\prod_{l=j+1}^{k-1} Z_l\right)\sigma_k^+\sigma_j^- \\ &= \left(\prod_{l=j+1}^{k-1} Z_l\right)(\sigma_k^+\sigma_j^- - \sigma_j^-\sigma_k^+) \\ &= 0. \end{aligned} \tag{5.2}$$

Here we have used the fact that Pauli operators acting on different qubits mutually commute (and thus the raising and lowering operators acting on different qubits also commute), that two Pauli Z-operators acting on the same qubit square to identity, and that different Pauli operators acting on the same qubit anti-commute, and thus we get $\sigma_j^- Z_j = -Z_j\sigma_j^-$.

Now for the case where $j = k$:

$$\begin{aligned}\{a_j, a_j^\dagger\} &= a_j a_j^\dagger + a_j^\dagger a_j \\ &= \left(\prod_{i=1}^{j-1} Z_i\right) \sigma_j^- \cdot \left(\prod_{i=1}^{j-1} Z_i\right) \sigma_j^+ + \left(\prod_{i=1}^{j-1} Z_i\right) \sigma_j^+ \cdot \left(\prod_{i=1}^{j-1} Z_i\right) \sigma_j^- \\ &= \left(\prod_{i=1}^{j-1} Z_i\right)^2 \sigma_j^- \sigma_j^+ + \left(\prod_{i=1}^{j-1} Z_i\right)^2 \sigma_j^+ \sigma_j^- \\ &= I \cdot (\sigma_j^- \sigma_j^+ + \sigma_j^+ \sigma_j^-) \\ &= I.\end{aligned} \tag{5.3}$$

Here we have used that $(\sigma_j^- \sigma_j^+ + \sigma_j^+ \sigma_j^-) = I$.

In a similar way we can show that it also holds that $\{a_j^\dagger, a_k^\dagger\} = I\delta_{jk}$ and $\{a_j, a_k\} = I\delta_{jk}$.

So we see that the Jordan-Wigner transformation is valid in the sense that it maps the fermionic operators onto qubit operators, while maintaining the anti-commutation relations.

So the Jordan-Wigner transformation makes us of the so called Z-strings to maintain the anti-commutation relations for the mapped operators. However, these strings also form the main issue of this mapping. To see this, consider the Fermi-Hubbard model, where the interactions are described by the following Hamiltonian [4]:

$$H_{FH} = -\sum_{\sigma=\{\uparrow,\downarrow\}} \sum_{<j,k>} t_{jk} \left(a_{j,\sigma}^\dagger a_{k,\sigma} + a_{k,\sigma}^\dagger a_{j,\sigma}\right) + \sum_j U_j n_{j,\uparrow} n_{j,\downarrow}, \tag{5.4}$$

where $< i, j >$ indicates that modes i and j are neighbouring modes.

In eq. 5.4, the first term models the hopping terms, the second term considers the on-site interactions and we sum over all possible interactions. We must also consider the different spin-states, since the Pauli exclusion principle implies that two fermions present at the same site must have opposite spin, and having two fermions present at the same site increases the total energy compared to having only one fermion at every site with a value U_j.

The on-site interaction terms do not cause any problems, since the number operators defined in eq. 2.9 are products of creation and annihilation operators on the same site. In that case their Z-strings will cancel out since they both work on the same qubits earlier in the ordering and thus square to identity.

To discuss the hopping terms, first consider the case of a 1D lattice. A hopping term in this case will transform as [4]:

$$a_j^\dagger a_{j+1} \xrightarrow{JW} \sigma_j^+ \sigma_{j+1}^-. \tag{5.5}$$

In this case, all Z-operators acting on qubits preceding the j-th qubit in the ordering will cancel out, so still no problem arises here. However, this is not the case if we consider a 2D square lattice. Consider the snake ordering of the modes as in fig. 5.2.

A hopping term between adjacent sites will map as given in eq. (5.5), but neighbouring modes who are not adjacent in the given ordering will map as [4]:

$$a_j^\dagger a_k \xrightarrow{JW} \sigma_j^+ \left(\prod_{j<p<k} Z_p\right) \sigma_k^-. \tag{5.6}$$

So now this mapping depends not only on the modes j and k, but also on all modes p with $j < p < k$. In the worst case, such a hopping term will depend on $2L$ qubits for an L x L square lattice, which will scale as $\mathcal{O}(N^{1/2})$. So, we see that in this case the Jordan-Wigner transformation provides non-local qubit Hamiltonian terms. As said before, there are ways of fixing this to make this mapping local, but these methods do not work for arbitrary local and sparse fermionic Hamiltonians, so we will not discuss this in more detail.

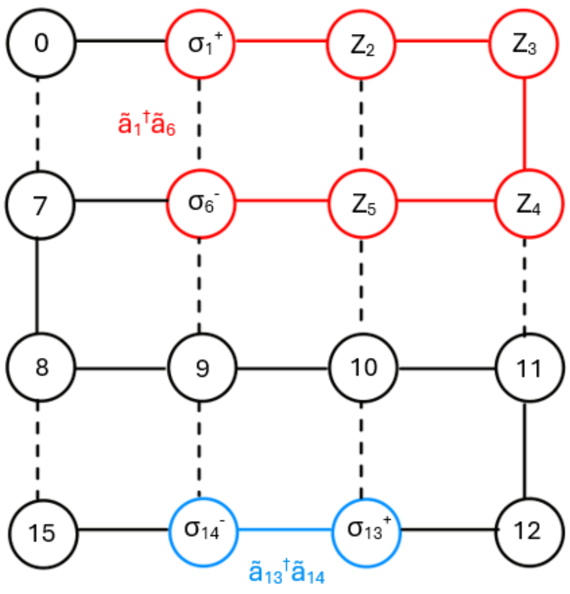


Figure 5.2: A Jordan-Wigner encoded lattice of a 4 x 4 square lattice of fermions. This qubit system encodes one of the two spin-layers of a Fermi-Hubbard system on a 4 x 4 grid. The qubits are numbered in a "snake" ordering along the solid line, the dashed lines are connections between modes who are not adjacent in the ordering. The Jordan-Wigner encoded hopping terms between adjacent (blue) and non-adjacent (red) modes are shown. It can be clearly seen that for neighbouring modes in the ordering, the Z-strings cancel each other out, while for non-neighbouring modes this is not the case, thus creating the problem of non-locality.

So while the Jordan-Wigner transformation is an intuitive transformation and relatively easy to understand, it is far from ideal in the sense of locality because of this behaviour caused by the Z-strings.

There has already been developed another transformation, called the Bravyi-Kitaev superfast encoding, which in contrast does preserve the locality and sparsity of the Hamiltonian terms.

5.2. The Bravyi-Kitaev superfast encoding

The Jordan-Wigner transformation is an example of an encoding where the n fermionic modes get mapped directly onto n qubits, by placing each qubit at a vertex in the graph G used for the encoding. The Bravyi-Kitaev superfast encoding works slightly different. In this encoding, a qubit is placed on every edge of this graph [5]. In this way, the n fermionic modes get mapped onto $m = \mathcal{O}(nd)$ qubits, where d is the degree of the graph G. This can be derived by the Handshaking lemma as described in sec. 3.1. From this, we immediately see the first reason that we consider a graph G with constant degree d. If d would not constant, then the required number of qubits would not be bounded by a constant, which is not beneficial since we want this number to be as low as possible as explained earlier. Fig. 5.3 shows the visualization of this transform for a 2D square lattice. Compare this with fig. 5.1 of the Jordan-Wigner transformation.

Essentially all physical observables are described by even operators, so we are in particular interested in transforming these kinds of operators. So let $\mathcal{F}$ be the algebra of all even operators[1].

For this encoding, we will use the Majorana operators given in 2.12 to describe a fermionic state. We will consider the following generators of $\mathcal{F}$, written in terms of these Majoranas [5]:

$$\begin{aligned} B_j &= -ic_{2j-1}c_{2j} \text{ for each vertex } j \in V, \\ A_{jk} &= -ic_{2j}c_{2k} \text{ for each edge } (j,k) \in E. \end{aligned} \tag{5.7}$$

[1]An algebra is a vector space equipped with a bilinear product, which is a linear operation combining elements of two vector spaces to obtain an element of a third vector space. Any element in an algebra can be expressed in terms of the generators of that given algebra [23].

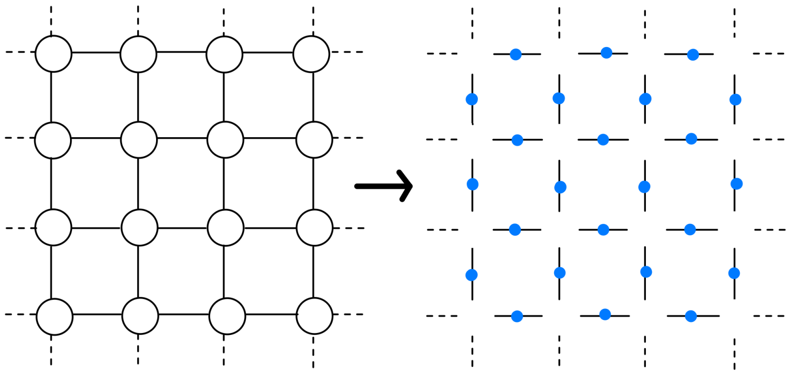


Figure 5.3: Visualization of the Bravyi-Kitaev superfast encoding for a 2D-square lattice. Here it can be seen that a qubit (blue) is placed at every edge between two fermionic modes (black).

It can be shown that these operators satisfy the following commutation rules[2]:

$$B_j^\dagger = B_j, \quad A_{jk}^\dagger = A_{jk}, \tag{5.8}$$

$$B_j^2 = I, \quad A_{jk}^2 = I, \tag{5.9}$$

$$B_j B_k = B_k B_j, \quad A_{jk} = -A_{kj}, \tag{5.10}$$

$$A_{jk} B_l = (-1)^{\delta_{jl}+\delta_{kl}} B_l A_{jk}, \tag{5.11}$$

$$A_{jk} A_{lm} = (-1)^{\delta_{jl}+\delta_{jm}+\delta_{kl}+\delta_{km}} A_{lm} A_{jk}, \tag{5.12}$$

$$i^s A_{\zeta(0),\zeta(1)} A_{\zeta(1),\zeta(2)} \cdots A_{\zeta(s-1),\zeta(0)} = I. \tag{5.13}$$

Where in the last equation ζ is any closed loop of length s in the graph that consists of vertices $\zeta(0), \zeta(1), \ldots \zeta(s) = \zeta(0) \in V$. From these commutation rules we clearly see that the anti-commutation relations of the fermionic creation and annihilation operators is reproduced for these generators.

Let X_{jk}, Y_{jk} and Z_{jk} be the Pauli operators acting on edge $(j,k) \in E$. We label the edges incident to vertex $j \in V$ as $1, \ldots, d(j)$, we denote the corresponding ordering of these edges as $<_j$ and we assume that every edge is oriented. Let $\epsilon_{jk} = 1$ if j is the head of the edge, and $\epsilon_{jk} = -1$ if j is the tail. We can now define the encoded versions of the generators B_j and A_{jk} as [5]:

$$\begin{aligned} \tilde{B}_j &= \prod_{k:(j,k)\in E} Z_{jk}, \\ \tilde{A}_{jk} &= \epsilon_{jk} X_{jk} \prod_{p:(j,p)<_j(j,k)} Z_{jp} \prod_{q:(k,q)<_k(k,j)} Z_{kq}. \end{aligned} \tag{5.14}$$

As an example consider the following (part of a) graph, shown in fig. 5.4. In this example, the edges incident to vertex 1 are ordered as follows: $(1,2) <_1 (1,3) <_1 (1,4)$. The edges incident to vertex 2 are ordered as $(2,5) <_2 (2,6) <_2 (2,1)$, and the edges incident to vertex 3 as $(3,7) <_3 (3,1)$.

We will find out how the generators $\tilde{B}_1$ and $\tilde{A}_{1k}$ are defined, for $k = 2,3,4$. We will start with the simplest one, namely $\tilde{B}_1$. This will become: $\tilde{B}_1 = Z_{12}Z_{13}Z_{14}$, so this is a product of Pauli Z-operators on the edges (1,2), (1,3) and (1,4). For $\tilde{A}_{1k}$ we get three terms:

$$\begin{aligned} \tilde{A}_{12} &= -X_{12}Z_{25}Z_{26}, \\ \tilde{A}_{13} &= X_{13}Z_{12}Z_{37}, \\ \tilde{A}_{14} &= X_{14}Z_{12}Z_{13}. \end{aligned} \tag{5.15}$$

[2]The derivation of all relations shown in this chapter are put in Appendix A

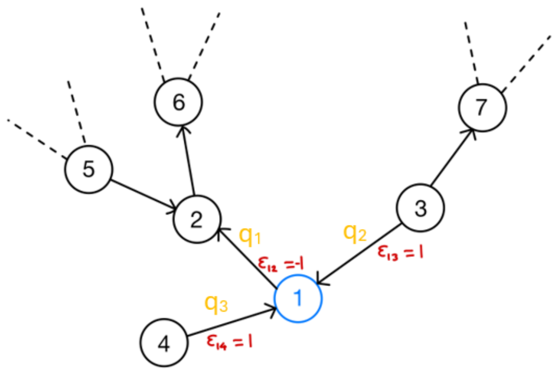


Figure 5.4: A small example of a (part of a) connected graph with vertices $V = \{1, \dots, 7\}$, where qubits are placed on each edge of the graph (the qubits on edges (2,5), (2,6) and (3,7) are not shown here for simplicity). We consider all edges incident to vertex 1 and determine how the generators $\tilde{B}_1$ and $\tilde{A}_{1k}$ are defined for $k = 2, 3, 4$.

From eq. 5.14 we see that in the Bravyi-Kitaev superfast encoding, local fermionic terms get mapped onto local qubit terms if the degree of the graph used for the encoding is $\mathcal{O}(1)$. This is because for a given fermionic term, as we showed in the example, the number of vertex and edge operators needed to construct the encoded qubit term depends on both the number of edges incident to the given vertex, possibly together with the number of edges incident to its neighbours. If the degree of every vertex is then $\mathcal{O}(1)$, we know that the resulting number of operators needed for a given qubit term will also be $\mathcal{O}(1)$.

In a similar way we show that the sparsity of the qubit Hamiltonian obtain from this encoding is also $\mathcal{O}(1)$. Every edge (j,k) that represents a qubit participates in two vertex operators $\tilde{B}_j$ and $\tilde{B}_k$. Besides this, the edge (j,k) participates in the edge operator $\tilde{A}_{jk}$ as a Pauli operator X_{jk} and in the edge operators of the neighbours of the vertices j and k as a Pauli operator Z_{jk} (or Z_{kj}). Since the degree of every vertex is $\mathcal{O}(1)$, we know that every edge participates in $\mathcal{O}(1)$ terms and thus the sparsity of the qubit Hamiltonian terms will be $\mathcal{O}(1)$.

It can be shown that the encoded generators given in 5.14 satisfy similar commutation relations as in eq. 5.8-5.12, but not eq. 5.13. This is among others due to the fact that in general, $X_{ij}X_{jk} \neq I$ for some $(i,j),(j,k) \in E$.

To overcome this issue, we have to restrict the operators $\tilde{B}_j$ and $\tilde{A}_{jk}$ on a certain subspace on which these additional constraints are satisfied. For this, we will define the loop operator for any closed loop ζ [5]:

$$\tilde{A}(\zeta) \equiv i^s \tilde{A}_{\zeta(0),\zeta(1)} \tilde{A}_{\zeta(1),\zeta(2)} \cdots \tilde{A}_{\zeta(s-1),\zeta(0)}. \tag{5.16}$$

It can be checked that $\tilde{A}(\zeta)$ commutes with $\tilde{A}_{jk}$, $\tilde{B}_j$ and itself. We now prepare a state $|\psi\rangle$ such that the following holds:

$$\tilde{A}(\zeta)\,|\psi\rangle = |\psi\rangle \quad \text{for all loops } \zeta. \tag{5.17}$$

Define the subspace $\mathcal{H}_{+1}$ as a stabilizer subspace:

$$\mathcal{H}_{+1} = \{|\psi\rangle : \tilde{A}(\zeta)\,|\psi\rangle = |\psi\rangle \quad \text{for all loops } \zeta\}. \tag{5.18}$$

The number of independent stabilizers is equal to the number of independent loops in the graph, which is the cyclomatic number of the graph: $s = |E| - |V| + 1$. So the s independent stabilizers define a stabilizer subspace of dimension:

$$\text{dim}\,(\mathcal{H}_{+1}) = 2^{|E|-s} = 2^{|V|-1}. \tag{5.19}$$

Since all terms in the qubit Hamiltonian $\tilde{H}$ commute with all loop operators $\tilde{A}(\zeta)$, we know that they can be simultaneously diagonalised so that the eigenstates of $\tilde{H}$ can be divided into $\{|\phi_j^{\text{all}+1}\rangle\}_j$ and $\{|\phi_k^{\text{not all}+1}\rangle\}_k$, where $\{|\phi_j^{\text{all}+1}\rangle\}_j$ is the collection of states such that they have an eigenvalue of $+1$ for all loop operators, and $\{|\phi_k^{\text{not all}+1}\rangle\}_k$ is the collection of states such that there is at least one loop operator for which they do not have an eigenvalue of $+1$.

Since the collection of states $\{|\phi_j^{\text{all}+1}\rangle\}_j$ consists of simultaneous eigenstates of a set of commutating Hermitian operators, we know that these states are orthogonal. Furthermore, since the subspace $\mathcal{H}_{+1}$ consists of all states that are simultaneous eigenstates of all $\tilde{A}(\zeta_j)$ who have eigenvalue $+1$, we actually know that the set $\{|\phi_j^{\text{all}+1}\rangle\}_j$ forms a basis for $\mathcal{H}_{+1}$.

This means that without loss of generality, we can write a state $|\psi\rangle \in \mathcal{H}_{+1}$ as:

$$|\psi\rangle = \sum_j \alpha_j \, |\phi_j^{\text{all}\,+1}\rangle \,, \tag{5.20}$$

where α_j is some normalisation factor.

Now using this expression, we can determine the effect of applying the loop operator $\tilde{A}(\zeta)$ onto the time-dependent state $|\psi(t)\rangle$, which is defined as in eq. 4.1. We can show that the following holds:

$$\tilde{A}(\zeta)\,|\psi(t)\rangle = |\psi(t)\rangle \quad \text{for all loops } \zeta, \tag{5.21}$$

This means that we stay within the subspace created by the stabilizer constraints for all time t, so these constraints will remain satisfied throughout the entire simulation, where we assume the simulation is performed without any errors.

If we take possible errors into account, we must measure the stabilizer constraints during the quantum simulation. This can be done using for example syndrome extraction circuits, which makes use of ancilla qubits. Such a circuit connects the ancilla qubits to the data qubits and performs a measurement on the ancilla qubits. In particular, this connection is a controlled operation, meaning that the unitary operator corresponding to the stabilizer term is applied controlled on the ancilla qubit [2].

The ancilla qubits are prepared in the $|+\rangle$ state and the measurement is then performed in the x-basis. Each loop operator $\tilde{A}(\zeta)$ has eigenvalues ± 1 and the outcome of these measurements indicates whether an error has occurred. If the measurement returns $+1$, the system remains in the stabilizer subspace created by the loop operators, meaning no detectable error has occurred.

But if the measurement returns -1, the state has been moved out of the subspace by an error that anti-commutes with at least one loop operator. So this is a sign that an error has occurred. To get rid of these errors, we only retain those runs in which all stabilizer measurements return $+1$, effectively filtering out states corrupted by detectable errors. This process of measuring the stabilizers during the simulation to detect possible errors is called quantum error detection[3].

The depth of the circuit used to perform the measurements during the simulation depends on the number of CNOT gates needed to entangle the ancillas to the data qubits. The number of CNOT gates required to measure a loop operator scales with its weight, so the depth of the circuit depends on the weight of the stabilizer constraints. Thus, if we have stabilizer constraints with constant locality and sparsity, we can perform these measurements using circuits of constant depth. This is the reason that besides obtaining a local and sparse qubit Hamiltonian as described in chapter 4, we also want to achieve local and sparse stabilizer constraints if we use the superfast encoding.

So to summarize, the Bravyi-Kitaev superfast encoding consists of a Hamiltonian $\tilde{H}$ which is constructed by replacing the edge and vertex operators A_{jk} and B_j, given in eq. 5.7, by their qubit transformations $\tilde{A}_{jk}$ and $\tilde{B}_j$ as given in eq. 5.14, restricted on the subspace $\mathcal{H}_{+1}$ created by the stabilizer constraints given in eq. 5.16.

As already explained, by defining the encoding in this way we preserve the locality and sparsity of the Hamiltonian terms. However, the main drawback of the superfast encoding is the fact that the stabilizer constraints are non-local in the way they are constructed and used as described in this section, since they involve a product of operators on any closed loop of undefined length.

A solution to this problem is to modify the original graph onto which we apply the encoding to obtain a new graph G', and then define the stabilizer constraints on a cycle basis of this new graph. If this cycle basis then only contains constant-length cycles, and if each edge in G' only occurs in a constant number of cycles, we obtain respectively local and sparse stabilizer constraints.

There are different ways of modifying the graph and constructing a cycle basis for it. In sections 5.3 and 5.4 we discuss two algorithms implementing this.

[3]The interested reader can find more information about quantum simulation and error detection in ref. [2], [8] and [9].

5.3. A stacking and sewing procedure

The idea of this algorithm is to first create a cycle basis for the graph G onto which we originally applied the superfast encoding, using another known algorithm. After this, we create as many copies of G as there are cycles in the cycle basis, and stack them on top of each other to create a new graph $\hat{G}$. Then finally, in each layer, we 'sew' the corresponding cycle of the cycle basis if their length is larger than some threshold, such that we obtained a cycle basis consisting of only small constant-length cycles. Together with this, every edge will be used a constant number of times. The original graph G will be an induced subgraph of $\hat{G}$, which means that the graph algebra of G is a subalgebra of the graph algebra of $\hat{G}$, and thus we still obtain a valid mapping of the fermionic Hamiltonian to a qubit Hamiltonian [6].

The algorithm is given in the proof of the Lemma given in Appendix B of ref. [6]. We present a slightly modified version of the proof below:

Lemma 5.3.1. *Consider a connected graph G on n vertices with degree $\mathcal{O}(1)$. There exists a (polynomial time construable) connected graph $\hat{G}$ on $\mathcal{O}(n^2)$ vertices with G as an induced subgraph, that has a cycle basis consisting of cycles of length at most 4 and which uses every edge in $\hat{G}$ at most 4 times.*

Proof.

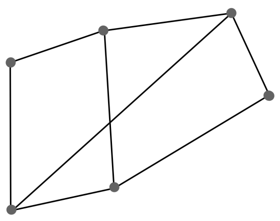
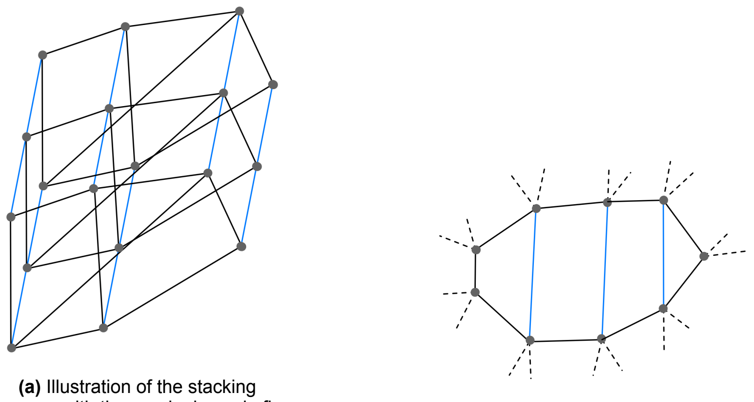


Figure 5.5: The graph $G = (V, E)$ consisting of six vertices that we use as an example.



(a) Illustration of the stacking process with the graph shown in fig. 5.5. The blue edges are the new edges added in between copies of the graph.

(b) Illustration of the sewing process applied to a cycle of length 9. The blue edges are again added in the process.

Figure 5.6: Illustration of the stacking- and sewing process.

We will explicitly construct the graph $\hat{G}$. First, compute a cycle basis C for G by for instance using the Spanning Tree algorithm or Horton's algorithm. We know from sec. 3.1 that this cycle basis has $|E(\hat{G})| - |V(\hat{G})| + c$ elements, where c is the number of connected components. For a connected graph, we have $c = 1$. Furthermore, since G has bounded degree, it is sparse, and thus we have $|E| = \mathcal{O}(|V|)$. Using this, we see that the cycle basis has $|E(\hat{G})| - |V(\hat{G})| + 1 = \mathcal{O}(n)$ elements.

Order the cycles in the basis C in some arbitrary way. We now construct the graph $\hat{G}$ as follows. For each cycle in C, we make a copy of the graph G and we "stack" these copies on top of each other. We do this by connecting each vertex in a certain copy to the corresponding vertex in the copies directly above and below (see fig. 5.6a for an illustration, using the graph shown in fig. 5.5). In this way, we create $|E(G)|\,(|C|-1)$ vertical cycles which have length 4, since the top and bottom layer are only connected to one other layer.

These vertical cycles form an independent set, since for every pair of vertical cycles, both cycles contain an edge that is not present in the other cycle. Furthermore, the set of cycles C is still an independent set in $\hat{G}$, since nothing has changed for them. Furthermore, the union of these two sets is also independent, which can be seen using the same argument as for the vertical cycles. In particular, this union forms a cycle basis for the graph $\hat{G}$, since the size of any cycle basis of $\hat{G}$ (i.e., $|E(\hat{G})| - |V(\hat{G})| + 1$) is equal the number of vertical cycles plus the dimension of the cycle space of G. This can be seen by directly calculating both values.

Continuing the construction, consider for each cycle in the set C the corresponding copy of the graph G. In this copy, we "sew" the cycle by adding edges across the cycle as illustrated in fig. 5.6b. For each cycle ζ in C this creates smaller cycles of length 3 or 4. In this sewing process, we add at most $\lceil\zeta\rceil/2$ edges to $\hat{G}$. In Theorem 4.4 of Ref. [24] it is shown that for any sparse graph there exists a cycle basis with weight $\mathcal{O}(n\log(n))$. So we see that we add at most $\mathcal{O}(n\log(n))$ edges in the sewing process. After sewing all the cycles in C, the construction of the graph $\hat{G}$ is completed. Note that the degree of the graph $\hat{G}$ is at most three higher than the degree of G, since to each vertex, we add two extra edges in the stacking process and at most one extra edge in the sewing process.

We now construct the following cycle basis for the graph $\hat{G}$. We take all vertical cycles, and all the small cycles created after applying the sewing process to each cycle in C. As explained before, the union of the vertical cycles and the cycle basis of G was a basis for $\hat{G}$ before the sewing process. Since the sewing process only adds independent cycles to the basis, the resulting set of cycles is also a basis for $\hat{G}$.

It can be easily seen that every cycle in this cycle basis has length at most 4. Furthermore, every edge occurs in at most a constant number of cycles in the basis. The vertical edges appear in a number of vertical cycles bounded by the degree of G. The edges in each copy of the graph participate only in the small cycles created in the sewing process plus at most two vertical cycles. So we see that the number of cycles in which a given edge occurs is bounded. This can be mathematically formulated as:

$$\text{sparsity} = \max\{4, d_f\}, \tag{5.22}$$

where the sparsity is equal to the maximal number of times that an edge is used in the obtained cycle basis. Furthermore, d_f is the degree of the original graph G, and the 4 comes from the edges in the obtained stacked graph $\hat{G}$ that are not part of a sewed cycle. So we see that if the degree d_f is $\mathcal{O}(1)$, then we also have a sparsity scaling as $\mathcal{O}(1)$. □

So, following the steps described in the proof of this Lemma, we obtain a graph $\hat{G}$ with the original graph G as an induced subgraph, and we obtain a cycle basis of $\hat{G}$ such that the locality and sparsity of the cycles in the basis (and thus of the stabilizer constraints following from the superfast encoding) are $\mathcal{O}(1)$.

However, the graph $\hat{G}$ is of size $\Theta(n^2)$, which is not beneficial for quantum simulations. So while this algorithm to construct a cycle basis, in combination with the Bravyi-Kitaev superfast encoding, may seem ideal at first site when looking at the locality and sparsity of the fermionic Hamiltonian and stabilizer constraints, it still has its disadvantages.

We will now discuss another algorithm for constructing a cycle basis, which is given in the Decongestion Lemma of Freedman-Hastings.

5.4. The Freedman-Hastings Decongestion Lemma

In this section, we present a way of obtaining a cycle basis of the original graph G by using the Freedman-Hastings Decongestion Lemma shown in Appendix B in [12]. This Lemma is based on an algorithm that provides a weakly fundamental cycle basis for the graph G onto which the encoding is applied, where each edge $e \in E$ is only used in a limited number of cycles in the cycle basis. This result of the Lemma can be used to make sure that the stabilizer constraints arising from the Bravyi-Kitaev superfast encoding can be made sparse while also limiting the required number of qubits.

We now present the Decongestion Lemma together with a slightly modified version of the proof as compared to the proof given in Appendix B in [12]:

Lemma 5.4.1 (**Freedman-Hastings Decongestion Lemma**). *Given a graph $G = (V, E)$ with the degree of each vertex $\mathcal{O}(1)$. There exists a weakly fundamental cycle basis in which each edge appears in at most $\mathcal{O}(\log(|V|)^2)$ cycles in the basis.*
Furthermore, there is an efficient randomized algorithm, given in the proof of this lemma, to construct such a basis. If we assign non-negative weights to each edge of the graph, then with probability $\Omega(1)$ the algorithm returns a cycle basis of total weight bounded by $\log(|V|)$ times the sum of edge weights of the graph.
Furthermore, suppose the graph is sparse, and say that two cycles in the basis intersect if they share at least one vertex. Then, each cycle in the constructed basis intersects at most polylogarithmically many cycles later in the basis.

Proof.
The proof is based on the following recursive and randomized algorithm A. We let $A(G)$ denote the cycle basis returned with a graph G as input. If G has no edges, then $A(G) = \emptyset$. The algorithm consists of the following steps:

1. If graph G has at least one vertex with degree 1, let v be an arbitrary vertex with degree equal to 1. Define G' as the graph with only that vertex removed. Return $A(G')$[4].
2. Else, if the graph G has at least one vertex v with degree 2, let v be an arbitrary vertex with degree equal to 2. There are two possibilities. **Case A**: If v has a self-edge e, let C be the cycle consisting of that self-edge. Remove this edge from G to obtain a graph G'. Return $\{C\} \cup A(G')$. **Case B**: If v does not have a self-edge, then v has edges to two other vertices x, y. Remove v from G and add an edge (x, y) to obtain a graph G'. Compute $A(G')$. Then, for each cycle in the resulting cycle basis, replace every occurrence of edge (x, y) with $(x, v), (v, y)$ and return this as $A(G)$.
3. Else, find a simple cycle C of length at most $\mathcal{O}(\log(|V|))$ in G. Let G' be the graph obtained from G by removing an edge of that cycle, choosing that edge uniformly at random. Return $C \cup A(G')$.

We first prove that this algorithm always returns a weakly fundamental cycle basis for G. After this we show that we can always find a cycle of length $\mathcal{O}(\log(|V|))$ in step 3. Then we use the algorithm to show that there exists a cycle basis where each edge appears at most $\mathcal{O}(\log(|V|)\log(|E|))$ times, after which we will prove the remaining claims.

We use induction to prove that the algorithm always returns a cycle basis. We assume that it returns a cycle basis for all graphs with $V' < V$ vertices and for all graphs with V vertices and $E' < E$ edges. We will use this to prove it holds for all graphs with V vertices and E edges. The base cases $V = 0$ or $V > 0, E = 0$ are trivial. Then we assume the statement to be true for a graph on $V' < V$ vertices or a graph on V vertices with $E' < E$ edges. Under this assumption, the statement will then also hold for any other graph with the assumed properties because the algorithm is recursive. Note that if we are in step 1, then v does not participate in any cycle, and so a cycle basis for G' gives a cycle basis for G. Furthermore, after step 2A or step 3 where we add a cycle C to the basis, we remove an edge in C. It is because of this that the algorithm returns a weakly fundamental cycle basis, since this removed edge cannot be present in any other cycles later in the basis.

To show that we can always find a cycle with length $\mathcal{O}(\log(|V|))$ in step 3, note that by assumption all vertices have degree at least 3. Hence, starting from any given vertex there are more than 2^l paths of length l starting at that vertex, since for each edge of this path with length l we have 3 vertices to which we can connect the edge. So, for $l \sim \log(|V|)$ there must be two different paths with the same endpoint following from a pigeonhole argument, since we have $|V|$ vertices in the graph and $2^{\log(|V|)}$ vertices in the path. This implies that we can always find a simple cycle of length $\mathcal{O}(\log(|V|))$.

We move on by providing a bound on the number of cycles in which some edge e appears in a cycle basis returned by the algorithm. The algorithm is recursive, starting with graph G and defining a

[4] In this algorithm we use the notation G' to indicate the graph obtained after applying the modifications in the algorithm. Note however that this graph will not be equal to the graph onto which the superfast encoding will be applied, since the algorithm of Freedman-Hastings constructs a cycle basis of the original graph G.

sequence of graphs $G', G'', G''', \dots$. Let $G_0 = G, G_1 = G', G_2 = G'', \dots$. Note that when the algorithm constructs G_{j+1} from G_j, the set of edges of G_{j+1} is some subset of the edges of G_j, possibly with an extra edge added after step 2B. We call an edge f in G_{j+1} the *child* of an edge e in G_j if f is the same as edge e or if f is obtained in case 2B, where $e = (x, v)$ or $e = (v, y)$ and $f = (x, y)$. An edge f in G_j is the *descendant* of an edge e in $G = G_0$ if there is a sequence $e_0 = e, e_1, e_2, \dots, e_j = f$ with e_{k+1} being the child of e_k for $k = 0, 1, \dots, j-1$. Note that if this descendant exists, then it is unique. Not every edge e has a descendant in G_j, because some edges are removed. We say that an edge $e \in G$ is removed on step j if e has a descendant in G_{j1} but not in G_j.

Then, for any edge $e \in G$, the number of cycles in which e appears in the cycle basis is equal to the number of times that the algorithm constructs a cycle C containing a descendant of e. If on the j-th step, we construct a cycle containing a descendant of e using step 2A of the algorithm, then e is removed on step j. So we see that the number of times that e appears in the cycle basis is bounded by one plus the number of times that the algorithm uses step 3 and constructs a cycle C containing a descendant of e.

However, each time case 3 occurs, the descendant is removed with probability $\Omega(1/\log(|V|))$, since we remove edges from the simple cycles of length $\Omega(1/\log(|V|))$ uniformly at random. So, the probability that an edge e appears in at least w such cycles is bounded by

$$\left[1 - \Omega\left(\frac{1}{\log(|V|)}\right)\right]^w. \tag{5.23}$$

As will become clear, we want to find for what w we can make this probability strictly smaller than $1/2|E|$. So after setting this $< 1/2|E|$ and taking the logarithm on both sides, we obtain:

$$w \cdot \log\left[1 - \Omega\left(\frac{1}{\log(|V|)}\right)\right] < -\log(2|E|). \tag{5.24}$$

Now we use the approximation $\log(1-x) \approx -x$ for small x to obtain:

$$w \cdot -\Omega\left(\frac{1}{\log(|V|)}\right) < -\log(2|E|). \tag{5.25}$$

And from this it follows that the inequality is satisfied for some $w = \mathcal{O}(\log(|V|)\log(|E|))$. So for this w we obtain $\left[1 - \Omega\left(\frac{1}{\log(|V|)}\right)\right]^w < \frac{1}{2|E|}$ as the probability that an edge e appears in at least w cycles. Now we take the union bound over all edges to see that all edges appear in at least w cycles with probability $< |E| \cdot \frac{1}{2|E|} = 1/2$. So from this, we see that no such edge exists with probability $\geq 1/2$.

For a dense graph we have $|E| = \mathcal{O}(|V|^2)$, but since we consider sparse graphs in this thesis we have $|E| = \mathcal{O}(|V|)$. So we see that with probability $\geq 1/2$ no edge exists in at least $\log(|V|)\log(|E|) = \mathcal{O}(\log(|V|)^2)$ cycles.

If G has multi-edges or self-edges, we can still show that the claim above holds. To obtain this, we must first modify the graph in some way before running the algorithm. First, remove the self-edges and replace every multi-edge with a single edge to obtain a graph $\tilde{G}$ with V vertices and $|\tilde{E}| = \mathcal{O}(|V|^2)$ edges. Then construct a cycle basis for $\tilde{G}$ using the algorithm. Then we know that every edge appears at most $\mathcal{O}(\log(|V|)^2)$ times in this cycle basis. Now, if an edge in $\tilde{G}$ connects two vertices which have multiple edges between them in G, replace each occurrence of that edge in the cycle basis with an arbitrary one of those multi-edges in G. Then, add every self-edge of G as a cycle to the cycle basis. Also, if two vertices u, v in G have multiple edges between them, labelled $e_1, e_2, \dots, e_k$ for some k, add the cycle $e_j e_{j+1}^1$ for each $j = 1, \dots, k-1$ to this cycle basis. The result is a cycle basis for G with the given property.

The algorithm A is efficient, since it is called at most $|E|$ times in the recursion, since we remove an edge in each step. Also, each step can be done efficiently. For example, a shortest cycle in the graph can be found efficiently by, for each edge in turn, considering the graph with that edge removed and finding a shortest path between the vertices which were endpoints of the edge. The algorithm finds a basis which satisfies the given properties with probability $\geq 1/2$, and the properties of the basis can be efficiently verified.

Now suppose that we add non-negative edge weights to each edge of the graph G. Then by linearity of the expectation, the expected cycle basis weight is equal to the sum over the edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.

Those cycles are constructed in step 3 of the algorithm. After constructing such a cycle with length $\mathcal{O}(\log(|V|))$, we remove an edge from this cycle with probability $\mathcal{O}\left(\frac{1}{\log(|V|)}\right)$. This process can be seen as the number of Bernoulli trials that is needed to get one success, where a success is defined as removing the given edge after constructing a cycle. The probability of a success is equal to the probability of removing the given edge, so $p = \mathcal{O}\left(\frac{1}{\log(|V|)}\right)$. The random variable corresponding to the number of trials until one success follows a Geometrical distribution, which has expected value $\frac{1}{p} = \mathcal{O}(\log(|V|))$ [25]. So we see that the expected number of times that the algorithm constructs a cycle C containing a descendant of the given edge is $\mathcal{O}(\log(|V|))$. Hence, the expected cycle basis weight is bounded by the total edge weight times $\mathcal{O}(\log(|V|))$. So with probability $\Omega(1)$, the cycle basis weight is bounded by the total edge weight times $\mathcal{O}(\log(|V|))$. Since in this thesis we consider the edge weights to be equal to one and we furthermore have $|E| = \mathcal{O}(|V|)$ since we consider sparse graphs, we see that the cycle basis weight is $\mathcal{O}(|V|\log(|V|))$.

Finally, suppose that we consider a sparse graph G. Then the graph remains sparse while running the algorithm as the degree can only decrease. When the algorithm constructs a cycle in step 2A or 3 with some graph G' as its input, the cycle has length at most logarithmic in the number of vertices of G'. All cycles constructed later on contain each edge of G', and since the graph is sparse also each vertex, at most polylogarithmically many times. This is because every edge appears in at most $\mathcal{O}(\log(|V|)^2)$ cycles in the basis, and the expected number of times that the algorithm constructs a cycle later in the basis containing a descendant of that edge is equal to $\mathcal{O}(\log(|V|))$. So, we see that the cycle intersects only polylogarithmically many cycles later in the basis. □

For better understanding of the algorithm given in the proof of this Lemma, we have put an example of this algorithm in fig. A.4 in Appendix A.4.

One of the main properties of the algorithm given above is de Decongestion property, stating that each edge appears in at most $\mathcal{O}(\log(|V|)^2)$ cycles in the basis. This is exactly the sparsity of the obtained cycle basis. So, if we have n fermionic modes, then we note that the algorithm of Freedman-Hastings is expected to have a sparsity which scales as $\mathcal{O}(\log(n)^2)$.

This Decongestion and the property that every cycle intersects only polylogarithmically many cycles later in the basis, are two important properties of the algorithm of Freedman-Hastings that we want to use in developing the improved fermionic encoding in chapter 7.

Besides the sparsity, we also want to say something about the length of the cycles in the cycle basis, and thus the locality of the stabilizer constraints in the Bravyi-Kitaev superfast encoding. For this we consider the part of the Lemma 5.4.1 which says that the total weight of the obtained cycle basis is $\mathcal{O}(n\log(n))$. Looking at the algorithm, we note that all cycles of the cycles basis are developed in step 2A and step 3, and since the total number of edges in the graph is $\mathcal{O}(|V|)$, the total size of the cycle basis will be $\mathcal{O}(|V|)$. Combining this with the bound for the total weight of the cycle basis, there are different scenarios possible, with the most interesting one regarding the locality for our case is the scenario where all but one cycle have constant length. Then the length of the cycle with non-constant length can be $\mathcal{O}(|V|\log(|V|))$, since $\frac{\mathcal{O}(|V|\log(|V|))}{\mathcal{O}(1)} = \mathcal{O}(|V|\log(|V|))$. So in this case the locality of the cycle basis will be $\mathcal{O}(|V|\log(|V|))$.

So we note that using this algorithm, both the locality and the sparsity of the cycle basis are not expected to be bounded by a constant, which is in that sense worse than the algorithm described in sec. 5.3. But since the required number of qubits of the algorithm of sec. 5.3 is $\mathcal{O}(n^2)$ compared to $\mathcal{O}(n)$ for the algorithm of Freedman-Hastings, we would like to combine these two algorithms described in this chapter in some way in the improved fermionic encoding.

The hypothesis is then that we will be able to achieve constant locality and sparsity of both the qubit Hamiltonian and the stabilizer constraints, while keeping the required number of qubits to achieve this somewhere between $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$. We will elaborate further on this part in chapter 7.

Before we discuss the numerical results of applying the algorithms considered in the previous sections, we first discuss two physical systems onto which we will apply these algorithms.

5.5. Physical systems to consider

As briefly mentioned in sec. 3.2, we need to construct a graph onto which we can apply the fermionic encoding. In this section, we discuss two fermionic systems which we will map onto qubits, namely the Fermi-Hubbard model and so-called sparse SYK Hamiltonians.

In general, we define the interaction graph representing a fermionic system, as a (hyper)graph where the vertices correspond to the modes present in each term in the Hamiltonian, and for each term we draw edges between all modes present in that particular term. As we see in eq. 5.4, the Fermi-Hubbard model has terms consisting of only two modes, so for each term we draw one edge between two modes to construct the interaction graph. For the sparse SYK Hamiltonians, we will see that this is not the case and that we obtain hyperedges there.

An important remark is that the fermionic interaction graph is not necessarily the graph onto which the fermionic encoding will be applied. In the next two sections, we show how to construct the graph used for the encoding for both fermionic systems, first for the Fermi-Hubbard model and after that for the sparse SYK Hamiltonians.

5.5.1. The Fermi-Hubbard model on sparse hopping graphs

The Fermi-Hubbard model describes a physical system with the Hamiltonian given in eq. 5.4. This model is often used to describe a physical system in condensed matter physics, and as previously stated in sec. 3.2, the graph corresponding to this Hamiltonian onto which we will apply the encoding, will be a sparse random graph. This graph is a slightly modified version of the Erdos-Renyi random graph, where the degree of some vertices is artificially decreased below a certain threshold. In this graph we add edges between every pair of vertices with $p \propto \frac{1}{n}$, and we set the threshold for the degree of each vertex equal to 4. We will now show why we can use this graph to find the results of transforming the Fermi-Hubbard model using the Bravyi-Kitaev superfast encoding.

First we consider the on-site interaction terms. These consists of the terms made up of the number operators: $n_{j,\uparrow}n_{j,\downarrow}$. It can be shown that we can write a number operator in terms of the Majorana operators in the following way: $n_j = \frac{1}{2}(I - ic_{2j-1}c_{2j})$[5]. If we do this for both spin-states, we can write:

$$n_{j,\uparrow}n_{j,\downarrow} = \frac{1}{4}(I - ic_{2j-1,\uparrow}c_{2j,\uparrow})(I - ic_{2j-1,\downarrow}c_{2j,\downarrow}). \tag{5.26}$$

If we look at eq. 5.7 from the Bravyi-Kitaev superfast encoding then we see that we can write:

$$n_{j,\uparrow}n_{j,\downarrow} = \frac{1}{4}(I + B_{j,\uparrow})(I + B_{j,\downarrow}), \tag{5.27}$$

and thus from this we see that for representing the on-site interaction terms of the Fermi-Hubbard model we only need the vertex operators $B_{j,\sigma}$, which can be directly applied onto vertices in the graph.

We do something similar with the hopping terms by rewriting $a^\dagger_{j,\sigma}a_{k,\sigma} + a^\dagger_{k,\sigma}a_{j,\sigma}$ in terms of the edge and vertex operators. It turns out that we can write:

$$\begin{aligned} a^\dagger_{j,\sigma}a_{k,\sigma} + a^\dagger_{k,\sigma}a_{j,\sigma} &= -\frac{i}{2}(c_{2k,\sigma}c_{2j-1,\sigma} + c_{2j,\sigma}c_{2k-1,\sigma}) \\ &= \frac{i}{2}A_{jk,\sigma}(B_{j,\sigma} - B_{k,\sigma}). \end{aligned} \tag{5.28}$$

[5]All derivations of this section are shown in Appendix A.5).

From eq. 5.28 we see that we can represent the hopping terms using the vertex operators $B_{j,\sigma}$ and $B_{k,\sigma}$ and the edge operators $A_{jk,\sigma}$. This means that for a hopping term, we only need the vertex operators and the edge operators on the edge between the two vertices participating in that given term.

So this means that the graph onto which we will apply the fermionic encoding must consist of vertices and edges between several pairs of those vertices for each hopping term. This is exactly the way how we construct the random graph in sec. 3.2.

Furthermore, again note from eq. 5.27 that for the on-site interacting terms we only need the vertex operator for that given vertex, and we do not need an edge operator to describe this term in the Bravyi-Kitaev superfast encoding. That means that while the fermionic interaction graph representing the Fermi-Hubbard model consists of two spin-layers connected by edges, we do not need the edge operators between the vertices with different spin who are at the same site in the interaction graph.

Also, since the graphs representing both spin-layers are identical to each other, the cycle basis and the corresponding locality and sparsity of the stabilizer constraints will be identical for both layers. And thus, we only need to perform the calculations of the superfast encoding for one modified Erdos-Renyi graph, and then we can use these results to investigate the properties for both spin-layers of the fermionic interaction graph.

So we see that for the Fermi-Hubbard model, the graph onto which we will apply the encoding is equal to one spin-layer of the fermionic interaction graph representing the Hamiltonian given in eq. 5.4.

5.5.2. Sparse SYK Hamiltonians

We consider one more physical system, which is a little bit more complicated than the Fermi-Hubbard model. These systems are described by so called sparse Sachdev-Ye-Kitaev (SYK) Hamiltonians. These Hamiltonians representing are a sum of terms consisting of four Majorana operators [26]:

$$H = \sum_{S\subseteq[n],|S|=4} J_S c_{S(1)} c_{S(2)} c_{S(3)} c_{S(4)}. \tag{5.29}$$

Here $c_{S(j)}$ is a Majorana operator working on the j-th component of the set S, and the J_S are some coefficients which are i.i.d. Gaussians with $\mu = 0$ and $\sigma = 1$.

Looking at eq. 5.29, we note that these Hamiltonians consist of $\Theta(n^4)$ terms, instead of the $\Theta(n)$ terms that we need for the Hamiltonian to be sparse. To solve this issue, we go through all terms and we remove each term with probability $1-p$, for some $p \in [0,1]$ [26]. We note that while first having $\Theta(n^4)$ terms, this procedure leaves us with $\Theta(pn^4)$ terms. To obtain $\Theta(n)$ terms, we thus note that we need $p \propto \frac{1}{n^3}$, where n is again the number of fermionic modes. After applying this procedure of removing some terms, we obtain a sparse Hamiltonian with a high probability.

For the Fermi-Hubbard model we saw that one spin-layer of the fermionic interaction graph is equal to the graph onto which we apply the superfast encoding. But in this case we will show that the graph that will be used in the encoding of this system is not equal to the interaction graph representing the Hamiltonian given in eq. 5.29.

To construct the graph onto which we apply the encoding, we investigate which vertex and edge operators we need to represent each term in the Hamiltonian. If we for example consider a Hamiltonian term like $c_{2j-1}c_{2k-1}c_{2l-1}c_{2m-1}$, then using eq. 5.7 from the Bravyi-Kitaev superfast encoding we directly see that we can write this as:

$$c_{2j-1}c_{2k-1}c_{2l-1}c_{2m-1} = -A_{jk}A_{lm}, \tag{5.30}$$

and thus we only need operators on the edges (j,k) and (l,m) to represent this Hamiltonian term in the graph onto which we will apply the encoding.

We could also have a Hamiltonian term like $c_{2j-1}c_{2k-1}c_{2l}c_{2m}$, but using eq. 5.7 and the anti-commutation relations of the Majorana operators, we write this in terms of the edge and vertex operators as:

$$c_{2j-1}c_{2k-1}c_{2l}c_{2m} = -B_j A_{jk} B_k A_{lm}, \tag{5.31}$$

and thus we see that we also only need the vertices j, k, l, m and the edges (j, k) and (l, m) to represent this term in the graph onto which we will apply the encoding.

Another example of a term in the sparse SYK Hamiltonian is a term like $c_{2j-1}c_{2j}c_{2k-1}c_{2k}$. In this case we only have to consider the vertex operators B_j and B_k since, again using eq. 5.7, we directly see that we can write:

$$\begin{aligned} c_{2j-1}c_{2j}c_{2k-1}c_{2k} &= (iB_j)(iB_k) \\ &= -B_jB_k. \end{aligned} \tag{5.32}$$

So we see that in this case we don't have to add an edge to the graph onto which we will apply the encoding.

In general, the terms in the sparse SYK Hamiltonian are made up of Majorana operators on the vertices j, k, l, m. If we order the Majoranas in these terms in the correct way by using their anti-commutation relations, we see that to construct the graph onto which we will apply the encoding, we only have to add edges between the first two vertices and between the last two vertices, only between the first two vertices or the last two vertices, or that we do not have to add an edge at all depending on the number of different modes present in the term. So we see that for each term in the Hamiltonian described in eq. 5.29 we have to add at most four vertices and two edges to the graph.

And finally, just like we did for the random graph described in sec. 3.2, we artificially modify the resulting graph to obtain a degree of all vertices of $\mathcal{O}(1)$ by removing incident edges of a given vertex uniformly at random until the degree becomes lower than some threshold value, which is set to 4.

So looking at the construction of the graph, we see that this graph is different than the interaction graph representing the sparse SYK Hamiltonians. Furthermore, it is also different than the graph onto which we apply the encoding if we consider the Fermi-Hubbard model. Where we have to add one edge to the graph for each term in the Fermi-Hubbard Hamiltonian, we have to add zero, one, or two edges to the graph for each term in the sparse SYK Hamiltonian.

In chapter 6 we discuss the results of numerically implementing the Bravyi-Kitaev superfast encoding in combination with the stacking- and sewing-procedure from sec. 5.3 and the algorithm of Freedman-Hastings from sec. 5.4. We apply this encoding first to the modified Erdos-Renyi graph corresponding to the Fermi-Hubbard model, and after this we apply this encoding to the graph corresponding to the sparse SYK Hamiltonians.

6 Numerical results for known transformations

In this chapter we will discuss the results of the numerical implementation of the algorithm described in chapter 5. We apply the algorithms both to the modified Erdos-Renyi graph described in sec. 3.2 corresponding to one spin-layer of the Fermi-Hubbard model, and onto the graph corresponding to the sparse SYK Hamiltonians as described in sec. 5.5.2.

Throughout the rest of the thesis, we will use the Bravyi-Kitaev superfast encoding to map the fermionic modes onto qubits, and in this chapter we use the stacking- and sewing algorithm described in sec. 5.3 and the algorithm developed by Freedman-Hastings to modify the original graph G and construct a cycle basis to the newly obtained graph G', to govern the properties of the stabilizer constraints arising from the superfast encoding.

We are mainly interested in investigating the locality and sparsity of the qubit Hamiltonian terms and of the stabilizer constraints, and the total number of qubits used in the encoding.

1. The locality of the qubit Hamiltonian terms is bounded by the degree of the vertices in the obtained graph G' as described in sec. 5.2. Note that if we add edges to the graph in for example the stacking- and sewing-process, than the degree of the graph G', and thus the locality of the qubit Hamiltonian terms will be increased.
2. In a similar way, as described in 5.2, the sparsity of a given qubit also depends on the degree of the graph G'.
3. The locality of the stabilizer terms is equal to the maximum length of the cycles in the cycle basis of the obtained graph G'.
4. The sparsity of the stabilizer terms is equal to the maximum number of cycles in the basis in which a given edge appears, with the maximum taken over all edges in the graph G'.
5. Finally, the number of qubits is equal to the number of edges in the graph G'.

Now that we know what the relevant numerical results will be to investigate the properties of a given encoding, we will discuss the results of the numerical implementation. We will first discuss the results of applying the encoding to the modified Erdos-Renyi graph corresponding to the Fermi-Hubbard model, followed by the results for the graph corresponding to the sparse SYK Hamiltonians.

6.1. Results for the modified Erdos-Renyi graph

The results of the numerical implementation of the algorithms on the modified Erdos-Renyi graph are shown in fig. 6.1 - fig. 6.4.

From the proof of Lemma 5.4.1, we expect the sparsity of the cycle basis obtained from the algorithm of Freedman-Hastings to be $\mathcal{O}(\log(n)^2)$, because of the Decongestion property. In fig. 6.1 we see that this bound seems to be satisfied for the graph that we investigate.

Furthermore, following the results shown in fig. 6.1, it also seems to be plausible that the locality of the cycles basis obtained by the algorithm of Freedman-Hastings is $\mathcal{O}(\log(n))$. This behaviour is also plotted in fig. 6.2b, where the average length of cycles in the cycle basis obtained from the algorithm of Freedman-Hastings is plotted against $\log(n)$. In this plot the data points eventually seems to lie on a straight line, indicating that the average cycle length is indeed $\mathcal{O}(\log(n))$. This scaling of the average cycle length implies that there must be at least one cycle of non-constant length in the cycle basis.

From fig. 6.1 we can conclude that our expectations regarding the performance of the different algorithms for constructing a cycle basis were true: the Spanning Tree algorithm performs the worst in both the locality and the sparsity, followed by the algorithm of Freedman-Hastings and the stacking- and sewing-procedure delivers the best results with both the locality and the sparsity being $\mathcal{O}(1)$.

One of the drawbacks of the stacking- and sewing-procedure is visualized in fig. 6.2a. Here the average degree of the obtained interaction graph is shown, and it can be seen that the degree of the vertices in the stacked graph G' is on average the degree of the original interaction graph G plus 2. This also matches our expectation, since the extra edges connected to a vertex $v \in G'$ are the vertical edges connecting vertices in different layers. This higher average degree has the consequence that the locality of the qubit Hamiltonian terms for the stacking- and sewing-procedure will be 2 higher than when using the algorithm of Freedman-Hastings.

Fig. 6.3a shows another drawback of the stacking- and sewing-procedure. In this figure the number of edges in the graph G' obtained from this algorithm (and thus the number of qubits) is shown, and as expected, this number is $\mathcal{O}(n^2)$. In fig. 6.3b it can be seen that after applying the algorithm of Freedman-Hastings, the number of edges is $\mathcal{O}(n)$, since this algorithm is applied to the original graph G.

Figures 6.4b and 6.4a can be used to verify another statement made in the proof of Lemma 5.4.1 of Freedman-Hastings. Namely, in fig. 6.4a we see that we indeed have a cycle basis of size $\mathcal{O}(n)$.

In fig. 6.4b we see that the statement made that the total weight is bounded by the total edge weight times $\mathcal{O}(\log(n))$ seems to be true for this type of graph. We consider the edge weights to be equal to one in this thesis and since the number of cycles in the basis is $\mathcal{O}(n)$, we see that the statement mentioned above implies that we expect the total weight of the basis to be $\mathcal{O}(n\log(n))$. In fig. 6.4b we plot the total weight of the basis versus $n\log(n)$ and the data points seems to lie on a straight line, so we conclude that this statement seems to be true for this type of graph.

Based on these results following from the numerical implementation, we can conclude that the algorithm of Freedman-Hastings and the stacking- and sewing-procedure perform well in different categories. The stacking- and sewing procedure obtains a cycle basis where the locality and sparsity are $\mathcal{O}(1)$, but the average degree of the obtained graph G' (and thus the locality of the qubit Hamiltonian terms) is the average degree of the original graph G plus 2, and the number of edges (and thus number of qubits) is $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n)$ for the algorithm of Freedman-Hastings.

So we see that none of these algorithms exhibit all desired properties if applied to the modified Erdos-Renyi graph. In the next chapter, we will combine these two methods to obtain an improved encoding. Before moving on to the construction of this encoding, we first discuss the numerical results of applying the known algorithms to the graph corresponding to the sparse SYK Hamiltonians.

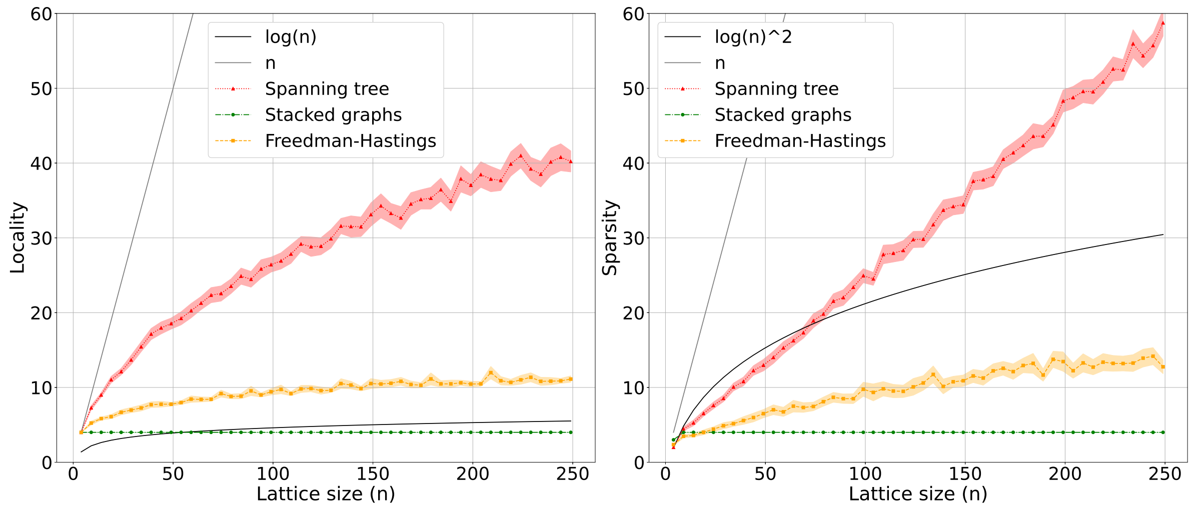
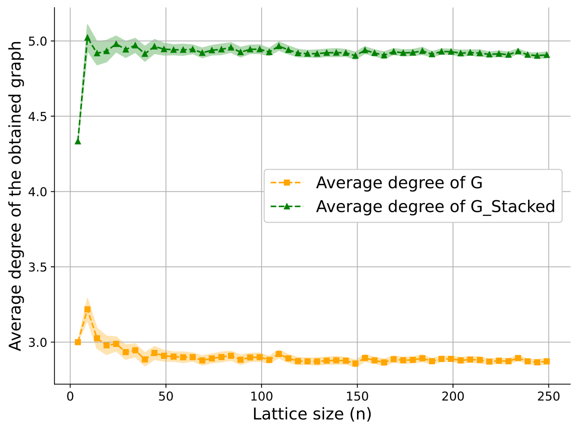
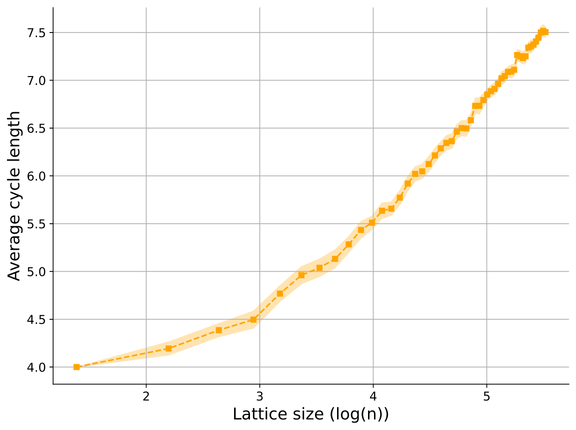


Figure 6.1: Locality (left) and sparsity (right) of the stabilizer constraints for the modified Erdos-Renyi graph, using different types of algorithms to construct a cycle basis for the Erdos-Renyi graph. We have plotted the Spanning Tree algorithm (red), the stacked-graphs algorithm described in sec. 5.3 (green) and the algorithm developed by Freedman-Hastings (orange) vs. the number of fermionic modes n. In the plot for the locality we have also plotted the function $\log(n)$, and in the plot for the sparsity the functions n and $\log(n)^2$. These curves can be used to compare how the locality/sparsity of the different algorithms scale with the number of fermionic modes n.

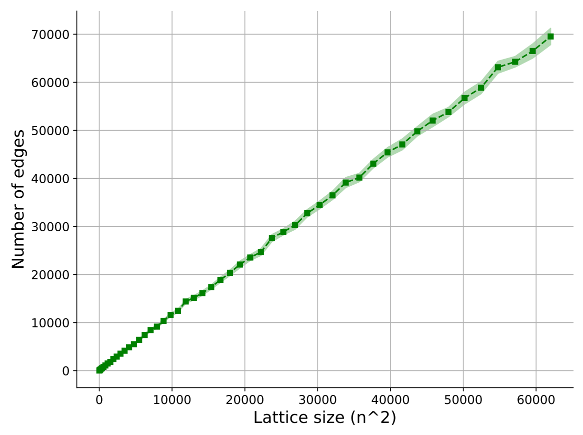


(a) The average degree of the vertices in the original Erdos-Renyi graph (left) and the obtained graph from the stacking and sewing procedure (right), plotted vs. n.

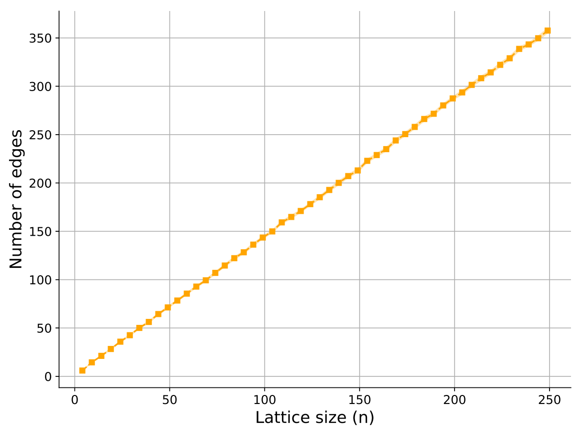


(b) The average length of cycles in the cycle basis for the Erdos-Renyi graph, obtained from the algorithm of Freedman-Hastings and plotted vs. $\log(n)$.

Figure 6.2: The average degree of the Erdos-Renyi graph (left) and average cycle length of the cycles in the cycle basis (right) for the Erdos-Renyi graph, obtained from the algorithm of Freedman-Hastings (right).

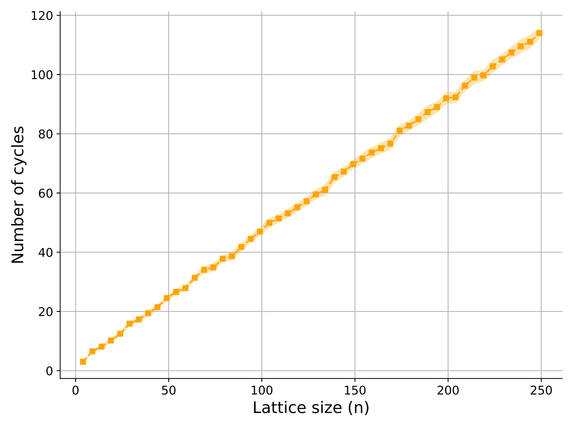


(a) Number of edges in the graph G' obtained from the stacking- and sewing procedure described in sec. 5.3, plotted vs. n^2.

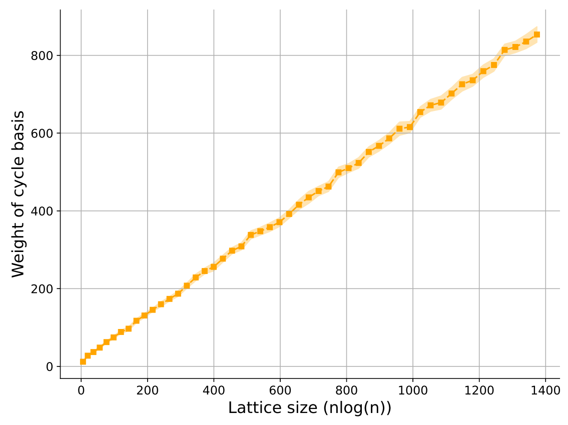


(b) Number of edges in the original Erdos-Renyi graph G, the graph onto which the algorithm of Freedman-Hastings is applied, plotted vs. n.

Figure 6.3: The number of edges in the finally obtained interaction graph after applying the the stacking- and sewing-procedure (left) and the algorithm of Freedman-Hastings (right). Eventually, the number of edges in the graph will be equal to the number of qubits in the fermionic encoding, following from the Bravyi-Kitaev superfast encoding.



(a) Size of the cycle basis for the Erdos-Renyi graph, obtained from the algorithm of Freedman-Hastings.



(b) Total weight of the cycle basis for the Erdos-Renyi graph, obtained from the algorithm of Freedman-Hastings.

Figure 6.4: The size (left) and total weight (right) of the cycle basis for the Erdos-Renyi graph, obtained from the algorithm of Freedman-Hastings. The size is equal to the number of cycles in the cycle basis. The total weight is defined as the sum of the length of all cycles in the cycle basis. For the algorithm of Freedman-Hastings, we saw in sec. 5.4 that this equals the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle c containing a descendant of the edge.

6.2. Results for the sparse SYK graph

In this section we present the numerical results of applying the different algorithms discussed in chapter 5 applied to the graph corresponding to the sparse SYK Hamiltonians as discussed in sec. 5.5.2.

The results in all figures presented below seem to follow the same trends as for the modified Erdos-Renyi graph, so the arguments given in sec. 6.1 also apply to this type of graph.

So, based on the numerical results presented in these sections, we can conclude that the algorithms from chapter 5 seem to be applicable to various types of graphs with bounded degree.

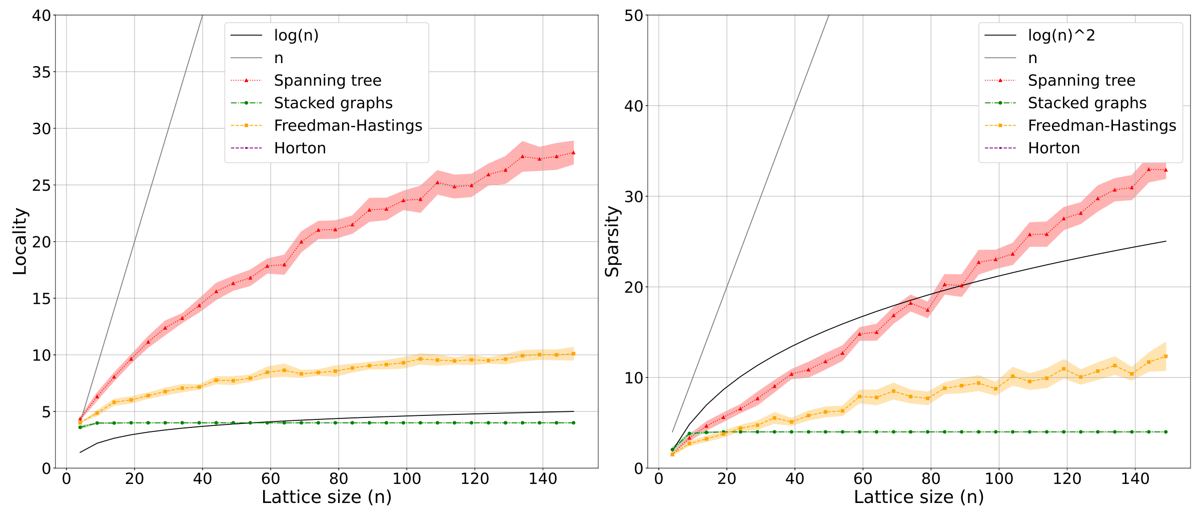
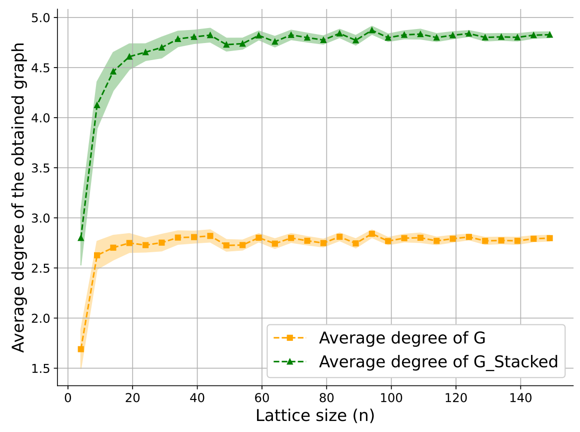
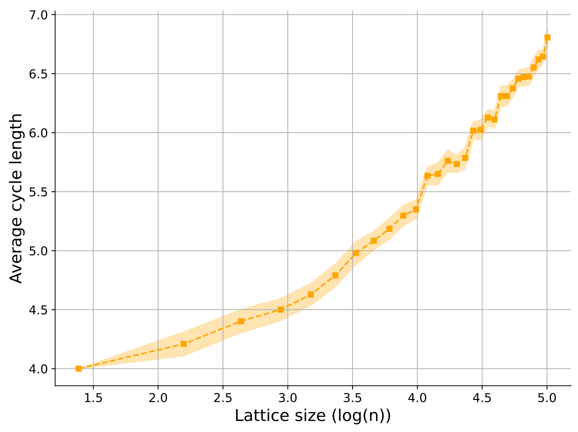


Figure 6.5: Locality (left) and sparsity (right) of the stabilizer constraints for a sparse SYK graph, using different types of algorithms to construct a cycle basis. We have plotted the Spanning Tree algorithm (red), the stacked-graphs algorithm described in sec. 5.3 (green) and the algorithm developed by Freedman-Hastings (orange) vs. the number of fermionic modes n. In the plot for the locality we have also again plotted the function $\log(n)$, and in the plot for the sparsity the functions n and $\log(n)^2$.

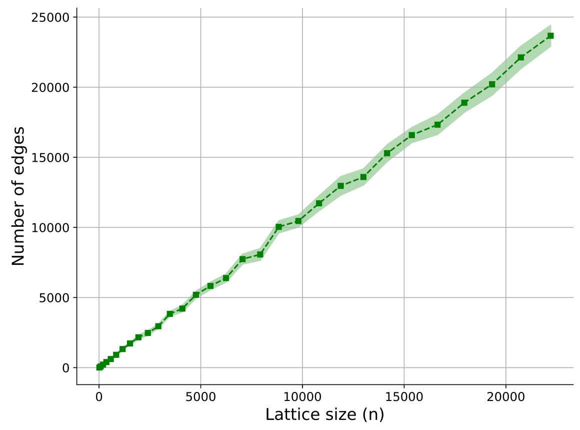


(a) The average degree of the vertices in the original sparse SYK graph (left) and the obtained graph from the stacking and sewing procedure (right), plotted vs. n.

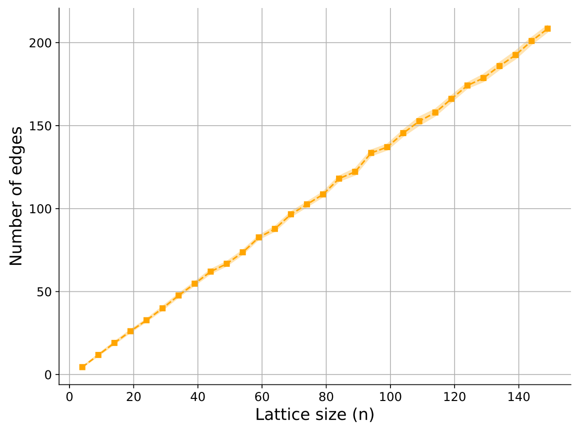


(b) The average length of cycles in the cycle basis for a sparse SYK graph, obtained from the algorithm of Freedman-Hastings, plotted vs. $\log(n)$.

Figure 6.6: The average degree of the graph (left) and average cycle length of the cycles in the cycle basis obtained from the algorithm of Freedman-Hastings (right).

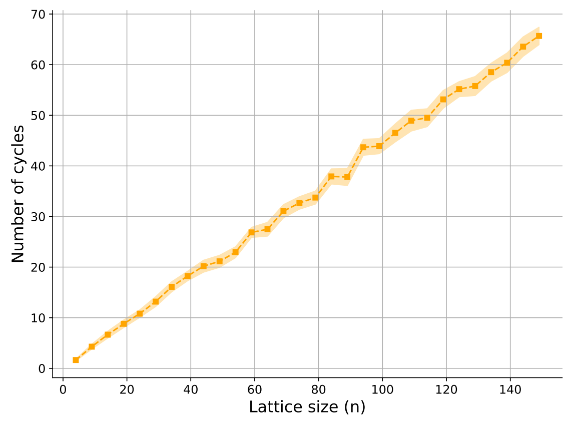


(a) Number of edges in the graph G' obtained from the stacking- and sewing procedure described in sec. 5.3, plotted vs. n^2.

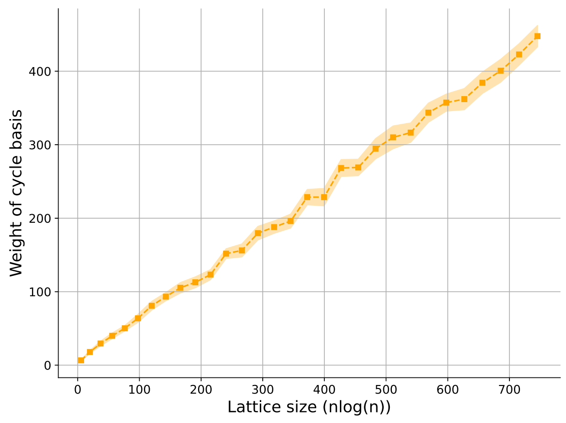


(b) Number of edges in the original sparse SYK graph G, so the graph on which the algorithm of Freedman-Hastings is applied, plotted vs. n.

Figure 6.7: The number of edges in the finally obtained interaction graph after applying the the stacking- and sewing-procedure (left) and the algorithm of Freedman-Hastings (right). Again, the number of edges in the graph will be equal to the number of qubits in the fermionic encoding.



(a) Size of the cycle basis for a sparse SYK graph, obtained from the algorithm of Freedman-Hastings.



(b) Total weight of the cycle basis for a sparse SYK graph, obtained from the algorithm of Freedman-Hastings.

Figure 6.8: The size (left) and total weight (right) of the cycle basis for a sparse SYK graph, obtained from the algorithm of Freedman-Hastings. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle c containing a descendant of the edge, and the size is equal to the number of cycles in the cycle basis.

7 Constructing the improved fermionic encoding

Now it is time to turn to the main part of this thesis, namely the construction of the improved fermionic encoding. We will consider the Fermi-Hubbard model and the sparse SYK Hamiltonian respectively described in sec. 5.5.1 and sec. 5.5.2, and we will derive an improved fermionic encoding which maps these Hamiltonians onto local and sparse qubit Hamiltonians, with sparse and local stabilizer constraints, while minimizing the required number of qubits.

The main idea is to use the sewing and stacking algorithm described in sec. 5.3, together with the algorithm of Freedman-Hastings described in sec. 5.4 and a yet to be described Vertex Coloring algorithm (which we will sometimes denote by ”VC”), to obtain a local and sparse cycle basis of the graph G' used for the encoding, where the qubits are placed at the edges of the graph just as in the Bravyi-Kitaev superfast encoding described in sec. 5.2.

The search for an improved fermionic encoding can be divided into two different scenarios. In the first scenario, we are interested in keeping the locality and sparsity of both the Hamiltonian terms and the stabilizer constraints as low as possible. To obtain this, we have to compromise in the total number of qubits. In the second scenario, we are interested in keeping the total number of qubits as low as possible, while allowing an increase in the locality and sparsity of the Hamiltonian terms and the stabilizer constraints.
In this chapter, we will first discuss the different scenarios in more detail, and after that we will construct the improved encoding for both scenarios. A formal proof of all properties of this improved encoding is given in chapter 9.

7.1. Two scenarios

As said before, the improved fermionic encoding depends on what one eventually wants to achieve. In this thesis, we consider two different scenario’s:

1. Getting the locality and sparsity of the qubit Hamiltonian terms and of the stabilizer constraints to be $\mathcal{O}(1)$ with a minimal pre-factor, while keeping the required number of qubits $\mathcal{O}(npoly(\log(n))$.
2. Increasing the locality and sparsity of the qubit Hamiltonian terms and stabilizer constraints for the benefit of the required number of qubits. This scenario can be divided into two sub scenarios:
   (a) Getting the required number of qubits to be $\mathcal{O}(n)$ at the expense of letting the locality of the stabilizer constraints to superconstant, by direct application of the Freedman-Hastings algorithm.
   (b) Getting the required number of qubits to be $\mathcal{O}(npoly(\log(n))$ with a reduced pre-factor compared to scenario 1.

Table 7.1 summarizes the results that we find for each scenario. Note that these are worst-case bounds, the most interesting results of numerically applying this improved encoding are shown in chapter 8.

	VC with no overlap	Freedman-Hastings	VC with small overlap
Locality qubit Hamiltonian terms	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(poly(\log(n)))$
Sparsity qubit Hamiltonian terms	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(poly(\log(n)))$
Locality stabilizer constraints	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Sparsity stabilizer constraints	$\mathcal{O}(1)$	$\mathcal{O}(poly(\log(n)))$	$\mathcal{O}(poly(\log(n)))$
Number of qubits	$\mathcal{O}(npoly(\log(n)))$	$\mathcal{O}(n)$	$\mathcal{O}(npoly(\log(n)))$ with a reduced pre-factor

Table 7.1: Bounds for the different properties of the improved fermionic encoding for each of the scenarios described in this section.

In each scenario, the general improved fermionic encoding will be modified a little bit to match the goals that we want to achieve. This consists of modifying the Vertex Coloring algorithm, as well as applying different kinds of sewing methods.

7.2. The Vertex Coloring algorithm

In the improved fermionic encoding, we use a different kind of algorithm to set up the stacking- and sewing process than the procedure described in sec. 5.3. In sec. 5.3, we sew only one cycle for each layer, causing the need for $\Theta(n^2)$ qubits. We develop a Vertex Coloring algorithm which enables us to sew multiple cycles in each layer, while also achieving constant locality and sparsity for both the Hamiltonian terms and stabilizer constraints. This causes the bound for the required number of qubits to drop to $\mathcal{O}(npoly\log(n))$ as we will show.

The idea behind the Vertex Coloring algorithm is the following:

1. For a given graph G, create a cycle basis using the algorithm of Freedman-Hastings.
2. Create a new graph $\hat{G}$ where the vertices correspond to the cycles in the obtained cycle basis for G.
3. For every pair of vertices $u, v \in \hat{G}$, draw an edge between them if the corresponding cycles overlap, which means that they share at least one vertex in the original graph G.
4. Now assign colors to each vertex in $\hat{G}$ in such a way that two vertices that share an edge do not have the same color. In order to construct this coloring we create a partition R of the cycles in the cycle basis obtained in step 1, using the property of the algorithm of Freedman-Hastings that every cycle intersect polylogarithmically many cycles later in the basis. This partition is constructed such that each subset R_i only contains non-overlapping cycles, and we then assign the same color to all vertices corresponding to cycles in the same subset R_i.
5. For each subset R_i make a copy of the original graph G and connect the layers with vertical edges to obtain a graph G'.
6. In each layer of the graph G', sew the cycles contained in the corresponding set R_i to obtain only constant-length cycles.
7. The final cycle basis consists of the small cycles obtained in step 7, together with the vertical cycles created by the edges connecting the different layers of the graph G'.

To create the coloring and partition of step 4 and 5, we use the following algorithm [27]:

1. Start with the ordered cycle basis $C = [c_1, \ldots, c_N]$ as obtained from the algorithm of Freedman-Hastings.
2. Initialize $R_i = \emptyset$. Then while $R \neq \emptyset$, do:

   (a) Enumerate C in reverse order, starting at the last cycle c_N in C, to get cycles c_j for $j = N, \dots, 1$.
   (b) If c_j overlaps with any cycle in R_i, set $j = j - 1$. Otherwise, add c_j to R_i and set $j = j - 1$.
   (c) Remove the cycles in R_i from R and set $i = i + 1$.

3. Return the computed partition $R = \bigcup_{i=1}^{t} R_i$.

The improved fermionic encoding then consists of using the Bravyi-Kitaev superfast encoding to map the fermionic operators to Pauli operators as described in eq. 5.14, which creates a valid mapping to a local and sparse qubit Hamiltonian when combined with the stabilizer constraints given in eq. 5.16. Then we use the Vertex Coloring algorithm to create a new graph G' which has the original graph G as a induced subgraph and develop a cycle basis for this such that the stabilizer constraints also become local and sparse.

Together with this, the Vertex Coloring algorithm in combination with the algorithm described in 1 - 3 enables us to use fewer layers than the algorithm described in sec. 5.3, such that the bound for the number of qubits will drop to $\mathcal{O}(n poly \log(n))$ as we will show in chapter 9.

Depending on which of the two scenarios we consider, we can modify the Vertex Coloring to match our goals. For scenario 1 where we are interested in keeping the locality and sparsity of the Hamiltonian terms and the stabilizer constraints as low as possible, we will use the Vertex Coloring algorithm as described above where no overlap is allowed between cycles that are sewed in the same layer. In this way, the sparsity of the edges of each layer and the degree of the vertices in the graph G' remain $\mathcal{O}(1)$, but we compromise a bit in the number of qubits since we need more layers than in the case where we admit some overlap.

This immediately introduces the modification needed for the second scenario, where we want to keep the number of qubits as low as possible while still keeping the locality of the stabilizer constraints $\mathcal{O}(1)$. For this scenario, we allow some overlap between the cycles sewed in the same layer, first on only one vertex and after that on at most two or three vertices. As expected, this will enable us to use even less layers in the stacking procedure and thus reducing the required number of qubits even further.

An important remark for this modification is that the average degree of the obtained graph G' can get to $\mathcal{O}(poly(\log(n)))$ in the worst case, compared to $\mathcal{O}(1)$ in the case where we allow no overlap. This is because it could be the case that one vertex in the graph is connected to $\mathcal{O}(poly(\log(n)))$ cycles in the cycle basis that are being sewed in the same layer, to which we will add $\mathcal{O}(poly(\log(n)))$ edges in the sewing process as shown in fig. 7.1. This increase in the average degree causes both the locality and sparsity of the qubit Hamiltonian terms to be $\mathcal{O}(poly(\log(n)))$ instead of $\mathcal{O}(1)$, since both depend on the degree of the graph as explained in sec. 5.2.

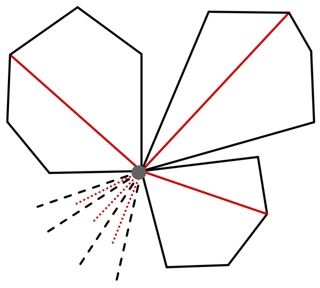


Figure 7.1: A given vertex (gray) which is part of $\mathcal{O}(poly(\log(n)))$ cycles and to which an edge is added for every cycle in the sewing process (shown by the red lines). The black and red dotted lines respectively represent more cycles connected and edges added to the given vertex.

This also causes the sparsity of the stabilizer constraints to be $\mathcal{O}(poly(\log(n)))$ instead of $\mathcal{O}(1)$. This is because the vertical edges between the different layers of the graph occur in the vertical cycles created by the edges connected to the vertices of that vertical edge in each layer.

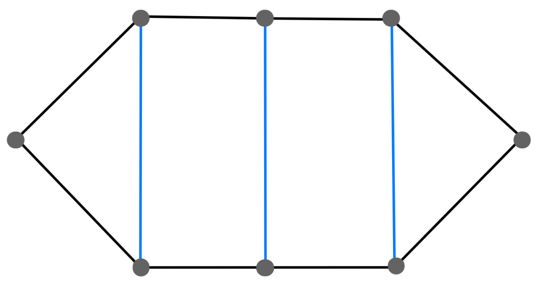
Besides modifying the Vertex Coloring for each of the two scenarios, we have also developed different types of methods to sew up the cycles in each layer of the stacked graph G'.

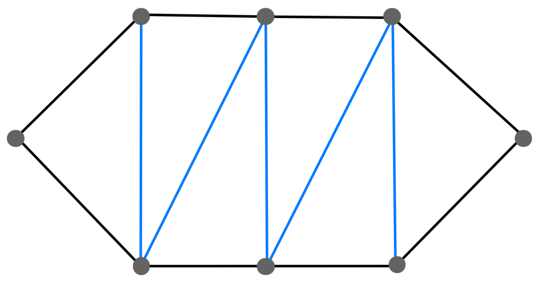
7.3. Different sewing methods

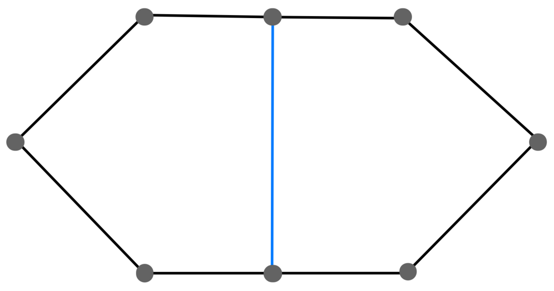
In this thesis, we consider four different types of sewing methods:

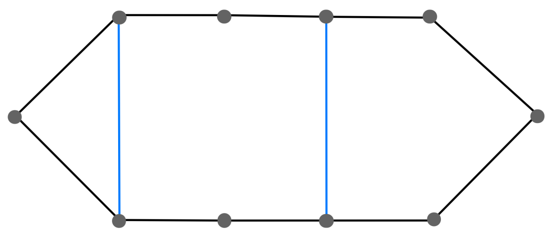
1. Straight sewing.
2. Triangular sewing.
3. Straight sewing which obtains cycles of length at most 6, from now on called the straight-6 sewing.
4. Skew sewing.

In fig. 7.2 all sewing methods are shown when applied to a cycle of length 8 (and length 10 in fig. 7.2d). Note that these methods could also be applied to a cycle of odd length.


(a) Straight sewing.


(b) Triangular sewing.


(c) Straight-6 sewing applied to a cycle of length 8.


(d) Straight-6 sewing applied to a cycle of length 10.

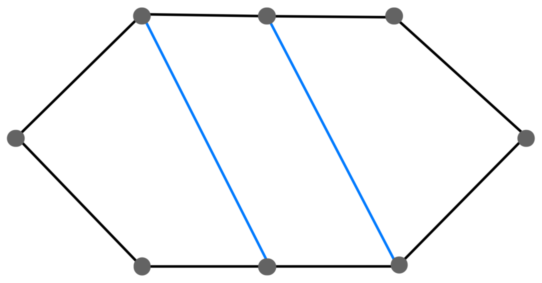

(e) Skew sewing

Figure 7.2: Different methods to sew a cycle.

In table 7.2 we summarize the properties of each sewing method. Here we let l denote the cycle length. Furthermore, note that the locality of the stabilizer constraints is given by the maximum length of the cycles in the cycle basis, and the locality and sparsity of the Hamiltonian terms are determined by the degree of the vertices in the obtained graph G'. In the second row of table 7.2, we indicate the increase in the degree of the vertices after adding edges in the sewing process, and thus we consider the increase in the locality and sparsity of the qubit Hamiltonian terms here.

The proof of one these properties is shown in Appendix A.6, the rest can be proven in a similar way.

	Straight sewing	Triangular sewing	Straight-6 sewing	Skew sewing
Locality stabilizer constraints	3 or 4	3	3, 5 or 6	4
Locality/sparsity Hamiltonian terms	+1	+1 or +2	+1	+1
Number of qubits (l even)	$\frac{l-2}{2}$	$l-3$	$\lfloor\frac{l-2}{4}\rfloor$	$\frac{l-2}{2}-1$
Number of qubits (l odd)	$\frac{l-3}{2}$	$l-3$	$\lfloor\frac{l-3}{4}\rfloor$	$\frac{l-3}{2}$

Table 7.2: Properties of the different sewing methods.

We will use the properties shown in table 7.2 in the process of constructing an improved fermionic encoding with these different types of sewing methods.

7.4. Combining everything

Now that we have discussed the possible modifications that we can apply to the Vertex Coloring algorithm and the different possible sewing methods, we can combine everything to obtain a list of different algorithms that we will test for both scenarios. The possible algorithms are the following:

1. Using the straight sewing where there is no overlap allowed between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.
2. Using the straight sewing where there is an overlap allowed on at most 1 vertex.
3. Using the straight sewing where there is an overlap allowed on at most 2 vertices.
4. Using the straight sewing where there is an overlap allowed on at most 3 vertices.
5. Using the triangular sewing where there is no overlap allowed between cycles in the same layer.
6. Using the skew sewing where there is no overlap allowed between cycles in the same layer.
7. Using the straight-6 sewing where there is an overlap allowed on at most 1 vertex.
8. Using the straight-6 sewing where there is an overlap allowed on at most 2 vertices.
9. Using the straight-6 sewing where there is an overlap allowed on at most 3 vertices.

All of these algorithms are used to construct an improved cycle basis compared to the ones described in chapter 5, where the cycle basis obtained from the algorithm of Freedman-Hastings is used as a starting basis. Then this obtained cycle basis is used to improve the locality and sparsity of the stabilizer constraints following from the Bravyi-Kitaev superfast encoding compared to using the algorithm of Freedman-Hastings, and also reduce the number of qubits compared to the stacking- and sewing-procedure described in sec. 5.3.

So the finally obtained improved fermionic encoding will consist of the Bravyi-Kitaev superfast encoding in combination with one of the algorithms shown above.

8 Numerical results for the improved fermionic encoding

In this chapter, we discuss the results of numerically applying the improved fermionic encoding constructed in chapter 7 to the modified Erdos-Renyi graph corresponding to one spin-layer of the Fermi-Hubbard model as described in sec. 5.5.1. Using the construction described in sec. 5.5.2, one could also numerically apply the improved encoding to the graph corresponding to the sparse SYK Hamiltonians.

In this chapter we will first discuss the general results of the numerical implementation, after which we will discuss the results for both scenarios described in sec. 7.1.

8.1. General results

In this section we show the general results after applying the different versions of the Vertex Coloring algorithm together with the different sewing methods onto the modified Erdos-Renyi graph. In fig. 8.1 - 8.4 we respectively show the locality and sparsity of the stabilizer constraints, the qubit-to-mode ratio and the average degree of the vertices in the obtained graph G' for all different methods.

From fig. 8.1 and fig. 8.2 we see that we will have an improvement in respectively the locality and the sparsity of the stabilizer constraints for all methods compared to the algorithm of Freedman-Hastings, and that the normal stacking- and sewing-procedure described in sec. 5.3 respectively achieves the second-to-lowest locality and the lowest sparsity of the stabilizer constraints together with some version of the Vertex Coloring algorithm.

In fig. 8.3 we see that the reverse holds for the total number of qubits: after using the algorithm of Freedman-Hastings we obtain the lowest qubit-to-mode ratio of $\mathcal{O}(1)$. When using the stacking- and sewing-procedure described in sec. 5.3 we obtain a ratio of $\mathcal{O}(n^2)$, which is the worst for all algorithms considered.

And finally, in fig. 8.4 we see that the algorithm of Freedman-Hastings achieves the lowest average degree of the vertices in the obtained graph G' and thus the locality and sparsity of the qubit Hamiltonian terms. This is because this algorithm does not modify the graph, causing the average degree to be equal to the average degree of the original graph G. All other methods modify the graph G in such a way that the average degree is increased.

Another important remark regarding fig. 8.4 is that it is hard to conclude whether we see the result that the average degree of vertices in the obtained graph G' can grow up to $\mathcal{O}(poly(\log(n)))$ if we allow some overlap in the Vertex Coloring algorithm. It could be the case that for example the average degree of the graph obtained after allowing an overlap on 3 vertices could grow up to $\mathcal{O}(poly(\log(n)))$, but that cannot be concluded with certainty from the numerical results.

What we can conclude is that the average degrees for all methods are eventually somewhat concentrated within the same band. This is a positive result considering the locality and sparsity of the qubit

Hamiltonian terms, because we see that we are able to reduce the number of qubits by allowing some overlap, but the locality and sparsity of the Hamiltonian terms do not seem to increase too much on average.

In sec. 8.2 and sec. 8.3 we discuss the numerical results in more detail for both scenarios and we determine which method delivers the best results for that given scenario.
All other relevant plots that are not displayed in this chapter are placed in Appendix B.

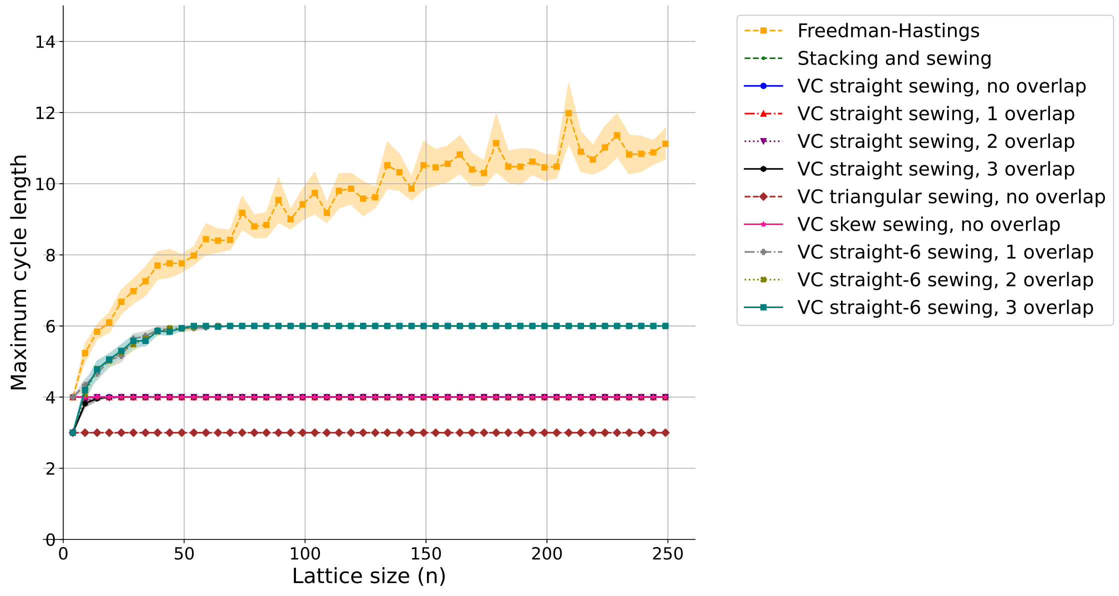


Figure 8.1: Locality of the stabilizer constraints for all different methods vs. the number of fermionic modes n.

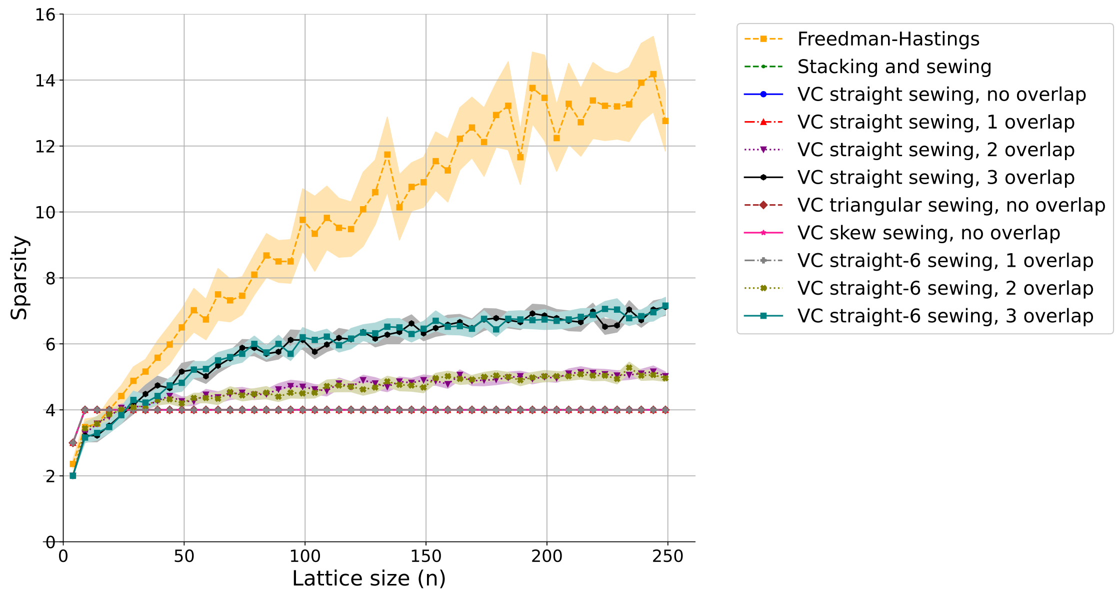


Figure 8.2: Sparsity of the stabilizer constraints for all different methods, vs. the number of fermionic modes n.

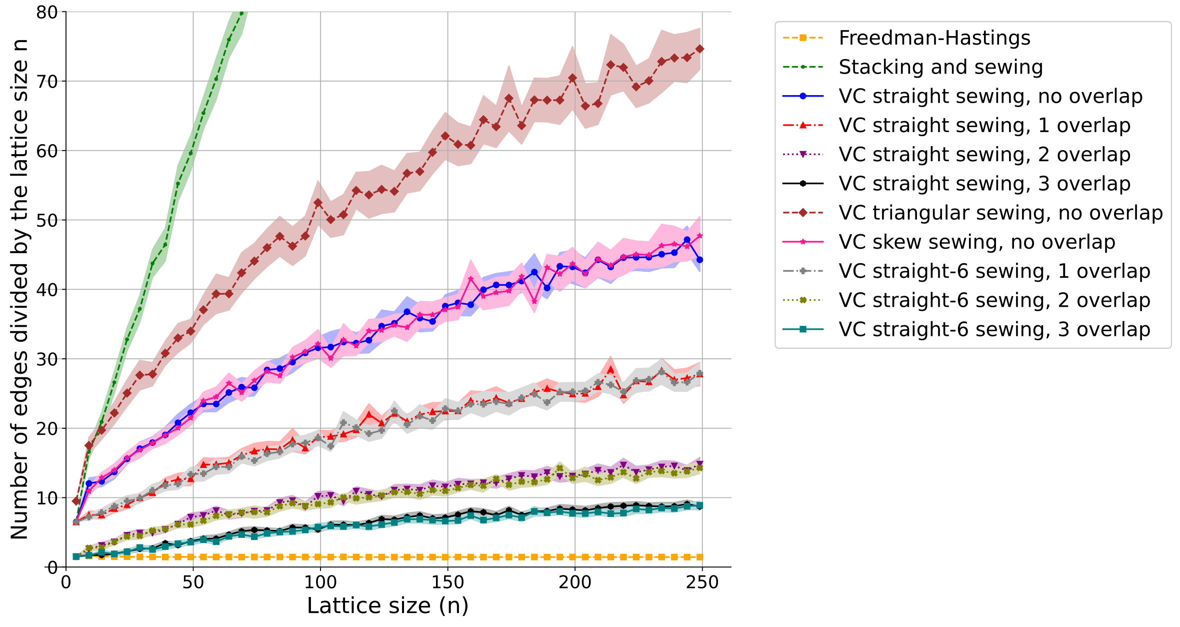


Figure 8.3: Number of edges divided by the number of fermionic modes n for all different methods, vs. the number of fermionic modes n.

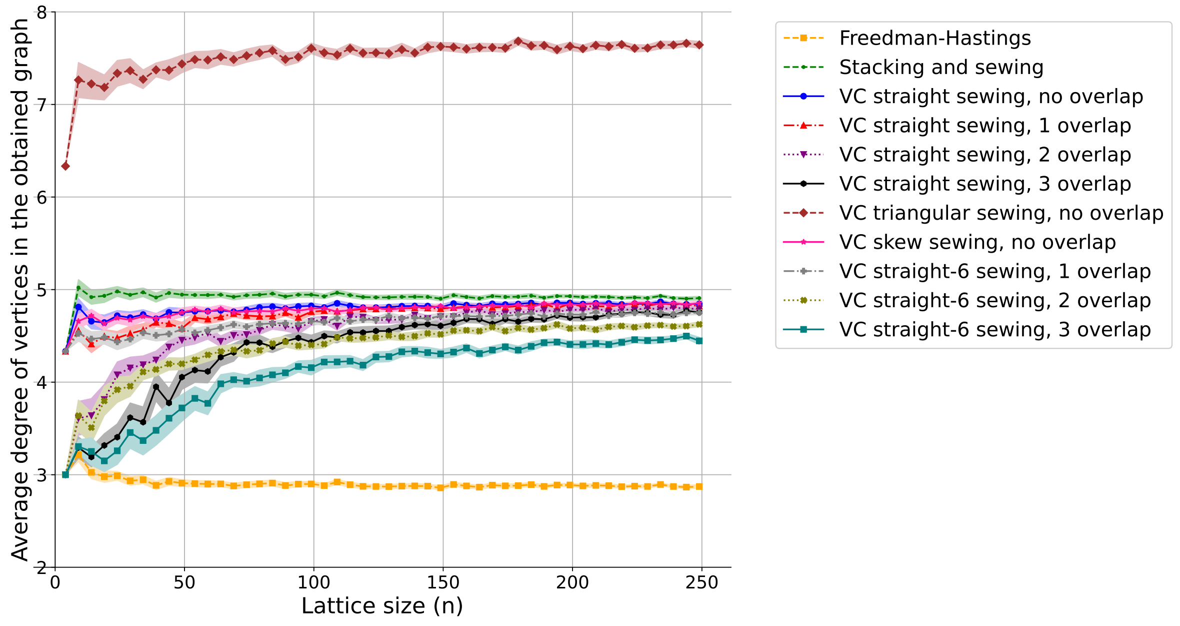


Figure 8.4: Average degree of the vertices in the obtained graph G' for all different methods, vs. the number of fermionic modes n. Note that the locality and sparsity of the qubit Hamiltonian terms are determined by the average degree of the vertices in G'.

8.2. Discussion for scenario 1: optimizing the locality and sparsity

In this scenario, we are interested in getting the locality and sparsity of both the Hamiltonian terms and the stabilizer constraints as low as possible, while compromising in the total number of qubits.

In fig. 8.1 and fig. 8.2 it can be seen that the locality and sparsity of the stabilizer constraints are both minimized when using the triangular sewing method where there is no overlap allowed in the Vertex Coloring algorithm. In this case, the locality and sparsity of the stabilizer constraints are resp. equal to 3 and 4. In fig. 8.3 we can see that the triangular sewing methods has the second to worst total number of qubits, so we see that we indeed have a trade off in terms of the number of qubits. However, looking at fig. 8.4, we see that the triangular sewing method has the worst average degree of the obtained graph G', causing the locality and sparsity of the qubit Hamiltonian to be the highest for all methods considered.

If we look again at fig. 8.1 and fig. 8.2, we see that for example the straight sewing method where there is an overlap allowed on at most 1 vertex, is the second best method for optimizing the locality and sparsity of the stabilizer constraints. For this method, both the locality and sparsity of the stabilizer constraints are equal to 4. In fig. 8.4 it can furthermore be seen that this method obtains a graph G' with average degree lower than for the triangular sewing method.

In fig. 8.3 we see that this method uses a lower amount of qubits than the sewing methods where we do not allow any overlap. We are not interested in keeping this number as low as possible in this scenario, but since all the other results are similar between this method and the ones where we do not allow any overlap, we can conclude that using the straight sewing method while allowing an overlap on 1 vertex in the Vertex Coloring algorithm, might be the best to consider in for scenario.

8.3. Discussion for scenario 2: optimizing the number of qubits

In this scenario, we want to keep the number of qubits as low as possible, while allowing an increase in the locality and sparsity of the qubit Hamiltonian terms and the stabilizer constraints.

In fig. 8.3 we see that after the algorithm of Freedman-Hastings, the algorithm which combines the straight or straight-6 sewing method with allowing an overlap on at most 3 vertices requires the lowest amount of qubits. In fig. 8.4 we see that this method also obtains a graph G' where the average degree of the vertices is not increased too much compared to other methods.

From fig. 8.1 we know that the locality of the stabilizer constraints after using straight sewing is equal to 4, where it is equal to 6 for straight-6 sewing. From fig. 8.2 it can however be seen that the sparsity of the stabilizer constraints for these methods is the highest after the algorithm of Freedman-Hastings. However, we still obtain an improvement in the sparsity of the stabilizer constraints compared to the algorithm of Freedman-Hastings.

If we look at the second best options in terms of the number of qubits, we see from fig. 8.3 that one could either choose the straight or straight-6 sewing method in combination with allowing a vertex overlap on at most 2 vertices. Again the locality of the stabilizer constraints for these methods is resp. equal to 4 and 6 and the sparsity of the stabilizer constraints is similar for both methods, but lower than in the case where we allow overlap on at most 3 vertices. Finally, from fig. 8.4 we can conclude that the average degree of the vertices in the obtained graph G' might seem to be a little bit lower for the straight-6 method compared to the straight sewing method.

So, if one does not care about the increase in sparsity of the stabilizer constraints, we conclude that combining the straight or straight-6 sewing method with allowing an overlap on at most 3 vertices in the Vertex Coloring algorithm is the best method to use for this scenario. If one is prepared to compromise a little bit in the number of qubits, to bound the increase in the sparsity of the stabilizer constraints, we conclude that combining the straight-6 sewing method with allowing an overlap on at most 2 vertices is the best method.

Now that we have constructed an improved fermionic encoding for both scenarios, we will formally prove all properties of this encoding in the next chapter.

9

Formal proof of the properties of the improved fermionic encoding

We start with the proof of the most important property of the improved fermionic encoding, namely that the number of qubits needed for the mapping is $\mathcal{O}(npoly(\log(n)))$. For both versions of the Vertex Coloring, the same argument applies so we will only prove this once.

The total number of qubits consists of three terms, namely it is the sum of the number of edges in all the layers of the graph, the number of vertical edges and the number of edges added in the sewing process. We determine the bounds for all these terms separately:

1. # of edges in the layers $= |E(G)| \cdot$ # of layers.
2. # of vertical edges $= |V(G)| \cdot$ (# layers $- 1)$.
3. $$\begin{aligned}\text{\# of sewing edges} &= \text{total weight of the cycle basis of Freedman-Hastings } \cdot \text{ constant}\\ &= \mathcal{O}(npoly(\log(n))) \cdot \text{constant}\\ &= \mathcal{O}(npoly(\log(n))).\end{aligned}$$

   Here we used for the first equality that the number of sewing edges per cycle is equal to a constant times the length of that cycle, and that the total weight of the cycle basis obtained from Freedman-Hastings is $\mathcal{O}(npoly(\log(n)))$ as has already been shown in sec. 5.4.

From this, we see that if we can prove that the number of layers needed to construct the graph G' is $\mathcal{O}(poly(\log(n)))$, we obtain:

1. # of edges in the layers $= |E(G)| \cdot$ # of layers $= \mathcal{O}(npoly(\log(n)))$.
2. # of vertical edges $= |V(G)| \cdot$ (# layers $- 1) = \mathcal{O}(npoly(\log(n)))$.
3. # of sewing edges $= \mathcal{O}(npoly(\log(n)))$.

And then the total number of qubits will be $3 \cdot \mathcal{O}(npoly(\log(n))) = \mathcal{O}(npoly(\log(n)))$, which gives the desired result[1].

So we need to prove that we only need $\mathcal{O}(poly(\log(n)))$ layers in the Vertex Coloring algorithm. Since in each layer we sew all cycles that correspond to vertices with the same color, the number of layers needed is equal to the number of different colors used in the Vertex Coloring algorithm. Recalling the construction of the vertex coloring described in step 4, we see that this is equal to the number of sets R_i making up the partition $R = \bigcup_{i=1}^{t} R_i$.

Thus we need to prove that the algorithm described in 1 - 3 obtains a partition $R = \bigcup_{i=1}^{t} R_i$, such that each R_i contains non-overlapping cycles and that it terminates after at most $\mathcal{O}(poly(\log(n)))$ steps.

[1]To be complete, it needs to be proven that all three bounds hold at the same time, since the results regarding the total number of layers and the total weight of the cycle basis of Freedman-Hastings do not necessarily both hold at the same time. Using a union bound, it can be proven that both results hold at the same time with probability $\Omega(1)$.

This proof mostly uses results gathered from [27], and furthermore uses the property of the algorithm of Freedman-Hastings that every cycle in the basis intersects only $\mathcal{O}(poly(\log(n)))$ other cycles later in the basis.

Proof.
By construction, we see that each R_i only contains non-overlapping cycles. We will use induction to prove that after i partitions computed, every cycle c remaining in R overlaps with at most $\mathcal{O}(poly(\log(n))) - i$ many cycles later in R. We assume the base case with $i = 0$ holds. We will show that the statement holds at step $i+1$. At step $i+1$, we have computed the partition R_{i+1}. For every cycle c_j in $R \setminus R_{i+1}$, there must exists a $c_{j'} \in R_{i+1}$ with $j' > j$ such that c_j overlaps with $c_{j'}$. Because if not, then c_j would be included in R_{i+1}. Therefore, the amount of cycles later in R overlapping with c_j must decrease by at least 1 after step $i+1$. This implies that after $poly(\log(n))$ steps there is only 1 cycle left that needs to be added to some set in the partition. So we see that the algorithm terminates after $poly(\log(n)) + 1 = \mathcal{O}(poly(\log(n)))$ steps. □

Thus, we see that we need to add $\mathcal{O}(poly(\log(n)))$ layers in the Vertex Coloring algorithm, implying that the total number of qubits required for this algorithm is $\mathcal{O}(npoly(\log(n)))$.

Now recall table 7.1 presenting the results of each scenario:

	VC with no overlap	Freedman-Hastings	VC with small overlap
Locality qubit Hamiltonian terms	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(poly(\log(n)))$
Sparsity qubit Hamiltonian terms	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(poly(\log(n)))$
Locality stabilizer constraints	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Sparsity stabilizer constraints	$\mathcal{O}(1)$	$\mathcal{O}(poly(\log(n)))$	$\mathcal{O}(poly(\log(n)))$
Number of qubits	$\mathcal{O}(npoly(\log(n)))$	$\mathcal{O}(n)$	$\mathcal{O}(npoly(\log(n)))$ with a reduced pre-factor

Table 9.1: Bounds for the different properties of the fermionic encoding for each of the scenarios described in sec. 7.1.

The results for the algorithm of Freedman-Hastings are proven in sec. 5.4 where we describe the algorithm. Furthermore, the results for the locality and sparsity of the qubit Hamiltonian terms for the Vertex Coloring algorithm without overlap is proven in sec. 5.2 where we describe the Bravyi-Kitaev superfast encoding. For the version with overlap, these results are proven in sec. 7.2, together with its results for the sparsity of the stabilizer constraints.

The results regarding the locality of the stabilizer constraints for both versions of the Vertex Coloring algorithm, and the sparsity of the stabilizer constraints for the version without overlap, are proven in sec. 5.3. This is because these results are similar to the results of the stacking- and sewing-procedure from sec. 5.3. In the version of the Vertex Coloring algorithm without overlap, we sew multiple cycles in each layer, but since those cycles do not overlap, we do not increase the average degree of the graph (and thus the sparsity of the stabilizer constraints) any more than if we use the procedure from sec. 5.3. Also, since we still sew the cycles to obtain small cycles, the locality of the stabilizer constraints for both versions of the Vertex Coloring algorithm will be the same as for the procedure from sec. 5.3.

With the proof regarding the number of qubits presented at the start of this chapter, we see that we have proven all the results shown in table 9.1.

10 Discussion and further research

Previous research on fermionic encodings obtained a way to map fermionic modes onto qubits by placing a qubit at every edge of the graph used in the encoding. This is done in the Bravyi-Kitaev superfast encoding and it obtains local and sparse qubit Hamiltonian terms, which is needed to perform a quantum simulation without using too many resources.

From this mapping, certain stabilizer constraints arise which are used for the quantum simulation. To further minimally strain the quantum computer, these stabilizer constraints should also be made local and sparse by modifying the graph used in the encoding and constructing a cycle basis for it. Furthermore, this should all be done while keeping the required number of qubits as low as possible.

Previous developed algorithms for constructing such a cycle basis, are a stacking- and sewing procedure of the graph and the algorithm developed by Freedman-Hastings. They obtained respectively local and sparse stabilizer constraints while using $\mathcal{O}(n^2)$ qubits, and a number of qubits scaling as $\mathcal{O}(n)$ but a locality and sparsity of the stabilizer constraints respectively scaling as $\mathcal{O}(n)$ and $\mathcal{O}(poly(\log(n)))$. So neither of them exhibit all the desired properties of a fermionic encoding. This gave rise to the research question of this thesis:

> *How to obtain a fermionic encoding from a local and sparse fermionic Hamiltonian, to a local and sparse qubit Hamiltonian, while minimizing the required number of qubits?*

In this thesis, we have developed a new algorithm to modify the graph used in the encoding and to construct a cycle basis for it. This algorithm combines properties of the stacking- and sewing-procedure and the algorithm developed by Freedman-Hastings, together with a newly developed Vertex Coloring algorithm which enables us to sew multiple cycles in each layer of the stacked graph. We use the Bravyi-Kitaev superfast encoding and govern the locality and sparsity of the stabilizer constraints that arise from this with the constructed cycle basis.

In doing so, we have obtained a locality and sparsity of the qubit Hamiltonian terms and the stabilizer constraints scaling as $\mathcal{O}(1)$, while keeping the required number of qubits $\mathcal{O}(npoly(\log(n)))$. We also presented a scenario in which we numerically obtained an even further reduced number of qubits, still $\mathcal{O}(npoly(\log(n)))$ but with a reduced pre-factor compared to the first scenario. In this second scenario, we still obtain constant locality of the stabilizer constraints, and the locality and sparsity of the qubit Hamiltonian terms and sparsity of the stabilizer constraints then become $\mathcal{O}(poly(\log(n)))$.

We see that these results form an improvement to the results of using the Bravyi-Kitaev superfast encoding in combination with the original stacking- and sewing-procedure or the algorithm of Freedman-Hastings.

We have furthermore shown how to apply this improved fermionic encoding to graphs corresponding to the Fermi-Hubbard model and sparse SYK Hamiltonians, and we have discussed the numerical implementation of the algorithms considered in this thesis to both types of graphs.

So, we conclude that the fermionic encoding constructed in this thesis provides an improvement compared to other known encodings in the sense of preserving the locality and sparsity of the qubit Hamiltonian terms and the stabilizer constraints following from the Bravyi-Kitaev superfast encoding, while keeping the required number of qubits as low as possible. This encoding can be used to simulate certain fermionic systems on a quantum computer which has limited resources.

Possible future research can be done about further increasing the vertex overlap between the cycles that are being sewed in the same layer to investigate whether it is possible to achieve constant locality and sparsity of the qubit Hamiltonian terms and stabilizer constraints while keeping the total number of qubits $\mathcal{O}(n)$, or $\mathcal{O}(npoly(\log(n)))$ with an even further reduced pre-factor than achieved in this thesis. Furthermore, one can think of other ways to sew the cycles which will have different properties regarding the locality of the qubit Hamiltonian terms and stabilizer constraints. This may be used to achieve alternative goals in quantum simulation than those discussed in this thesis.

A final option for follow-up research could be to investigate whether the Decongestion Lemma of Freedman-Hastings still holds if the expected degree of the graph is constant, instead of the maximum degree. If that would be true, then it would not be needed to artificially modify the graph to bound the degree of all vertices below some threshold value. Some thinking about this has been done throughout this research, but no final conclusion has been drawn yet.

Bibliography

[1] P. Sutter, *What is the Standard Model, the subatomic physics theory that has been tested more than any other?* Sep. 2022. [Online]. Available: https://www.livescience.com/the-standard-model.

[2] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, 10th ed. Cambridge, UK: Cambridge University Press, 2010, ch. 2.

[3] J. Schneider and I. Smalley, *What is a qubit?* Feb. 2024. [Online]. Available: https://www.ibm.com/think/topics/qubit.

[4] C. Derby, "Compact fermion to qubit mappings for quantum simulation," Ph.D. dissertation, University College London, London, United Kingdom, 2022.

[5] K. Setia, S. Bravyi, A. Mezzacapo, and J. D. Whitfield, "Superfast encodings for fermionic quantum simulation," *Phys. Rev. Res.*, vol. 1, p. 033 033, 3 Oct. 2019. DOI: 10.1103/PhysRevResearch.1.033033. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.1.033033.

[6] Y. Herasymenko, A. Anshu, B. Terhal, and J. Helsen, *Fermionic hamiltonians without trivial low-energy states*, 2023. arXiv: 2307.13730 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2307.13730.

[7] S. B. Bravyi and A. Y. Kitaev, "Fermionic quantum computation," *Annals of Physics*, vol. 298, no. 1, pp. 210–226, 2002, ISSN: 0003-4916. DOI: https://doi.org/10.1006/aphy.2002.6254. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003491602962548.

[8] L. Clinton, J. Bausch, and T. Cubitt, "Hamiltonian simulation algorithms for near-term quantum hardware," *Nature Communications*, vol. 12, no. 1, Aug. 2021, ISSN: 2041-1723. DOI: 10.1038/s41467-021-25196-0. [Online]. Available: http://dx.doi.org/10.1038/s41467-021-25196-0.

[9] V. Havlíček, M. Troyer, and J. D. Whitfield, "Operator locality in the quantum simulation of fermionic models," *Phys. Rev. A*, vol. 95, p. 032 332, 3 Mar. 2017. DOI: 10.1103/PhysRevA.95.032332. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.95.032332.

[10] C. Derby, J. Klassen, J. Bausch, and T. Cubitt, "Compact fermion to qubit mappings," *Phys. Rev. B*, vol. 104, p. 035 118, 3 Jul. 2021. DOI: 10.1103/PhysRevB.104.035118. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.104.035118.

[11] C. Derby and J. Klassen, *A compact fermion to qubit mapping part 2: Alternative lattice geometries*, 2021. arXiv: 2101.10735 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2101.10735.

[12] M. Freedman and M. B. Hastings, *Building manifolds from quantum codes*, 2021. arXiv: 2012.02249 [math.DG]. [Online]. Available: https://arxiv.org/abs/2012.02249.

[13] Y. O. and, "Indistinguishable particles in quantum mechanics: An introduction," *Contemporary Physics*, vol. 46, no. 6, pp. 437–448, 2005. DOI: 10.1080/00107510500361274. eprint: https://doi.org/10.1080/00107510500361274. [Online]. Available: https://doi.org/10.1080/00107510500361274.

[14] D. J. Griffiths and D. F. Schroeter, *Introduction to Quantum Mechanics*. Cambridge University Press, Aug. 2018, ch. 5.1.1.

[15] D. Gijswijt, "The power of shaking hands," Delft University of Technology, Tech. Rep. [Online]. Available: https://diamhomes.ewi.tudelft.nl/~dgijswijt/papers/handshake.pdf.

[16] M. P. Radcliffe, *Cycle bases*, Lecture Notes, 2018. [Online]. Available: https://www.math.cmu.edu/~mradclif/teaching/241F18/CycleBases.pdf.

[17] C. Liebchen and R. Rizzi, “Classes of cycle bases,” *Discrete Applied Mathematics*, vol. 155, no. 3, pp. 337–355, 2007, ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2006.06.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X06003052.

[18] GeeksforGeeks, *Depth first search or DFS for a graph*, Mar. 2025. [Online]. Available: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/.

[19] GeeksforGeeks, *Time and space complexity of DFS and BFS algorithm*, Mar. 2024. [Online]. Available: https://www.geeksforgeeks.org/time-and-space-complexity-of-dfs-and-bfs-algorithm/.

[20] J. D. Horton, “A polynomial-time algorithm to find the shortest cycle basis of a graph,” *SIAM Journal on Computing*, vol. 16, no. 2, pp. 358–366, 1987. DOI: 10.1137/0216026. eprint: https://doi.org/10.1137/0216026. [Online]. Available: https://doi.org/10.1137/0216026.

[21] GeeksforGeeks, *Erdos Renyl Model (for generating Random Graphs)*, Jul. 2022. [Online]. Available: https://www.geeksforgeeks.org/erdos-renyl-model-generating-random-graphs/.

[22] G. A. Margulis, “Explicit constructions of graphs without short cycles and low density codes,” *Combinatorica*, vol. 2, no. 1, pp. 71–78, Mar. 1982, ISSN: 1439-6912. DOI: https://doi.org/10.1007/BF02579283.

[23] D. Gijswijt, “Algebra 1,” Delft University of Technology, Tech. Rep., 2021.

[24] T. Kavitha, C. Liebchen, K. Mehlhorn, *et al.*, “Cycle bases in graphs characterization, algorithms, complexity, and applications,” *Computer Science Review*, vol. 3, no. 4, pp. 199–243, 2009, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2009.08.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013709000483.

[25] Y. Huang, “Expected values of discrete random variables,” University of Chicago, Tech. Rep. [Online]. Available: https://www.stat.uchicago.edu/~yibi/teaching/stat234/2022/L04.pdf.

[26] P. Orman, H. Gharibyan, and J. Preskill, *Quantum chaos in the sparse syk model*, 2024. arXiv: 2403.13884 [hep-th]. [Online]. Available: https://arxiv.org/abs/2403.13884.

[27] Z. He, A. Cowtan, D. J. Williamson, and T. J. Yoder, *Extractors: Qldpc architectures for efficient pauli-based computation*, 2025. arXiv: 2503.10390 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2503.10390.

[28] K. Aardal, L. van Iersel, and R. Janssen, *Optimization: Lecture Notes AM2020 / IN4344*. 2023, ch. 10, Faculty of Electrical Engineering, Mathematics, and Computer Science.

A
Derivations and examples

A.1. Example using the Majorana operators

In this section, we provide an example using the Majorana operators defined in 2.12.

Suppose we have an operator $K = i^m \prod_{\substack{S\subseteq[n],\\ |S|=q,\\ q \text{ even}}} c_{S(1)}c_{S(2)}\cdots c_{S(q)}$, where m and n are some integers and $c_{S(i)}$ is a Majorana operator working on the ith element of our set S. The question is now what value of m do we need if we want K to be Hermitian, so $K^\dagger = K$. We first write out $K^\dagger$ using the properties described in 2.13:

$$\begin{aligned} K^\dagger &= \left(i^m \prod_{\substack{S\subseteq[n],\\ |S|=q,\\ q \text{ even}}} c_{S(1)}c_{S(2)}\cdots c_{S(q)} \right)^\dagger \\ &= (-i)^m \prod_{\substack{S\subseteq[n],\\ |S|=q,\\ q \text{ even}}} c^\dagger_{S(q)}c^\dagger_{S(q-1)}\cdots c^\dagger_{S(2)}c^\dagger_{S(1)} \\ &= (-i)^m \prod_{\substack{S\subseteq[n],\\ |S|=q,\\ q \text{ even}}} c_{S(q)}c_{S(q-1)}\cdots c_{S(2)}c_{S(1)} \end{aligned} \tag{A.1}$$

Now we need to swap the Majoranas to get the original K operator. Note that $c_{S(q)}$ needs to swap $q-1$ places to get to its original position, $c_{S(q-1)}$ needs $q-2$ swaps, $c_{S(q-2)}$ needs $q-3$ swaps, et cetera. So in total we need $\sum_{k=1}^{q-1} k$ swaps to get all the Majoranas to their original position. Using the well-known formula $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$, we see that we need $\sum_{k=1}^{q-1} k = \frac{q(q-1)}{2}$ swaps, so we pick up $\frac{q(q-1)}{2}$ minus signs in the process. Thus we get:

$$\begin{aligned} K^\dagger &= (-i)^m(-1)^{q(q-1)/2} \prod_{\substack{S\subseteq[n],\\ |S|=q,\\ q \text{ even}}} c_{S(1)}c_{S(2)}\cdots c_{S(q)} \\ &= i^m(-1)^{m+q(q-1)/2} \prod_{\substack{S\subseteq[n],\\ |S|=q,\\ q \text{ even}}} c_{S(1)}c_{S(2)}\cdots c_{S(q)} \end{aligned} \tag{A.2}$$

To obtain $K^\dagger = K$, we then note that we need $m + \frac{q(q-1)}{2} = 2l$ for some $l = 0, \pm1, \pm2, \dots$. We can reduce this modulo 2, and then we finally see that we need $m \equiv \frac{q(q-1)}{2} \bmod 2$ for K to be a Hermitian operator.

A.2. Commutation relations in the Bravyi-Kitaev superfast encoding

Here the derivation is shown of the commutation relations described in eq. 5.8-5.13:

$$\begin{aligned}
B_j^\dagger &= (-ic_{2j}c_{2j+1})^\dagger = ic_{2j+1}c_{2j} = -ic_{2j}c_{2j+1} = B_j,\\
A_{jk}^\dagger &= (-ic_{2j}c_{2k})^\dagger = ic_{2k}c_{2j} = -ic_{2j}c_{2k} = A_{jk},\\
B_j^2 &= (-ic_{2j}c_{2j+1})(-ic_{2j}c_{2j+1}) = c_{2j}c_{2j}c_{2j+1}c_{2j+1} = I,\\
A_{jk}^2 &= (-ic_{2j}c_{2k})(-ic_{2j}c_{2k}) = c_{2j}c_{2j}c_{2k}c_{2k} = I,\\
B_jB_k &= (-ic_{2j}c_{2j+1})(-ic_{2k}c_{2k+1}) = c_{2k}c_{2k+1}c_{2j}c_{2j+1} = B_kB_j,\\
A_{jk} &= -ic_{2j}c_{2k} = ic_{2k}c_{2j} = -A_{kj},\\
A_{ij}B_k &= (-ic_{2i}c_{2j})(-ic_{2k}c_{2k+1}) = (-ic_{2k}c_{2k+1})(-ic_{2i}c_{2j}) = B_kA_{ij}, \quad i,j \neq k,\\
A_{kj}B_k &= (-ic_{2k}c_{2j})(-ic_{2k}c_{2k+1}) = -(-ic_{2k}c_{2k+1})(-ic_{2k}c_{2j}) = -B_kA_{kj}, \quad i = k, j \neq k,\\
A_{ik}B_k &= (-ic_{2i}c_{2k})(-ic_{2k}c_{2k+1})\\
&= -(-ic_{2k}c_{2k+1})(-ic_{2i}c_{2k})\\
&= -B_kA_{ik}, \quad i \neq k, j = k,\\
A_{ij}A_{kl} &= (-ic_{2i}c_{2j})(-ic_{2k}c_{2l})\\
&= (-ic_{2k}c_{2l})(-ic_{2i}c_{2j}) = A_{kl}A_{ij}, \quad i,j \neq k,l,\\
i^5A_{ij}A_{jk}A_{kl}A_{lm}A_{mi} &= i(-ic_{2i}c_{2j})(-ic_{2j}c_{2k})(-ic_{2k}c_{2l})(-ic_{2l}c_{2m})(-ic_{2m}c_{2i})\\
&= i \cdot (-i)^5 c_{2i}c_{2j}c_{2j}c_{2k}c_{2k}c_{2l}c_{2l}c_{2m}c_{2m}c_{2i}\\
&= c_{2i} \cdot I \cdot I \cdot I \cdot I \cdot c_{2i} = I, \text{ for } s = 5, V = \{i,j,k,l,m\}.
\end{aligned} \tag{A.3}$$

For clarity, all other cases of 5.12 are omitted, but they follow in a similar way.

For completeness, we also show the derivation of the same commutation relations, but now for the encoded generators 5.14. For this derivation, keep in mind that Pauli operators commute if they work on different qubits:

$$\begin{aligned}
\tilde{B}_j^\dagger &= \left(\prod_{k:(j,k)\in E} Z_{jk}\right)^\dagger = \prod_{k:(j,k)\in E} Z_{jk} = \tilde{B}_j, \\
\tilde{A}_{jk}^\dagger &= \left(\epsilon_{jk} X_{jk} \prod_{p:(j,p)<_j(j,k)} Z_{jp} \prod_{q:(k,q)<_k(k,j)} Z_{kq}\right)^\dagger \\
&= \epsilon_{jk} X_{jk} \prod_{p:(j,p)<_j(j,k)} Z_{jp} \prod_{q:(k,q)<_k(k,j)} Z_{kq} = \tilde{A}_{jk}, \\
\tilde{B}_j^2 &= \left(\prod_{k:(j,k)\in E} Z_{jk}\right)\left(\prod_{k:(j,k)\in E} Z_{jk}\right) = I, \text{ since } Z^2 = I, \\
\tilde{A}_{jk}^2 &= \left(\epsilon_{jk} X_{jk} \prod_{p:(j,p)<_j(j,k)} Z_{jp} \prod_{q:(k,q)<_k(k,j)} Z_{kq}\right)\left(\epsilon_{jk} X_{jk} \prod_{p:(j,p)<_j(j,k)} Z_{jp} \prod_{q:(k,q)<_k(k,j)} Z_{kq}\right) = I,
\end{aligned}$$

since $X^2 = Z^2 = I,$ and X and Z commute since they work on different qubits,

$$\tilde{B}_j\tilde{B}_k = \left(\prod_{a:(j,a)\in E} Z_{ja}\right)\left(\prod_{a:(k,a)\in E} Z_{ka}\right) = \tilde{B}_k\tilde{B}_j,$$

by a similar reasoning,

$$\begin{aligned}
\tilde{A}_{jk} &= \epsilon_{jk} X_{jk} \prod_{p:(j,p)<_j(j,k)} Z_{jp} \prod_{q:(k,q)<_k(k,j)} Z_{kq} \\
&= -\left(\epsilon_{kj} X_{kj} \prod_{p:(k,p)<_k(k,j)} Z_{kp} \prod_{q:(j,q)<_j(j,k)} Z_{jq}\right) = -\tilde{A}_{kj}, \\
\tilde{A}_{ij}\tilde{B}_k &= \left(\epsilon_{ij} X_{ij} \prod_{p:(i,p)<_i(i,j)} Z_{ip} \prod_{q:(j,q)<_j(j,i)} Z_{jq}\right) \prod_{l:(k,l)\in E} Z_{kl} \\
&= \prod_{l:(k,l)\in E} Z_{kl} \left(\epsilon_{ij} X_{ij} \prod_{p:(i,p)<_i(i,j)} Z_{ip} \prod_{q:(j,q)<_j(j,i)} Z_{jq}\right) = \tilde{B}_k\tilde{A}_{ij}, \quad i,j \neq k,
\end{aligned}$$

since if k is not a neighbour of i and j, this follows by independence of the Pauli operators. And if k is a neighbour of i and j, then we get a term of for example $Z_{jk}Z_{jk} = I,$

$$\begin{aligned}
\tilde{A}_{kj}\tilde{B}_k &= \left(\epsilon_{kj} X_{kj} \prod_{p:(k,p)<_k(k,j)} Z_{kp} \prod_{q:(j,q)<_j(j,k)} Z_{jq}\right) \prod_{l:(k,l)\in E} Z_{kl} \\
&= -\prod_{l:(k,l)\in E} Z_{kl} \left(\epsilon_{kj} X_{kj} \prod_{p:(k,p)<_k(k,j)} Z_{kp} \prod_{q:(j,q)<_j(j,k)} Z_{jq}\right) = \tilde{B}_k\tilde{A}_{kj}, \quad i = k, j \neq k,
\end{aligned}$$

since Z anti-commutes with itself when acting on the same qubit,

$$\begin{aligned}
\tilde{A}_{ij}\tilde{A}_{kl} &= \left(\epsilon_{ij} X_{ij} \prod_{p:(i,p)<_i(i,k)} Z_{ip} \prod_{q:(j,q)<_j(j,i)} Z_{jq}\right)\left(\epsilon_{kl} X_{kl} \prod_{p:(k,p)<_k(k,l)} Z_{kp} \prod_{q:(l,q)<_l(l,k)} Z_{lq}\right) \\
&= \left(\epsilon_{kl} X_{kl} \prod_{p:(k,p)<_k(k,l)} Z_{kp} \prod_{q:(l,q)<_l(l,k)} Z_{lq}\right)\left(\epsilon_{ij} X_{ij} \prod_{p:(i,p)<_i(i,k)} Z_{ip} \prod_{q:(j,q)<_j(j,i)} Z_{jq}\right) = \tilde{A}_{kl}\tilde{A}_{ij}.
\end{aligned} \tag{A.4}$$

A.3. Horton's algorithm

In this section, an example of applying Horton's algorithm is given. Before giving the actual example, first Dijkstra's and the Greedy algorithms are discussed, since they are used in Horton's algorithm.

A.3.1. Dijkstra's algorithm

Dijkstra's algorithm for a graph $G = (V, E)$ is an algorithm to find a shortest path in a graph between two points $s, t \in V$ [28]. In this algorithm, $\rho(u)$ is the length of the shortest path from s to u for some $u \in V$, and b is the weight of an edge (throughout this paper we use $b = 1$). The step-by-step formulation of the algorithm is then as follows:

1. Set $W = \{s\}$, where s is the source node.
2. Set $\rho(s) = 0$.
3. Set $\rho(u) = b_{su}$ for all $(s, u) \in E$ such that $u \in V \setminus \{s\}$.
4. For all $u \in V \setminus \{s\}$ such that $(s, u) \notin E$, set $b_{su} = \infty$.
5. While $V \backslash W \neq \emptyset$, do the following:
    (a) Find the node $u \in V \setminus W$ with the smallest $\rho(u)$.
    (b) Add u to the set W.
    (c) For each neighbour $v \in V \setminus W$ such that $(u, v) \in E$, update the shortest known distance: $\rho(v) := \min(\rho(v), \rho(u) + b_{uv})$
6. When $V \backslash W = \emptyset$, return $\rho(t)$, the shortest path distance from s to t.

A.3.2. Example of Horton's algorithm

For the example, we consider the graph given in fig. A.1, where the weights of all edges are set to 1.

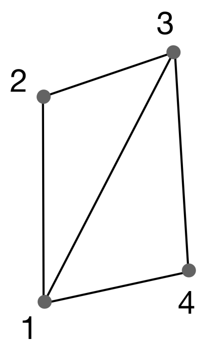
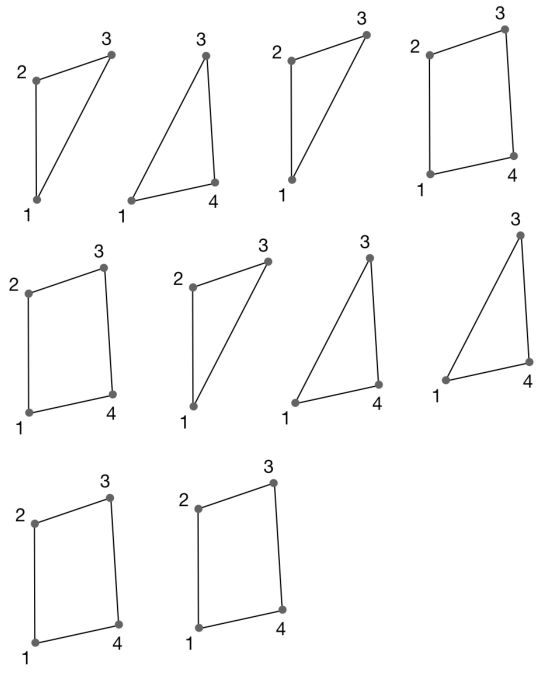


Figure A.1: The graph which will be used as an example of applying Horton's algorithm.

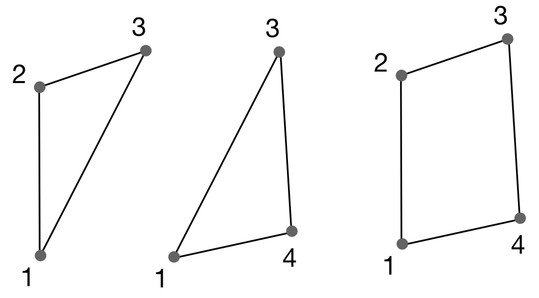
Since all weights are equal to one, applying Dijkstra's algorithm to this graph to find a shortest path from node 2 to 4 for example is easy, we then get the path 2-3-4 or 2-1-4. After applying the first step of Horton's algorithm (1), we get the set of cycles displayed in fig. A.2a. We see that some cycles are displayed more than once, so after removing all duplicates and ordering the cycles by weight, we get the set of cycles given in fig. A.2b.

We denote these cycles respectively by C_1, C_2 and C_3. We now apply step 3 of Horton's algorithm to these cycles. We obtain the following cycle-edge incidence matrix:

	$(1,2)$	$(1,3)$	$(1,4)$	$(2,3)$	$(3,4)$
C_1	1	1	0	1	0
C_2	0	1	1	0	1
C_3	1	0	1	1	1



(a) All cycles obtained after applying the first three steps of Horton's algorithm, using the graph from fig. A.1.



(b) All distinct cycles, ordered by weight, obtained from the first three steps of Horton's algorithm.

Figure A.2: Cycles obtained after applying the first three steps of Horton's algorithm.

We see that the weights of the rows (and thus of the cycles) are respectively equal to 3, 3 and 4. So we will process the cycles in this order. Since we start with an empty basis, we add the cycle C_1 to the basis. After this, we move on to cycle C_2. Using Gaussian elimination, we see that this cycle is linearly independent from C_1 in the cycle-edge incidence matrix, so we add it to the basis. Doing the same for cycle C_3, we see that this cycle is not linearly independent from the cycles which are already in the basis, so we do not add this cycle to the basis (note that we are working modulo 2 since we consider a 0-1 matrix here).

So we obtain a final cycle basis as shown in fig. A.3.

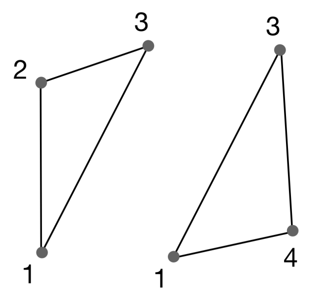


Figure A.3: The cycle basis obtained after applying all steps from Horton's algorithm.

A.4. Example of the algorithm of Freedman-Hastings

This section provides a visual example of the algorithm described in the proof of the Freedman-Hastings Decongestion Lemma in 5.4.1. The example is displayed in fig. A.4.

In each step, the edge, vertex or cycle that we consider (depending on in which of the three cases we are) is marked red, and the result of removing that vertex/edge and/or replacing it with another edge is given after the right arrow. When we are in step three of the algorithm, the edge that we will remove from the cycle is pointed out also by an arrow.

The resulting cycle basis C is shown in the bottom right part.

In the last step shown in fig. A.4c, we use the last part of step 2B of the algorithm to replace the left edge between nodes 3 and 9 with the original edges $(2,3)$ and $(2,9)$ that we removed in step 4 of the example, before adding the cycle to the cycle basis.

A.5. Derivations for representing the Fermi-Hubbard model on a graph

In this section we first present the derivations of the statements made in sec. 5.5.1 about representing the Fermi-Hubbard model on a graph.

The first thing that we need to prove is that we can write:

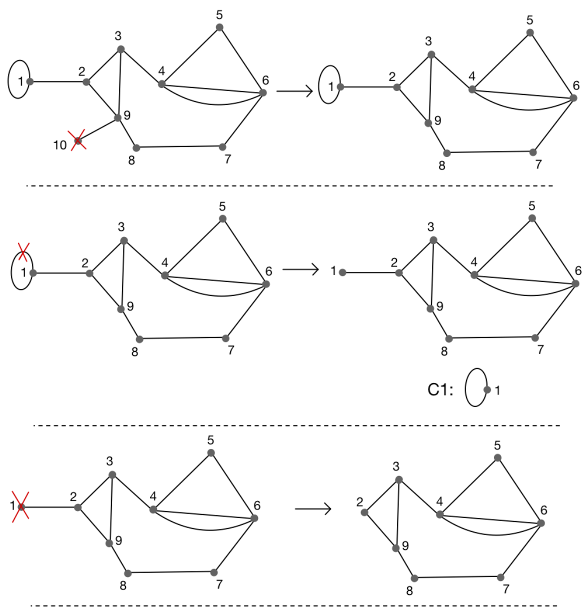
$$n_j = \frac{1}{2}(I - ic_{2j-1}c_{2j}). \tag{A.5}$$

To do this, remember the definition of the Majorana operators given in eq. 2.12 and the anti-commutation relations of the fermionic creation and annihilation operators given in eq. 2.8. We now present the derivation:

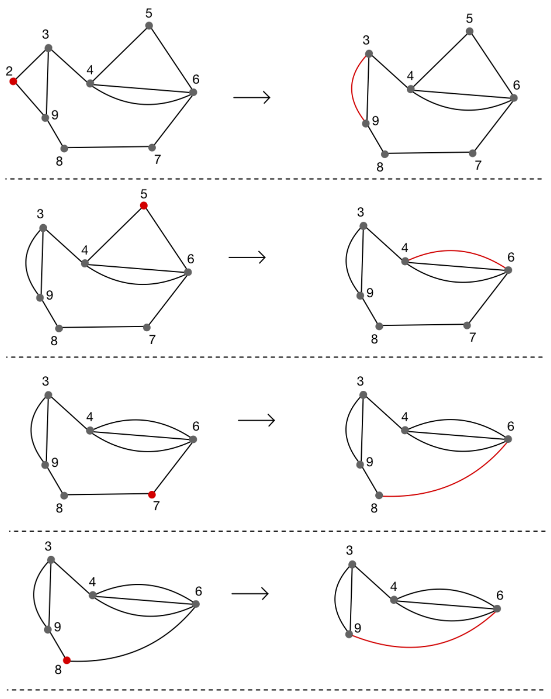
$$\begin{aligned}
\frac{1}{2}\left(I - ic_{2j-1}c_{2j}\right) &= \frac{1}{2}\left(I - i(a_j + a_j^\dagger)\frac{a_j - a_j^\dagger}{i}\right) \\
&= \frac{1}{2}\left(I - (a_j + a_j^\dagger)(a_j - a_j^\dagger)\right) \\
&= \frac{1}{2}\left(I - a_j^2 + a_{j,\uparrow}a_j^\dagger - a_j^\dagger a_j + (a_j^\dagger)^2\right) \\
&= \frac{1}{2}\left(I + a_j a_j^\dagger + a_j a_j^\dagger - I\right) \\
&= \frac{1}{2}\left(2a_j a_j^\dagger\right) \\
&= a_j a_j^\dagger \\
&= n_j.
\end{aligned} \tag{A.6}$$

The second thing that we must show is that we can write:

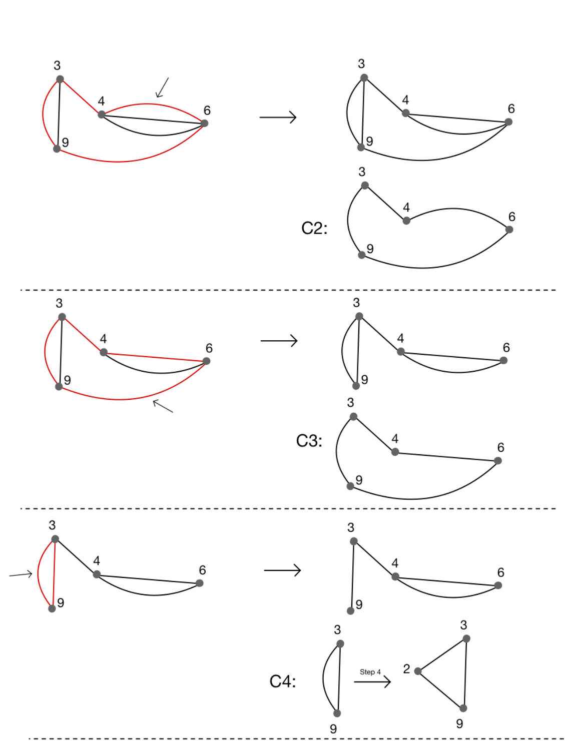
$$\begin{aligned}
a_{j,\sigma}^\dagger a_{k,\sigma} + a_{k,\sigma}^\dagger a_{j,\sigma} &= -\frac{i}{2}(c_{2k,\sigma}c_{2j-1,\sigma} + c_{2j,\sigma}c_{2k-1,\sigma}) \\
&= \frac{i}{2}A_{jk,\sigma}(B_{j,\sigma} - B_{k,\sigma}).
\end{aligned} \tag{A.7}$$



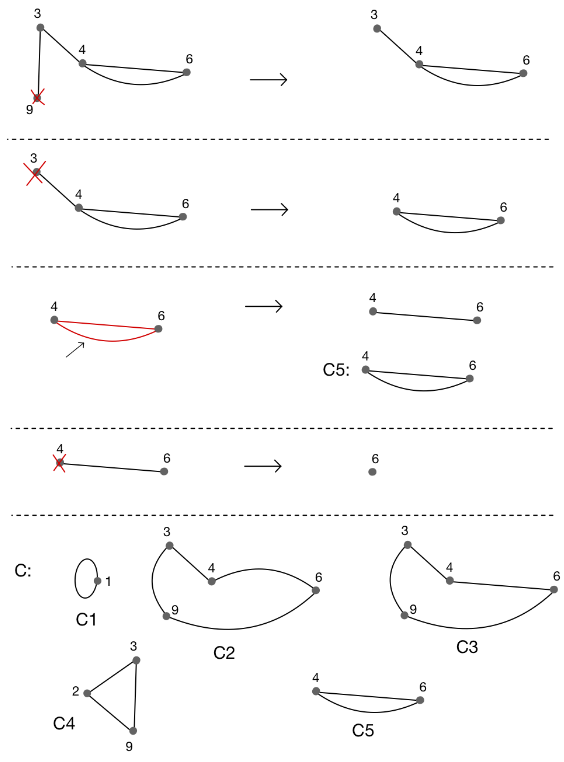
(a) Steps 1-3 of the algorithm.



(b) Steps 4-7 of the algorithm.



(c) Steps 8-10 of the algorithm.



(d) Steps 11-14 of the algorithm.

Figure A.4: Example of the algorithm described in the proof of the Freedman-Hastings Decongestion Lemma given in 5.4.1.

We will derive both equalities separately, starting with the second equality:

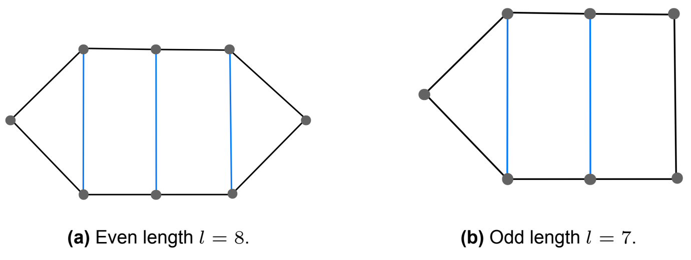
$$\begin{aligned}\frac{i}{2}A_{jk,\sigma}(B_{j,\sigma}-B_{k,\sigma}) &= \frac{1}{2}c_{2j}c_{2k}\left(-ic_{2j-1}c_{2j}+ic_{2k-1}c_{2k}\right)\\ &= \frac{i}{2}c_{2j}c_{2k}\left(-c_{2j-1}c_{2j}+c_{2k-1}c_{2k}\right)\\ &= \frac{i}{2}\left(-c_{2j}c_{2k}c_{2j-1}c_{2j}+c_{2j}c_{2k}c_{2k-1}c_{2k}\right)\\ &= \frac{i}{2}\left(c_{2j}c_{2k}c_{2j}c_{2j-1}-c_{2j}c_{2k}c_{2k}c_{2k-1}\right)\\ &= \frac{i}{2}\left(-c_{2j}c_{2j}c_{2k}c_{2j-1}-c_{2j}c_{2k}c_{2k}c_{2k-1}\right)\\ &= -\frac{i}{2}\left(c_{2k}c_{2j-1}+c_{2j}c_{2k-1}\right).\end{aligned} \tag{A.8}$$

Now we show that this is equal to $a_j^\dagger a_k + a_k^\dagger a_j$:

$$\begin{aligned}-\frac{i}{2}\left(c_{2k}c_{2j-1}+c_{2j}c_{2k-1}\right) &= -\frac{1}{2}\left[\left(a_k-a_k^\dagger\right)\left(a_j+a_j^\dagger\right)+\left(a_j-a_j^\dagger\right)\left(a_k+a_k^\dagger\right)\right]\\ &= -\frac{1}{2}\left[a_ka_j+a_ka_j^\dagger-a_k^\dagger a_j-a_k^\dagger a_j^\dagger+a_ja_k+a_ja_k^\dagger-a_j^\dagger a_k-a_j^\dagger a_k^\dagger\right]\\ &= -\frac{1}{2}\left[-a_ja_k-a_j^\dagger a_k+a_ja_k^\dagger+a_j^\dagger a_k^\dagger+a_ja_k+a_ja_k^\dagger-a_j^\dagger a_k-a_j^\dagger a_k^\dagger\right]\\ &= -\frac{1}{2}\left[-2a_j^\dagger a_k+2a_ja_k^\dagger\right]\\ &= a_j^\dagger a_k + a_k^\dagger a_j.\end{aligned} \tag{A.9}$$

A.6. Proof of the properties of the different sewing methods

In this section, we provide the proof of one of the properties shown in table 7.2 displaying the properties of the different sewing methods. The other properties shown in this table can be proven in a similar way. The properties regarding the locality of the Hamiltonian terms and stabilizer constraints are straightforward to see when looking at the image illustrating the sewing method, so we will prove the relation between the number of qubits and the cycle length l. We will prove these results for the straight sewing method.
We illustrate the straight sewing method once more in fig. A.5, now for both a cycle with even and odd length l.



(a) Even length $l = 8$. **(b)** Odd length $l = 7$.

Figure A.5: Straight sewing for a cycle with even length l (left) and odd length l (right). The blue edges are the edges added in the sewing process.

Note that the number of qubits added in the sewing process is equal to the number of edges added. Now, if the length l is even, then we determine the number of edges that need to be added by first removing the most left and most right vertices from the cycle. Then there are $l-2$ vertices remaining. We add an edge between every pair of vertices left, so the number of edges added is equal to $\frac{l-2}{2}$.

If l is odd, then we first remove the most left vertex and the two most right vertices. There are then $l-3$ vertices remaining, and we again add an edge between every pair of vertices left, so we add $\frac{l-3}{2}$ edges in this case.

B

Relevant plots

In this chapter we show all other relevant plots that are not previously showed in this thesis. In sec. B.1 we show the locality and sparsity of the cycle basis obtained by using among others Horton's algorithm.

In the rest of this chapter, we show the relevant plots for the different methods considered in chapter 8 that are not already shown there.

B.1. Horton's algorithm applied to the Erdos-Renyi graph

In fig. B.1 we show the locality and sparsity for the cycle basis obtained by among others Horton's algorithm, applied to the Erdos-Renyi graph described in sec. 3.2.

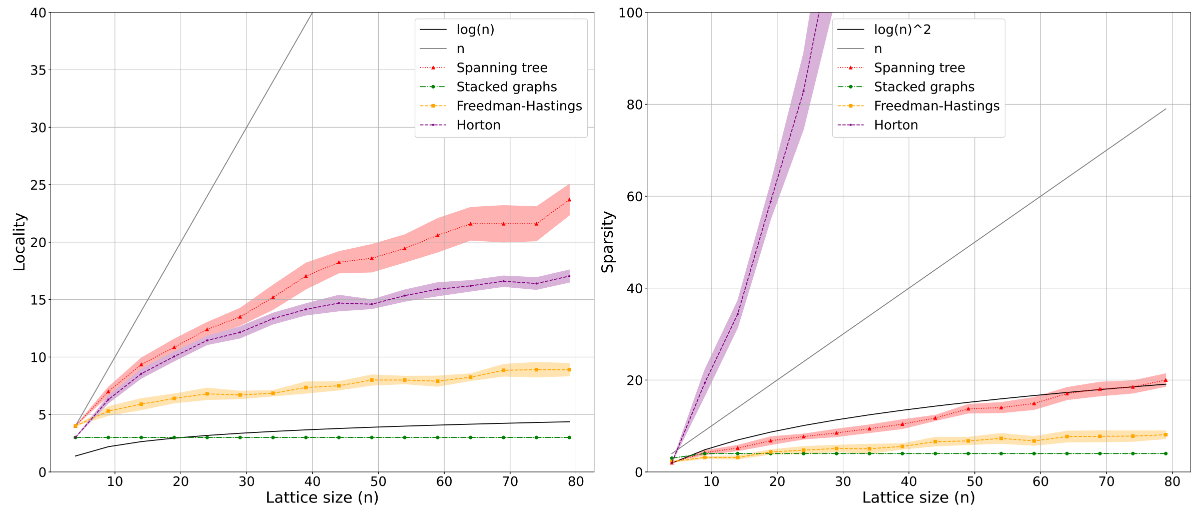
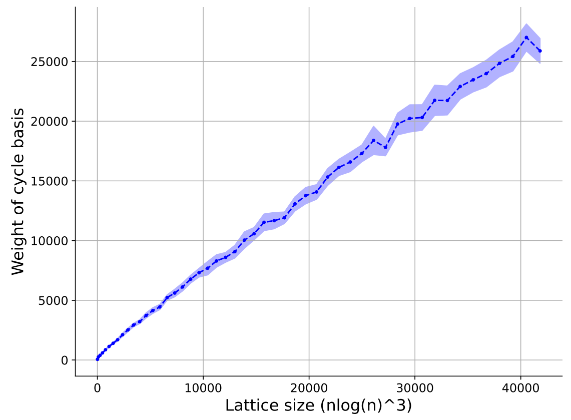


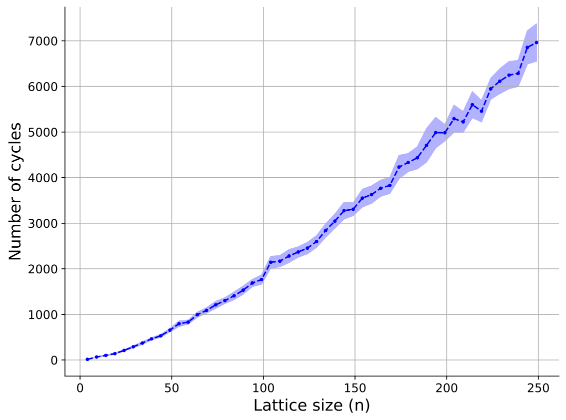
Figure B.1: Locality (left) and sparsity (right) of the cycle basis obtained by among others Horton's algorithm, applied to the Erdos-Renyi graph. From the locality plot, we can conclude that it seems to be true that there exists at least one cycle with length at least $\mathcal{O}(\log(n))$.

B.2. Straight sewing

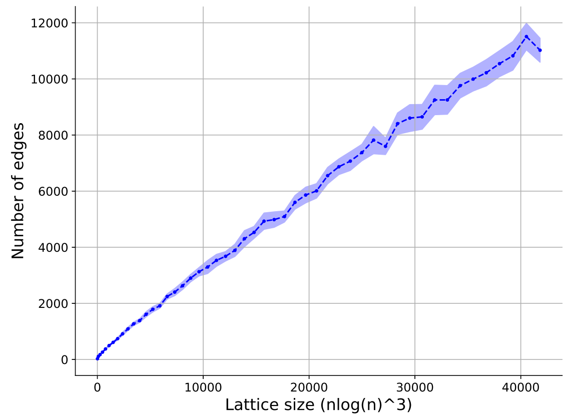
B.2.1. No vertex overlap



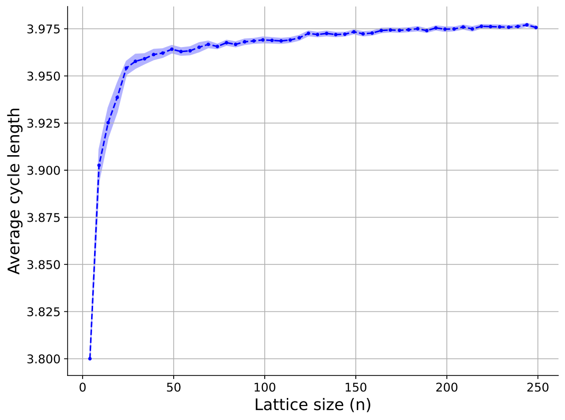
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



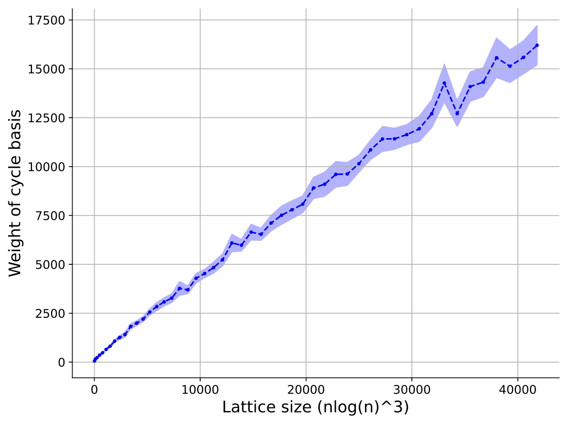
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



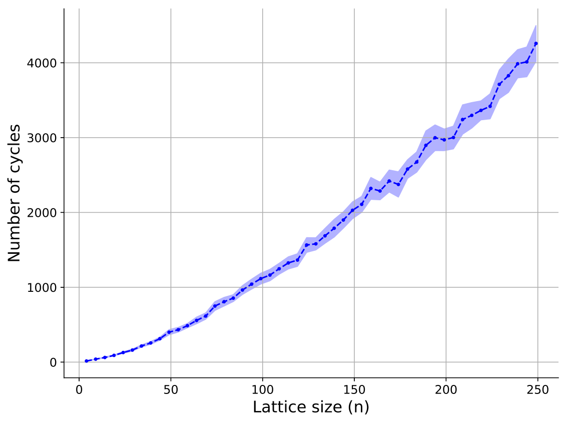
(d) The average length of all cycles in the obtained cycle basis.

Figure B.2: Properties of the fermionic encoding after combining the straight sewing method with allowing no overlap between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

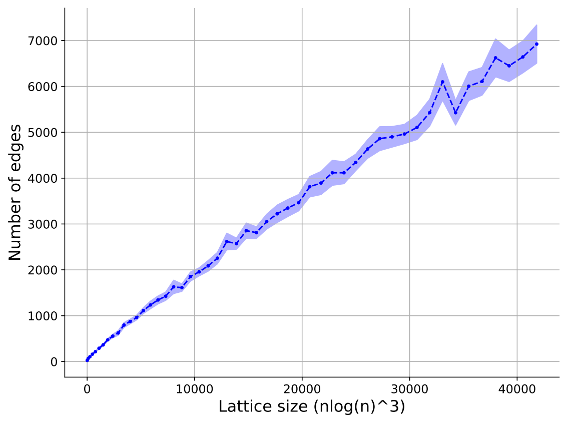
B.2.2. Overlap on at most 1 vertex



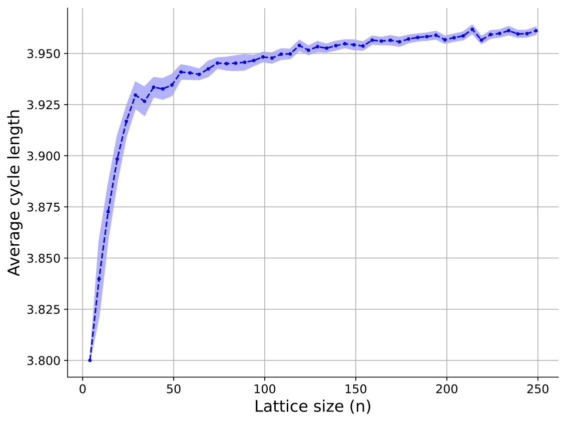
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



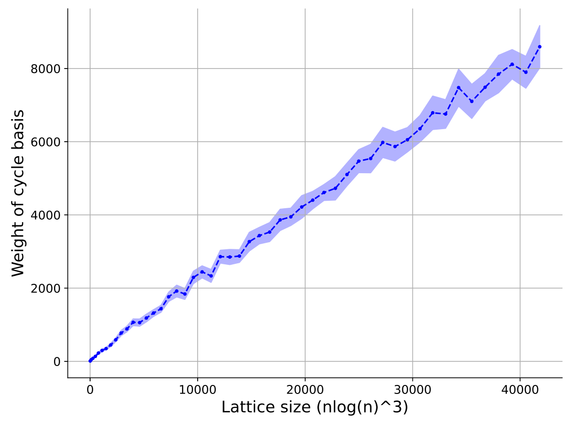
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



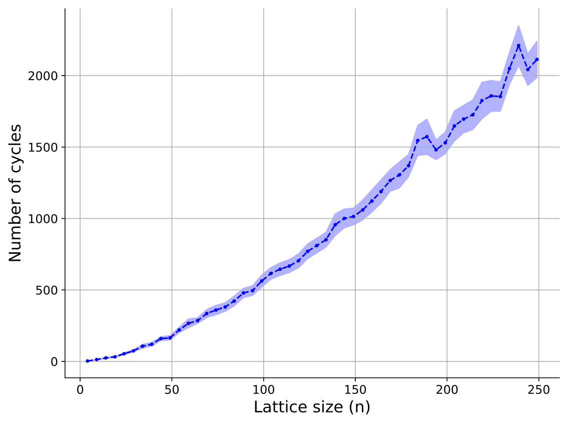
(d) The average length of all cycles in the obtained cycle basis.

Figure B.3: Properties of the fermionic encoding after combining the straight sewing method with allowing an overlap on at most 1 vertex between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

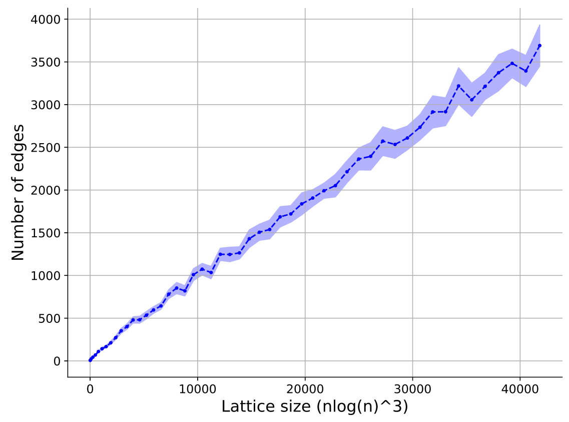
B.2.3. Overlap on at most 2 vertices



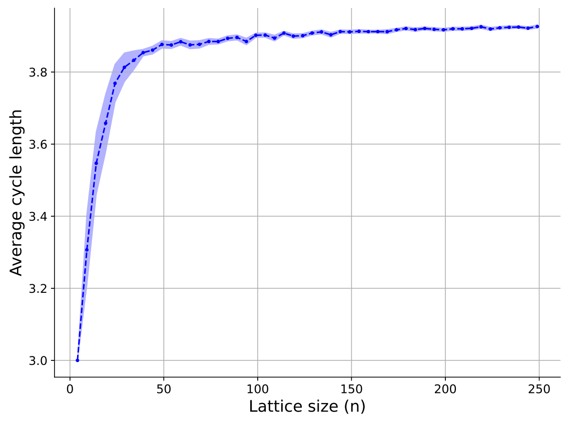
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



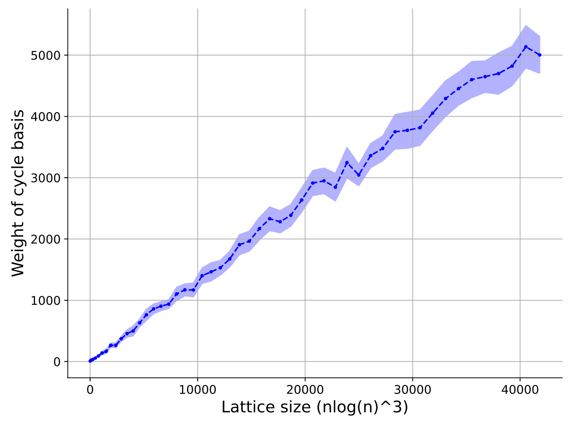
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



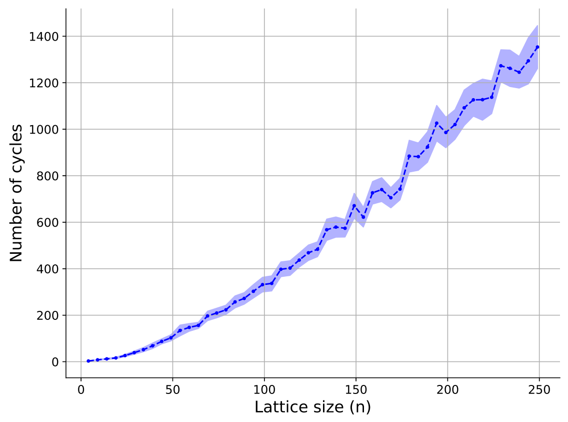
(d) The average length of all cycles in the obtained cycle basis.

Figure B.4: Properties of the fermionic encoding after combining the straight sewing method with allowing an overlap on at most 2 vertices between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

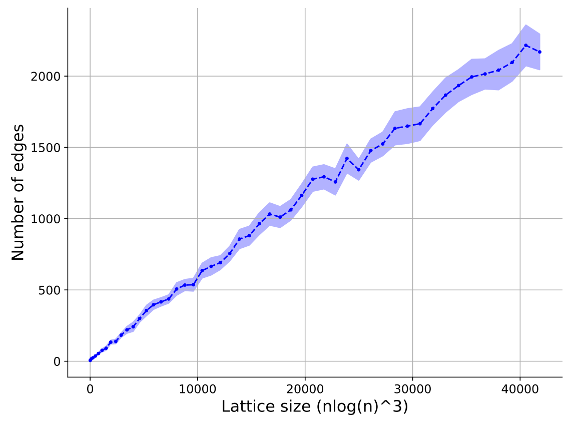
B.2.4. Overlap on at most 3 vertices



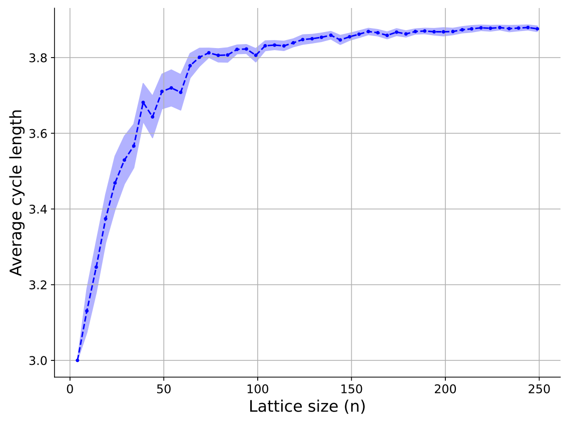
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



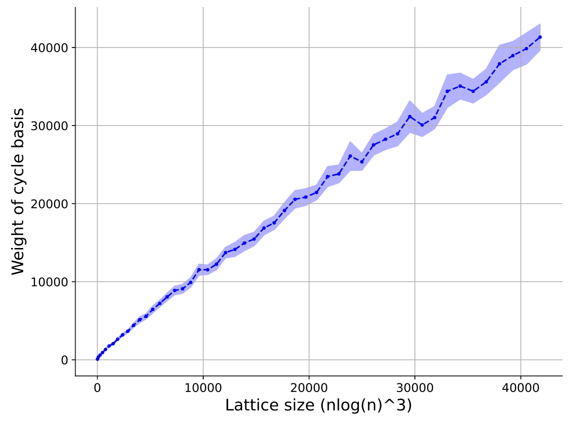
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



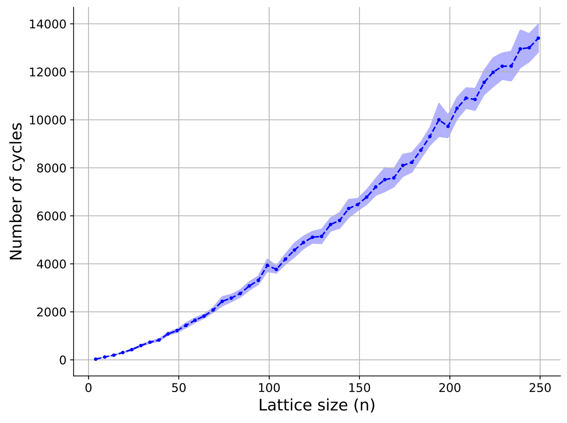
(d) The average length of all cycles in the obtained cycle basis.

Figure B.5: Properties of the fermionic encoding after combining the straight sewing method with allowing an overlap on at most 3 vertices between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

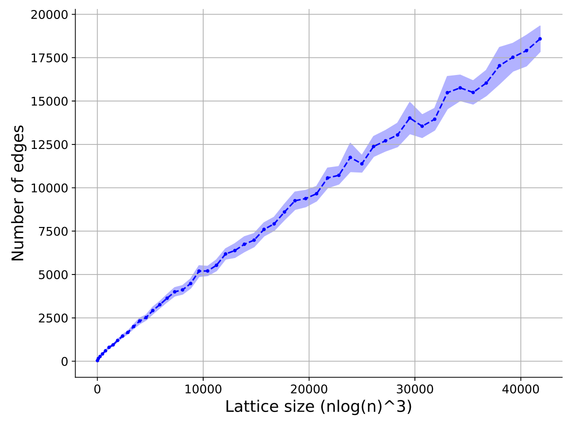
B.3. Triangular sewing without overlap



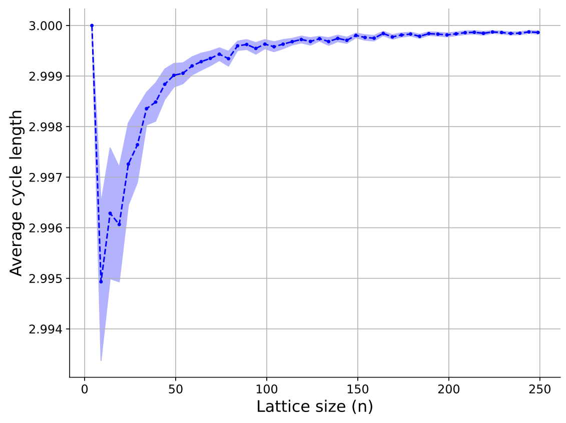
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.

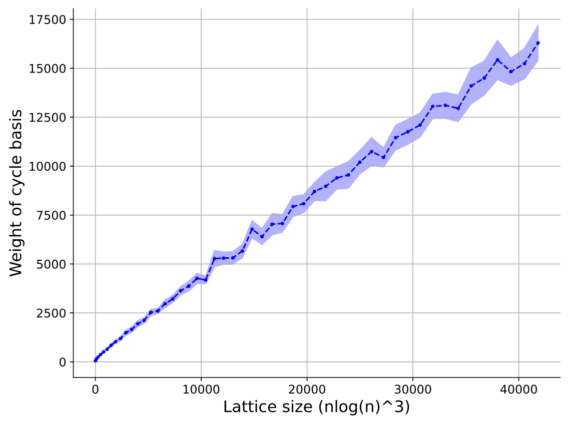


(d) The average length of all cycles in the obtained cycle basis.

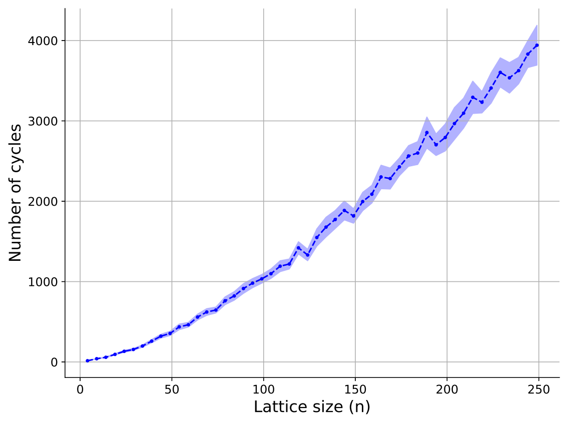
Figure B.6: Properties of the fermionic encoding after combining the triangular sewing method with allowing no overlap between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

B.4. Straight-6 sewing

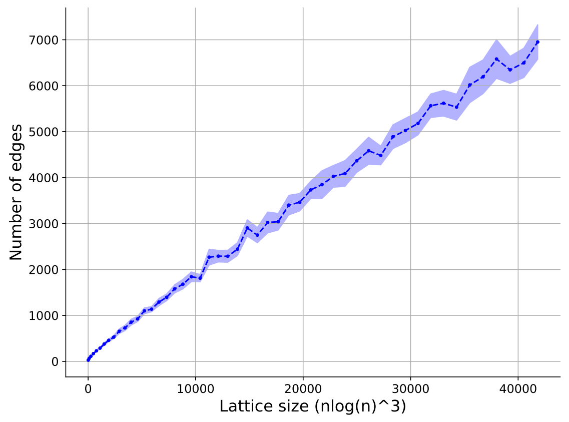
B.4.1. Overlap on at most 1 vertex



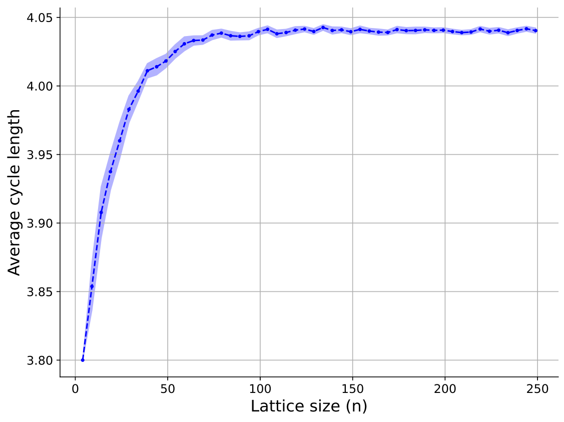
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



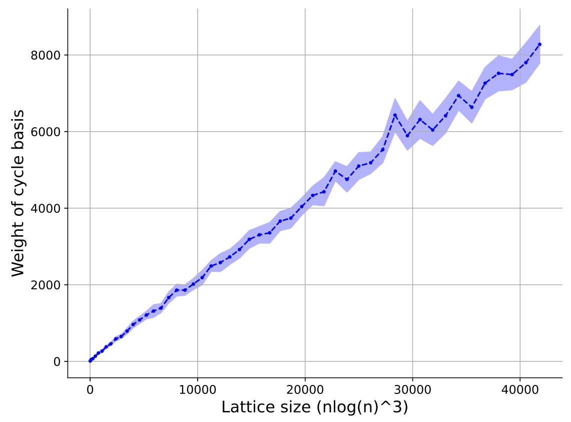
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



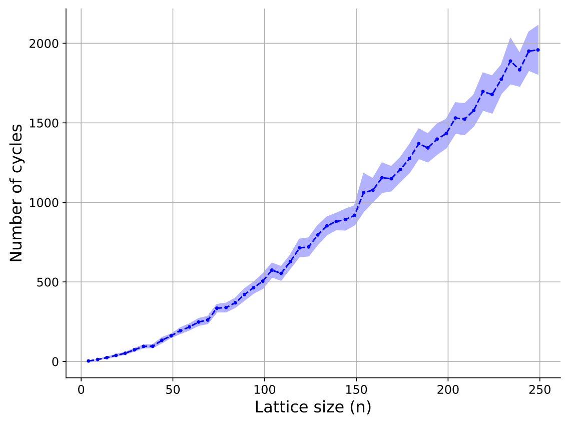
(d) The average length of all cycles in the obtained cycle basis.

Figure B.7: Properties of the fermionic encoding after combining the straight-6 sewing method with allowing an overlap on at most 1 vertex between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

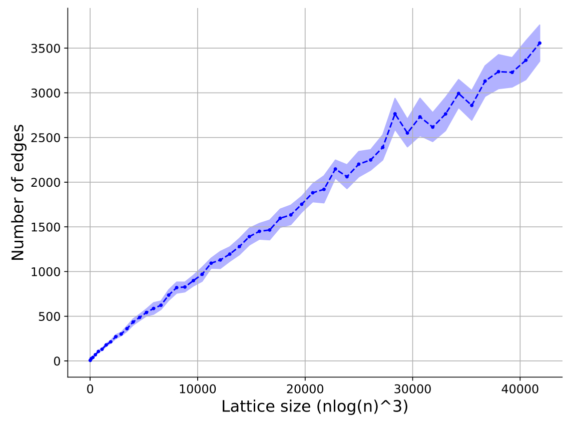
B.4.2. Overlap on at most 2 vertices



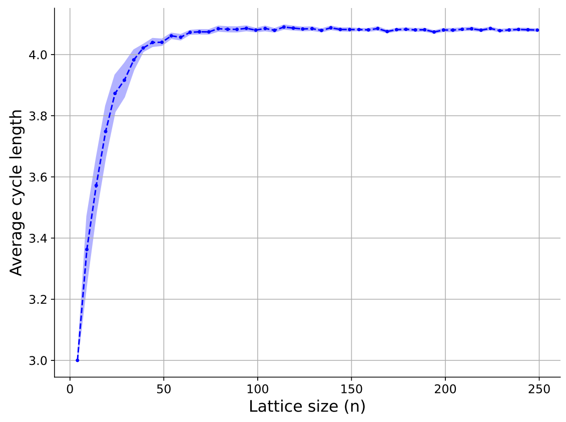
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



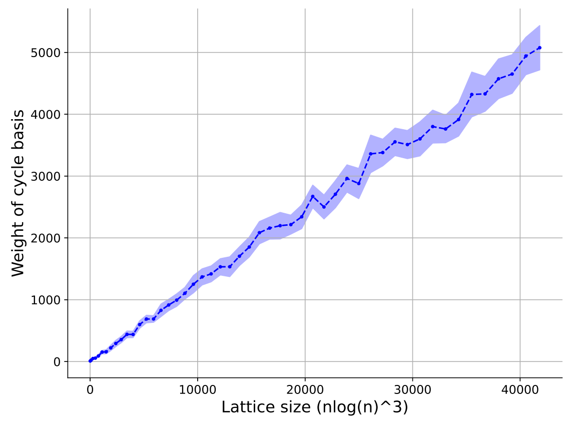
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



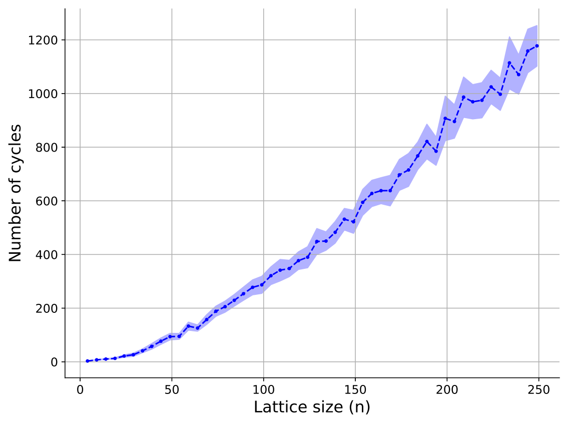
(d) The average length of all cycles in the obtained cycle basis.

Figure B.8: Properties of the fermionic encoding after combining the straight-6 sewing method with allowing an overlap on at most 2 vertices between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

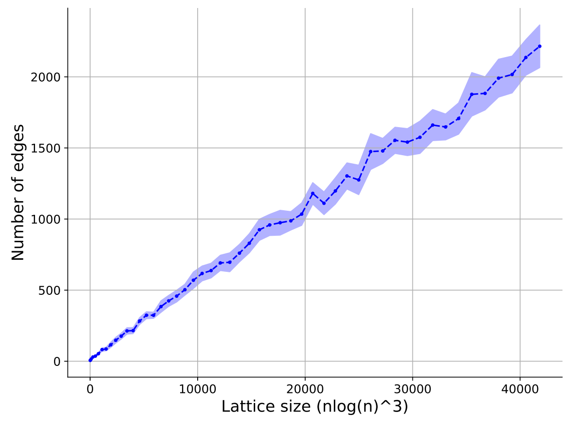
B.4.3. Overlap on at most 3 vertices



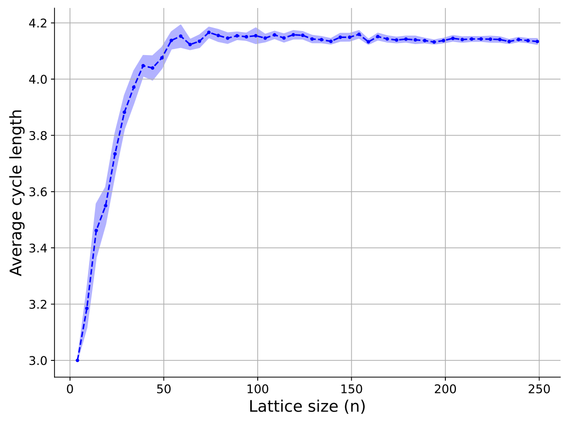
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



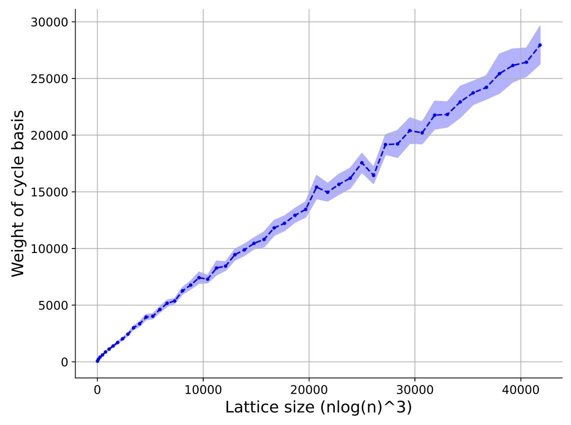
(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



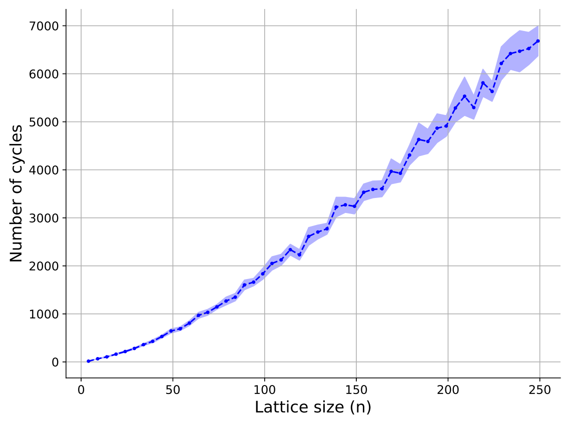
(d) The average length of all cycles in the obtained cycle basis.

Figure B.9: Properties of the fermionic encoding after combining the straight-6 sewing method with allowing an overlap on at most 3 vertices between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

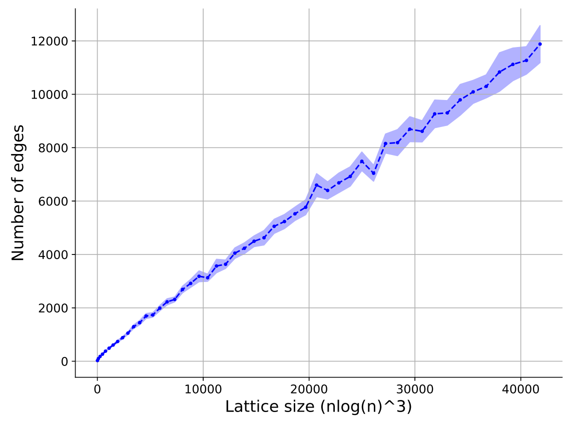
B.5. Skew sewing without overlap



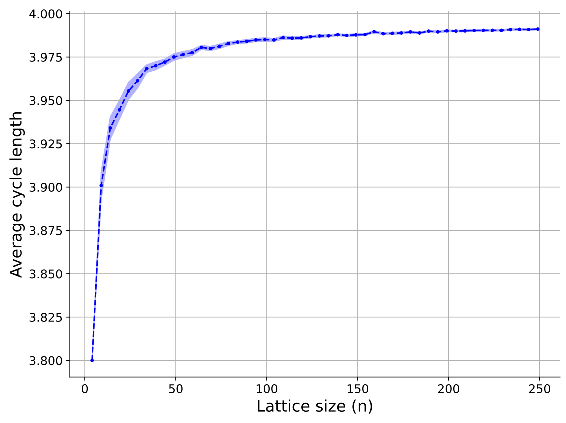
(a) Total weight of the obtained cycle basis. The total weight is equal to the sum over edges of the weight of that edge times the expected number of times the algorithm constructs a cycle C containing a descendant of the edge.



(b) Size of the obtained cycle basis. The size is equal to the number of cycles in the cycle basis.



(c) Number of edges in the obtained graph G'. In the final fermionic encoding, the number of edges is equal to the number of qubits required for the encoding, since we place a qubit at every edge.



(d) The average length of all cycles in the obtained cycle basis.

Figure B.10: Properties of the fermionic encoding after combining the skew sewing method with allowing no overlap between cycles that are being sewed in the same layer in the Vertex Coloring algorithm.

C
GitHub

All the relevant Python code used for the numerical implementation of the algorithms discussed in this thesis can be found at the GitHub page:
https://github.com/KoenEggen/Bachelor_Thesis_Koen_Eggen.git