

Classical Simulation
of
Quantum Computations

Klassische Simulation
der
Quantenrechnung

Masterarbeit eingereicht von
Henrik Dreyer
als Auflage zum Promotionsstudium

Abstract

Let C_n be a uniform family of quantum circuits comprising parity preserving 2-qubit gates that act on nearest neighbour qubit lines with a final measurement in the computational basis on a single line. Then we can calculate the resulting probability distribution with a classical circuit that consists of a number of logic gates that scales polynomially with the input size n . We will examine how this result arises both from Valiant's Matchgate formalism and non-interacting fermion systems as proposed by Terhal and DiVincenzo. A Python simulation for illustration and computation of these probabilities in the Matchgate model is proposed. After investigating how we can reestablish universal quantum computing with minimal quantum supplements we will try to extend the formalism to qutrit (3-level) systems which is the first step to a generalisation to qudits.

Contents

1	Introduction	3
2	Statement of the Problem	4
3	Matchgates	7
3.1	The Character is computable in polynomial time	8
3.2	Certain gates can be expressed as Character matrices	13
3.3	Circuits are expressible as Character matrices	18
3.4	A PYTHON routine to simulate Matchcircuits	21
4	Non-interacting fermions	22
4.1	Time evolution in an exponentially scaling space	23
4.2	Certain systems evolve in a linearly scaling subspace	25
4.3	The time evolution operator as a quantum gate	26
4.4	Particle number mod 2-conserving Hamiltonians span $\mathfrak{su}(2) \oplus \mathfrak{su}(2)$	28
4.5	Majorana fermions & particle number mod 2 conservation	31
5	Establishing ties	32
6	Minimal quantum aids & Universal computation	33
7	Can the formalism be generalised to qudits?	35
7.1	Pauli-operators on qutrits	36
7.2	One-qutrit gates	36
7.3	Further questions and outlook	38
A	Modifiers	41
B	Calculation of the interaction commutator	42
C	Planarity	42
D	Source File for the Matchgate Simulator	43
E	Declaration	55

1 Introduction

Interest in the experimental implementation of quantum computation has risen significantly over the past years. This trend is because it is widely believed that these quantum computational devices will be more powerful than their classical counterpart. While there are various aspects of quantum information processing that are lacking in the classical analogue, it is not clear which defining characteristic is responsible for the additional power that those devices are conjectured to possess. The significance of studying this question is twofold: It is of technological interest to figure out which parts of a quantum computer could actually be performed by a classical machine; it is conceivable that the quantum computer will consist of a small quantum "portion" that is designed to solve a specific task, whereas the major component of the machine will comprise classical pre- and post-processing. The larger the latter part, the easier it is to implement the machine. Secondly, this topic is of scientific interest. What can we learn about the limit of what is computable in the physical world? Much like in Hamiltonian complexity theory, we can ask what these considerations tell us about the physical system and about quantum mechanics itself. While this thesis will not address these questions in their entirety, it will shed light upon several lines of thought in this area of research.

The method that we are going to employ to investigate these questions is attempting to simulate quantum computation classically. While this approach may appear to be against progress (after all, if we are successful, it would undermine all advances toward a technological implementation of a quantum computer), it is a fruitful one. Even if we do not succeed in simulating a given quantum computation, we still glean more insight to the nature of the problem. The most interesting findings in this area of research have actually come from *failures* rather than successes. We will expand on this notion over the course of the paper.

Section 2 will define what quantum computation and classical simulation are. It will allow us to state the problem mathematically and we will see that certain restrictions have to be made in the quantum computation to successfully do such a simulation. Section 3 follows with a graph-theoretic approach in terms of "matchgates" that relies on the efficient computation of the *Pfaffian*, a quantity that a priori consists of an exponential number of terms. Computing this quantity is connected to the task of counting perfect matchings in planar graphs. This formalism will prove the main theorem of this thesis. As a conclusion to the section, a Python program that illustrates the simulation of suitable quantum circuits through matchcircuits will be presented. We will then move on to section 4 where a model of non-interacting fermions will be presented. Even though seemingly unrelated, it will prove the simulatability of the same kind of circuits as in the matchgate formalism. The deciding mathematical observation in this part of the paper will be that time evolution can be described in terms of the generators of a group rather than the group itself. Here, the exponential gap arises from the fact that a group with n elements is generated by only $\log(n)$ elements. For this compact description, the Hamiltonian of the system will be required to be *quadratic* which corresponds physically to a system of non-interacting fermions. We will briefly review the connection of Matchgates and fermions in section 5.

As mentioned previously, it is particularly interesting to see which quantum aids would be needed in order to arrive back at universal quantum computing. In section 6, we will find that the separation is quite subtle and is related to the *determinant condition* that arises naturally from both previously reviewed models. In fact adding any parity preserving nearest neighbour 2-qubit that does not fulfill this condition reestablishes universal quantum computing, the most intuitive of these gates being the gate that swaps two qubit lines.

Section 7 concludes with some ideas regarding the generalization of the developed formalism to qudits, the d -level generalization of a qubit. Even though the technique used in section 4 does not generalize in a straightforward manner, we will see why a generalization should be theoretically possible.

2 Statement of the Problem

In this section we will define *quantum computation*, *classical simulation*, *polynomial time* and explain why it is generally hard to find such a polynomial time simulation for a given quantum computation.

Subsequently, we will constrain ourselves to *decision problems* i.e. computational tasks with single bit output. This does not imply any loss of generality however. For example, the problem of finding factors of an n -bit number N can be reduced to the decision problem whether there is a factor of N smaller than a given k . More general arguments can be found in the literature. In particular, we can compute the output of a function $f : \{0, 1\}^n \rightarrow \mathbb{R}$ to M -digit precision by *deciding* whether the i -th bit of the outcome is one.

We first want to give a formal definition of a *classical computation*:

Definition 2.1. ([5]) A *Classical Boolean circuit* of input size n and output size m is a directed acyclic graph with $n + w$ vertices of in-degree 0 (input and workspace), m vertices of out-degree 0 (output), and all other vertices have out-degree 1 and in-degree 1 or 2. Each vertex with non-0 in-degree is labelled by a logic gate (AND, OR or NOT), where in-degree 1(2) vertices are labelled by gates on 1-bit (2-bits). We denote the output of the circuit given an input x by $C(x)$. If the number of output vertices is one, the circuit implements a Boolean function $f_n(x)$. Since this set of gates is universal, there is a circuit C_n for every Boolean function $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$. The circuit is required to be independent of the input. The number of gates in a circuit is called the *runtime* $T(C_n)$ of the circuit. A *circuit family* $(C_1, C_2, \dots, C_n, \dots)$ is a set of exactly one circuit for each n .

Definition 2.2. Let $B^* = \cup_{n=1}^{\infty} \{0, 1\}^n$ be the set of all n -bit strings for all n . A subset of B^* is called a *language*.

Note that there is also the notion of *randomised classical computation*: A uniformly randomised bit string $r_1 \dots r_k$ is appended to the input. The output $C_n(x)$ will then be a sample from the probability distribution of all possible outputs. We will then not refer to the *output* of the circuit, but rather to the *probability of outputting y given input x* $\text{Prob}[C(x) = y]$.

Definition 2.3. The complexity class **P** is the set of languages such that for all $L \in \mathbf{P}$ and all $n \in \mathbb{N}$ the following statements hold:

1. There exists a circuit family $C = (C_1, C_2, \dots, C_n, \dots)$ such that

$$C_n(x_n) = \begin{cases} 1 & \text{if } x_n \in \{0, 1\}^n \cap L \\ 0 & \text{if } x_n \in \{0, 1\}^n \setminus L \end{cases} \quad (2.1)$$

with certainty.

2. There exist two real constant $c, k \in \mathbb{R}$ and a natural number $N \in \mathbb{N}$ for which it holds that

$$\forall n > N : T(C_n) < cn^k$$

We will from now on abbreviate the latter condition by saying $T(C_n) = \mathcal{O}(n^k)$.

Definition 2.4. The complexity class **BPP** is defined exactly as **P** above (2.3), replacing (2.1) by

$$\text{Prob}[C_n(x_n) = 1] \begin{cases} \geq \frac{2}{3} & \text{if } x_n \in \{0, 1\}^n \cap L \\ \leq \frac{1}{3} & \text{if } x_n \in \{0, 1\}^n \setminus L \end{cases} \quad (2.2)$$

We note, that the probabilities in the definition are arbitrary to a certain extent. In particular, these probabilities can depend on the input size, as long as the acceptance probabilities of YES- and NO-instances are separated by at least the inverse of a polynomial function of the input size (which is explained in [7]). On the other hand, by running the circuit polynomially often, one can achieve acceptance probabilities exponentially close to 1 and 0, respectively, as the tail of the probability distribution (i.e. the events where one guesses wrong) is suppressed by the Chernoff bound.

We are now ready to define a *quantum* computation.

Definition 2.5. ([5]) A quantum circuit from n qubits to m qubits is a directed acyclic graph, with $n + w$ vertices of in-degree 0 (input and workspace), m vertices of out-degree 0 (output), and all other vertices have in-degree 1 or 2 and out-degree equal to in-degree. Each vertex with non-0 in-degree is labelled by a quantum gate, where degree 1 (2) vertices are labelled by 1-qubit (2-qubit) quantum gates. The *input* of the circuit is a computational basis state $|\psi_0\rangle = |b_1\rangle \otimes |b_2\rangle \otimes \dots \otimes |b_n\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle$ where $b_i \in \{0, 1\}$ and the additional zero-states represent the *workspace* of the circuit. The *output* is the result of a quantum measurement of the specified qubit m lines in the computational basis. Hence, application of the circuit to a given input $C(|b_1\rangle \otimes |b_2\rangle \otimes \dots \otimes |b_n\rangle)$ is a sample of the probability distribution of all possible outputs. Note that for a measurement set consisting of a single qubit line, the output will attain the value 1 with probability $p_1(C)$ and 0 with probability $p_0(C) = 1 - p_1(C)$. The *runtime* of a quantum circuit is the number of gates in the circuit C and is denoted by $T(C)$. A *family of quantum circuits* $(C_1, C_2, \dots, C_n, \dots)$ is a set of exactly one circuit for each n and is also called a *quantum algorithm*.

In the following, we will omit the tensor product in product states, i.e. we will write $|b_1\rangle \otimes |b_2\rangle = |b_1\rangle |b_2\rangle$.

Definition 2.6. The complexity class **BQP** is defined exactly as **BPP** above (2.4), replacing “circuit family” by “family of quantum circuits”.

We consider languages in **BQP** to constitute the tasks that we can realistically perform on a quantum computer, whereas those in **BPP** (of which **P** is a subset) are feasible on a classical computer. The reason is that typical everyday computational tasks involve input sizes of several hundred bits for which runtimes that grow faster than every polynomial (i.e. exponential) often exceed reasonable bounds (e.g. several hundreds of years). Therefore, languages which are not in **BPP** are *hard* for a classical computer, and those outside **BQP** are hard for a quantum computer. For the remainder of this essay, we will only consider problems in **BQP**. It is interesting to investigate which are the features of problems in **BQP** \setminus **BPP**. A possible strategy is *simulation*:

Definition 2.7. Let $C = (C_1, C_2, \dots, C_n, \dots)$ be a quantum algorithm with single qubit output and corresponding outcome probabilities $p_1(C_i)$. Define $(p_1(C_i))_k \in B^*$ to be the binary representation of the decimal places of $p_1(C_i)$, truncated at the k -th decimal point. Let L_i be the language of all these different accuracies i.e. $L_i = \cup_{k=1}^{\infty} (p_1(C_i))_k$. Then the quantum algorithm is *simulable in the strong sense* if $L_i \in \mathbf{P}$ for all $i \in \mathbb{N}$.

There is also the notion of *weak simulation* which requires the existence of a randomised classical computation whose output is drawn from the same probability distribution as the quantum circuit itself. Note that strong simulation requires the ability to *compute the outcome probabilities* to a high level of precision whereas the quantum circuit itself only outputs a *sample* of this probability distribution and in this sense, the quantum circuit *only* weakly simulates itself. Thus, it seems more natural to attempt a weak simulation of families of quantum circuits. However, the results in this essay will solely be concerned with actually calculating the measurement probabilities to high accuracy and hence *strong* simulation. It should be further noted that strong simulatability implies weak simulatability, while the converse is generally not true: according to the law of large numbers, a high number of samples will eventually approximate the probability distribution. However, an accuracy of M digits (i.e. an accuracy up to 2^{-M}) cannot be achieved with a polynomial number of samples.

We will now explore why simulating quantum computation is - in general - hard. First of all, it is evident that if we look at a classical circuit as a quantum circuit then the AND, NOT and OR-gates will always transfer a computational basis state into another, rendering the implementation of e.g. the Hadamard gate impossible. Another approach is to use the description of the quantum circuit to calculate the product of the matrices $U_1 \dots U_m$. These will, in general, act on the whole state vector $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ of the system (even neglecting any potential workspace for now) which means that $U_i \in U(2^n)$ is a $2^n \times 2^n$

matrix.

$$U_i = \begin{pmatrix} \langle 00 \dots 00 | U | 0 \dots 00 \rangle & \dots & \langle 00 \dots 00 | U | 1 \dots 11 \rangle \\ \langle 00 \dots 01 | U | 0 \dots 00 \rangle & \dots & \\ \vdots & & \vdots \\ \langle 11 \dots 11 | U | 0 \dots 00 \rangle & \dots & \langle 11 \dots 11 | U | 1 \dots 11 \rangle \end{pmatrix}$$

As matrix multiplication takes $\mathcal{O}(n^3)$ steps, a classical circuit family computing this probability will need a number of steps scaling like $\mathcal{O}((2^n)^3)$ which is not bounded by any polynomial.

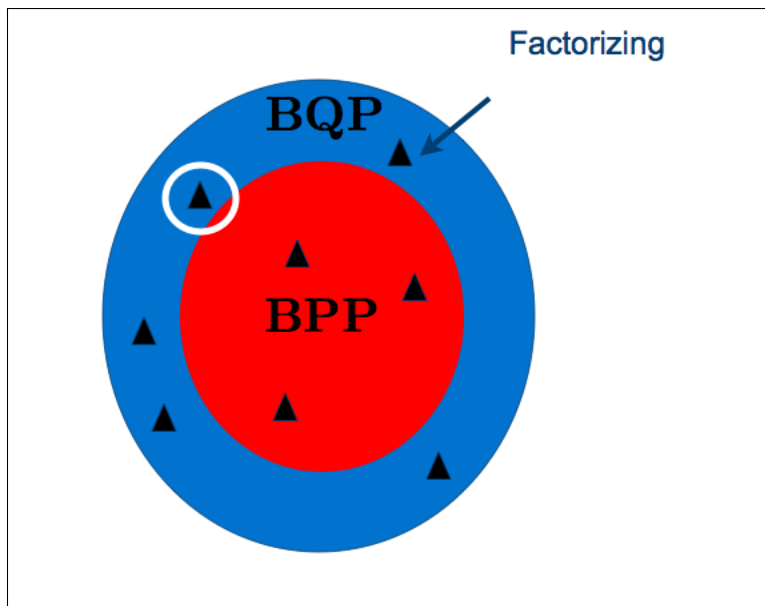


Figure 1: An overview of what is generally **believed** to be the relation between the complexity classes **BQP** and **BPP**. The black triangles are languages (i.e computational problems) where factorizing has been singled out as an example. Simulation is the attempt to pick a class of tasks in BQP (eg. the encircled one) and find a family of classical Boolean circuits such that one can show that the language is actually in **BPP**.

This section defined the notions of a quantum algorithm, compared weak versus strong classical simulation and demonstrated why it is hard to simulate a general quantum circuit. The findings point us to the following question: "What kind of restriction on the quantum computation can we impose in order to (classically) efficiently compute the resulting probability?". The next sections will give possible answers to this question.

3 Matchgates

This construction was introduced by Leslie Valiant in [2]. We will first review basic definitions for Pfaffians, related quantities and their connection to graphs. An important result

from the following section will be that there are classical algorithms which compute these quantities in $\mathcal{O}(\text{poly}(n))$ time. We will then prove in 3.2 that all 1-qubit unitaries and a certain set of 2-qubit gates can be expressed in terms of these so-called characters. In 3.3, we will then prove that computing the outcome probabilities remains feasible when we append multiple gates to each other. A short explanation of a Python program used to simulate the progress concludes the section.

3.1 The Character is computable in polynomial time

We will first review some basic elements of graph theory.

Definition 3.1. An *undirected, weighted graph* G is a triple (V, E, w) . Here, V is a set of *vertices* and we will label the vertices in this section by positive integers, i.e. $V \subset \mathbb{N}$. The set $E \subset V \times V$ are the *edges* of the graph and $w : E \rightarrow \mathbb{C}$ is a symmetric function (i.e. $w(i, j) = w(j, i)$), representing the *weights*. A pair of edges $(a, b), (c, d) \in E$ is *overlapping* if $a < c < b < d$ or $c < a < d < b$. A vertex v is *saturated* if there exists $u \in V$ with $u \neq v$ such that $w(v, u) \neq 0$. A graph is *complete* if for every pair of vertices the corresponding edge is part of the graph i.e. $E = V \times V$.

Remark 3.2. We will always consider complete graphs in this section but will omit edges of weight zero in the pictorial representation of a graph.

There is an equivalent way of expressing complete graphs in terms of antisymmetric matrices.

Definition 3.3. The *(graph-)matrix* of a complete graph with n vertices is an $n \times n$ square matrix G with entries defined as

$$G_{ij} = \begin{cases} w(i, j) & \text{if } i < j \\ -w(i, j) & \text{if } i > j \\ 0 & \text{if } i = j \end{cases}$$

We will often employ the letter G for denoting both the graph and its matrix. See example 3.7 for an illustration.

We observe that the graph-matrix is skew-symmetric by construction. For these matrices, we can define the *Pfaffian*:

Definition 3.4. The *Pfaffian* of a skew-symmetric $n \times n$ matrix G is

$$\text{Pf}(G) := \begin{cases} \frac{1}{2^k k!} \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^k G_{\sigma(2i-1), \sigma(2i)} & \text{if } n \text{ is even} \\ 0 & \text{if } n \text{ is odd} \end{cases}$$

where $k = \frac{n}{2}$ for legibility.

Definition 3.5. A *perfect matching* of a graph $G = \{V, E, W\}$ is a set of edges $P \subset E$ such that for every vertex $v \in V$ in graph $G' := \{V, P, W\}$ there is exactly one edge $e \in P$ saturating it. The *sign* of a perfect matching $\text{sgn}(P)$ is $+1$ if the number of overlaps as defined above is even and -1 if it is odd.

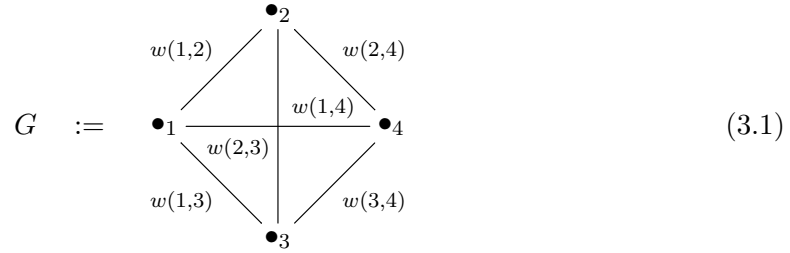
Lemma 3.6. The Pfaffian of a skew-symmetric matrix G is related to its graph via

$$Pf(G) = \sum_{\text{all perfect matchings } P_i \text{ within } G} sgn(P_i) \prod_{(v_1, v_2) \in P_i} w(v_1, v_2)$$

Proof. Compare each monomial to a perfect matching or refer to [2] for a short explanation. \square

Note that a graph containing an odd number of vertices cannot have any perfect matchings and the Pfaffian vanishes naturally. Let us illustrate these definitions with an example.

Example 3.7. Consider the graph shown below



The set of vertices and edges is

$$V = \{1, 2, 3, 4\} \quad (3.2)$$

$$E = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\} \quad (3.3)$$

and the weight function is not specified further. Hence, the corresponding matrix is given by

$$G = \begin{pmatrix} 0 & w(1, 2) & w(1, 3) & w(1, 4) \\ -w(1, 2) & 0 & w(2, 3) & w(2, 4) \\ -w(1, 3) & -w(2, 3) & 0 & w(3, 4) \\ -w(1, 4) & -w(2, 4) & -w(3, 4) & 0 \end{pmatrix} \quad (3.4)$$

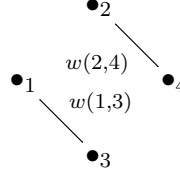
The Pfaffian of this even-dimensional graph-matrix is now computable via perfect matchings. The first perfect matching is



To determine the sign of this matching, we write the vertices on a line

$$\bullet_1 \quad \text{---} \quad \bullet_2 \quad \bullet_3 \quad \text{---} \quad \bullet_4 \quad (3.6)$$

and count the overlaps. Since we count an even number (zero), the first term in the Pfaffian is $+w(1,2)w(3,4)$. We proceed similarly with the remaining perfect matchings



$$(3.7)$$



$$(3.8)$$

$$\rightarrow -w(1,3)w(2,4)$$

and



$$(3.9)$$



$$(3.10)$$

$$\rightarrow +w(1,4)w(2,3)$$

so that we conclude the Pfaffian to be

$$Pf(G) = w(1,2)w(3,4) - w(1,3)w(2,4) + w(1,4)w(2,3) \quad (3.11)$$

This quantity is of major interest for what follows since it has the following property.

Lemma 3.8. For a skew-symmetric matrix G it holds that

$$Pf^2(G) = \det(G)$$

Proof. See [8]. □

Theorem 3.9. *There is a classical algorithm as defined in (2.1) that outputs the Pfaffian of a planar graph in a number of steps that scales polynomially with the graph size n and the desired accuracy k i.e. $T((C_k)_n) = \mathcal{O}(\text{poly}(n, k))$*

Proof. We will not give the perfect-matching (the so-called FKT) algorithm here but refer the interested reader to [9, 10, 11]. An intuitive reasoning goes as follows: it is known, that the determinant of an $n \times n$ -matrix is computable in $\mathcal{O}(n^3)$ steps. By the above Lemma, the Pfaffian should also be a polynomial-time computable quantity. \square

We have arrived at a nontrivial quantity that a priori consists of an exponential number of terms but can be computed in polynomial time. Since our strategy is to relate a single 1-qubit gate to a single graph we need to be able to draw four independent quantities from the graph, which are dependent on the input and output we are interested in. We hence need to impose more structure on our graph which leads us to

Definition 3.10. A *Matchgate* is a set $\Gamma(G, X, Y, T)$ of a graph $G(V, E, w)$ and disjoint subsets of $V \supset X, Y, T$; $X \cap Y = X \cap T = T \cap Y = \emptyset$. We call X and Y the *input* and *output set*, respectively. The set T is called the *set of omittable vertices*.

We acquire the needed flexibility by cutting certain vertices out of our graph. The first instance is reflected in the notion of

Definition 3.11. The *Pfaffian Sum* (PfS) of a Matchgate $\Gamma(G(V, E, w), X, Y, T)$ is defined as

$$PfS(\Gamma) := \sum_{S \subseteq T} Pf(G(V - S, E, w))$$

where the sum runs over all possible subsets of the set of omittable vertices T (including the empty set and the full set) and E and w are restricted to vertices in $V - S$ in the respective terms of the sum. We should note that the Pfaffian Sum in general consists of an exponential number of Pfaffians. However, by modifying the respective graph suitably, we can find a way to reduce the problem of finding the Pfaffian Sum to computing one single Pfaffian only ([2]).

Secondly, we require to obtain different quantities depending on various possibilities of input and output. This leads us to

Definition 3.12. The *naked Character* ([13]) of a Matchgate $\Gamma(G(V, E, w), X, Y, T)$ with respect to a given input and output $Z \subseteq X \cup Y$ is

$$\begin{aligned} \chi(\Gamma, Z) &= PfS(\Gamma(G(V - Z, E, w), X, Y, T)) \\ &= \sum_{S \subseteq T} Pf(G(V - Z - S, E, w)) \end{aligned}$$

where again, E and w are accordingly restricted.

The character is *naked* because in the original definition, Valiant included a *modifier* μ that accounted for sign flips that arose by considering 2-qubit gates in a circuit in isolation. We are ultimately interested in simulating *matchcircuits* which are not externally connected. The modifier for 1-qubit gates is also always $+1$. The only case where the modifier is

needed is when considering SWAP-like transition amplitudes in isolated 2-qubit gates and we will make a remark at the appropriate point in the discussion. For now, when we refer to character we will always mean the naked version.

We can now form matrices of these characters that resemble unitary operations:

Definition 3.13. The *Character Matrix* of a Matchgate $\Gamma(G(V, E, w), X, Y, T)$ is

$$\begin{pmatrix} \chi(\Gamma, \{\}) & \chi(\Gamma, \{y\} \subset Y) & \cdots \\ \chi(\Gamma, \{x\} \subset X) & \chi(\Gamma, \{x, y\} \subset X \cup Y) & \\ \vdots & & \end{pmatrix}$$

where the first row comprises all possible subsets of output set Y and the first column comprises all possible subsets of input set X in their definition. The remaining entries of the Character Matrix are the unions naturally arising from the according sets in the first row and column.

Example 3.14. How does this look like for the graph in the previous example 3.7? We equip $G(V, E, w)$ with $X := 1 \subset V, Y := 4 \subset V$ and $T := 2 \subset V$ as required to make it a Matchgate. We first compute

$$\begin{aligned} \chi(\Gamma, \{\}) &= PfS(\Gamma(G(V - \{\}), E, w), X, Y, T) \\ &= \sum_{S \subseteq T = \{2\}} Pf(G(V - S, E, w)) \\ &= Pf(G - \{\}) + Pf(G(V - \{2\})) \\ &= Pf \left(\underbrace{\begin{pmatrix} \bullet 2 & & \\ w(1,2) & & w(2,4) \\ \bullet 1 & w(1,4) & \bullet 4 \\ w(2,3) & & \\ w(1,3) & & w(3,4) \\ & \bullet 3 & \end{pmatrix}}_{\text{as computed in (3.11)}} + Pf \left(\underbrace{\begin{pmatrix} \bullet 1 & & \bullet 4 \\ w(1,4) & & \\ w(1,3) & & w(3,4) \\ & \bullet 3 & \end{pmatrix}}_{\text{odd-dimensional} \rightarrow 0} \right) \right) \\ &= w(1,2)w(3,4) - w(1,3)w(2,4) + w(1,4)w(2,3) \end{aligned} \tag{3.12}$$

The other entries of the character matrix are

$$\begin{aligned}
\chi(\Gamma, \{1\}) &= PfS(\Gamma(G(V - \{1\}, E, w), X, Y, T)) \\
&= \sum_{S \subseteq T = \{2\}} Pf(G(V - \{1\} - S, E, w)) \\
&= Pf(G - \{1\}) + Pf(G(V - \{1, 2\})) \\
&= Pf \left(\begin{array}{c} \bullet 2 \\ | \quad \diagup w(2,4) \\ w(2,3) \quad \bullet 4 \\ | \quad \diagdown w(3,4) \\ \bullet 3 \end{array} \right) + Pf \left(\begin{array}{c} \bullet 4 \\ \diagdown w(3,4) \\ \bullet 3 \end{array} \right) \\
&\quad \underbrace{\hspace{10em}}_{\text{odd-dimensional} \rightarrow 0} \quad \underbrace{\hspace{10em}}_{\text{is its own only perfect matching} \rightarrow w(3,4)} \\
&= w(3, 4)
\end{aligned} \tag{3.13}$$

By symmetry $\chi(\Gamma, \{4\}) = w(1, 3)$ and finally

$$\begin{aligned}
\chi(\Gamma, \{1, 4\}) &= PfS(\Gamma(G(V - \{1, 4\}, E, w), X, Y, T)) \\
&= \sum_{S \subseteq T = \{2\}} Pf(G(V - \{1\} - S, E, w)) \\
&= Pf(G - \{1, 4\}) + Pf(G(V - \{1, 2, 4\})) \\
&= Pf \left(\begin{array}{c} \bullet 2 \\ | \quad w(2,3) \\ \bullet 3 \end{array} \right) + Pf \left(\begin{array}{c} \bullet 3 \end{array} \right) \\
&\quad \underbrace{\hspace{10em}}_{\text{is its own only perfect matching} \rightarrow w(2,3)} \quad \underbrace{\hspace{10em}}_{\text{odd-dimensional} \rightarrow 0} \\
&= w(2, 3)
\end{aligned} \tag{3.14}$$

so we end up with

$$\begin{aligned}
\chi(\Gamma) &= \begin{pmatrix} \chi(\Gamma, \{\}) & \chi(\Gamma, \{4\}) \\ \chi(\Gamma, \{1\}) & \chi(\Gamma, \{1, 4\}) \end{pmatrix} \\
&= \begin{pmatrix} w(1, 2)w(3, 4) - w(1, 3)w(2, 4) + w(1, 4)w(2, 3) & w(1, 3) \\ w(3, 4) & w(2, 3) \end{pmatrix}
\end{aligned} \tag{3.15}$$

3.2 Certain gates can be expressed as Character matrices

We are now prepared for constructing the connection between quantum gates and match-gates.

Theorem 3.15. (First Matchgate Theorem) For every $\begin{pmatrix} a & b \\ c & d \end{pmatrix} =: U \in U(2)$ there exists a Matchgate $\Gamma(G(V, E, w), X, Y, T)$ with $V = \{1, 2, 3, 4\}$, $X = \{1\}$, $Y = \{4\}$ and $T = \{2\}$ such that

$$U = \chi(\Gamma)$$

Proof. By construction: we know the character matrix of the constructed Matchgate is (3.15). Since U is non-singular, c and d cannot simultaneously be zero. Define the weight function w to fulfill

$$\begin{aligned} w(1, 2) &= \frac{a}{c} & w(2, 4) &= 0 \\ w(1, 3) &= b & w(2, 3) &= d \\ w(1, 4) &= 0 & w(3, 4) &= c \end{aligned} \tag{3.16}$$

if c is non-zero and

$$\begin{aligned} w(1, 2) &= 0 & w(2, 3) &= d \\ w(1, 3) &= b & w(2, 4) &= 0 \\ w(1, 4) &= \frac{a}{d} & w(3, 4) &= c \end{aligned} \tag{3.17}$$

if c is zero but d is not. In the first case, we see that the constructed Matchgate with the graph



has the desired Character Matrix and the second case follows analogously. \square

Let us briefly reflect on the consequences of this theorem. If we are given a collection of n disjoint circuits that all comprise exactly one 1-qubit gate, and denote the input into the ensemble of circuits as \mathbf{x} , then we can compute the probability $|\langle \mathbf{y} | U_1 U_2 \cdots U_n | \mathbf{x} \rangle|^2$ of obtaining a given output \mathbf{y} classically in a number of steps that scales polynomially with the number of circuits that we are to evaluate. Note that U_1, U_2, \dots, U_n all act on separate circuits. It is readily observable that this has been an easy task from the outset and there are two main ingredients missing for universal quantum computation. Firstly, we need to be able to simulate quantum gates that act nontrivially on more than one qubit. Secondly, to model complex quantum circuits, a way of combining those gates such that they can act on a single qubit is required. We will start by fixing the first issue:

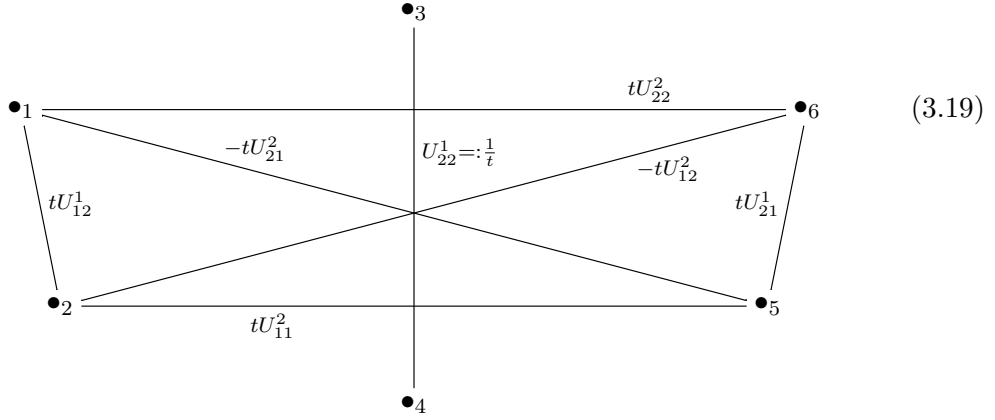
Theorem 3.16. (Second Matchgate Theorem) For every

$$\begin{pmatrix} U_{11}^1 & & & U_{12}^1 \\ & U_{11}^2 & U_{12}^2 & \\ & U_{21}^2 & U_{22}^2 & \\ U_{21}^1 & & & U_{22}^1 \end{pmatrix} =: U \in U(4)$$

with $U^1, U^2 \in U(2)$ and $\det(U^1) = \det(U^2)$ there exists a Matchgate $\Gamma(G(V, E, w), X, Y, T)$ with $V = \{1, 2, 3, 4, 5, 6\}$, $X = \{1, 2\}$, $Y = \{5, 6\}$ and $T = \{3, 4\}$ such that

$$U = \chi(\Gamma)$$

Proof. Again, we explicitly construct a Matchgate with the desired properties. Without loss of generality we assume $U_{22}^1 \neq 0$, since again, by unitarity, no row or column can consist of only zeros and if the zero happens to be on the bottom right, we just choose $U_{21}^1 \neq 0$ and the proof goes through similarly. We define $t = \frac{1}{U_{22}^1}$ and claim that the following Matchgate (the graph of which we denote by G_6)



equipped with X, Y and T as in the proof has the desired Character Matrix. This Matchgate has no omissible vertices which means that for all subgraphs $G' \subseteq G_6$ it holds that $PfS(G') = Pf(G')$. This fact immediately leads us to the conclusion that $\chi(\Gamma, Z) = 0$ for all sets Z that have an odd number of elements. That is also why we needed an omissible vertex in our 1-qubit case, otherwise, we would have restricted ourselves to diagonal matrices. In this case it simply means that the character matrix reduces to

$$\begin{aligned} \chi(\Gamma) &= \begin{pmatrix} \chi(\Gamma, \{3\}) & \chi(\Gamma, \{6\}) & \chi(\Gamma, \{5\}) & \chi(\Gamma, \{5, 6\}) \\ \chi(\Gamma, \{1\}) & \chi(\Gamma, \{1, 6\}) & \chi(\Gamma, \{1, 5\}) & \chi(\Gamma, \{1, 5, 6\}) \\ \chi(\Gamma, \{2\}) & \chi(\Gamma, \{2, 6\}) & \chi(\Gamma, \{2, 5\}) & \chi(\Gamma, \{2, 5, 6\}) \\ \chi(\Gamma, \{1, 2\}) & \chi(\Gamma, \{1, 2, 6\}) & \chi(\Gamma, \{1, 2, 5\}) & \chi(\Gamma, \{1, 2, 5, 6\}) \end{pmatrix} \\ &\rightarrow \begin{pmatrix} \chi(\Gamma, \{3\}) & & & \chi(\Gamma, \{5, 6\}) \\ & \chi(\Gamma, \{1, 6\}) & \chi(\Gamma, \{1, 5\}) & \\ & \chi(\Gamma, \{2, 6\}) & \chi(\Gamma, \{2, 5\}) & \\ \chi(\Gamma, \{1, 2\}) & & & \chi(\Gamma, \{1, 2, 5, 6\}) \end{pmatrix} \end{aligned} \quad (3.20)$$

The computation of these characters is again quite straightforward. The way of computing $\chi(\Gamma, \{\})$ is now illustrated: Note that our prosed graph exhibits an interesting feature regarding vertices 3 and 4. These are *only* connected with each other but disconnected from the rest of the graph. This means that if we want to find a perfect matching then it will surely contain $w(3, 4)$. Therefore, we can simplify the computation as follows:

$$\begin{aligned}
\chi(\Gamma, \emptyset) &= Pf \left(\begin{array}{c} \bullet 3 \\ \begin{array}{ccc} \bullet 1 & & \bullet 6 \\ & \text{---} tU_{21}^2 \text{---} & \\ & U_{22}^1 =: \frac{1}{t} & \\ & \text{---} -tU_{12}^2 \text{---} & \\ \bullet 2 & & \bullet 5 \\ & \text{---} tU_{11}^2 \text{---} & \end{array} \\ \bullet 4 \end{array} \right) \\
&= \underbrace{w(3, 4)}_{= \frac{1}{t}} Pf \left(\begin{array}{ccc} \bullet 1 & & \bullet 6 \\ & \text{---} tU_{21}^2 \text{---} & \\ & \text{---} -tU_{12}^2 \text{---} & \\ \bullet 2 & & \bullet 5 \\ & \text{---} tU_{11}^2 \text{---} & \end{array} \right) \\
&\quad \underbrace{\hspace{10em}}_{\text{Pfaffian of a complete 4-graph computed in (3.11) with } 5, 6 \rightarrow 3, 4} \\
&= \frac{1}{t} \left[w(1, 2)w(5, 6) - w(1, 5)w(2, 6) + w(1, 6)w(2, 5) \right] \\
&= \frac{1}{t} \left[\underbrace{t^2 U_{12}^1 t U_{21}^1}_{= t^2 (U_{11}^1 U_{22}^1 - \det(U^1))} \quad \underbrace{-t U_{21}^2 t U_{12}^2 + t U_{22}^2 t U_{11}^2}_{= t^2 \det U^2} \right] \\
&= t U_{11}^1 U_{22}^1 \\
&= U_{11}^1 \tag{3.21}
\end{aligned}$$

where we have used $\det(U^1) = \det(U^2)$ and $t = \frac{1}{U_{22}^1}$ in the last two lines. We see that the determinant condition is essential to the success of this approach. Indeed, we are not even referring to U_{11}^1 in the construction of the Matchgate.

It is relatively straightforward to check the remaining entries of the Character Matrix:

$$\begin{aligned}
\chi(\Gamma, \{1, 6\}) &= \underbrace{w(3, 4)}_{=\frac{1}{t}} \underbrace{Pf \left(\begin{array}{c} \bullet_5 \\ \nearrow tU_{11}^2 \\ \bullet_2 \end{array} \right)}_{tU_1^2} \\
&= U_{11}^2
\end{aligned} \tag{3.22}$$

And by symmetry:

$$\begin{aligned}
\chi(\Gamma, \{2, 6\}) &= w(3, 4)w(1, 5) \\
&= -U_{21}^2
\end{aligned} \tag{3.23}$$

$$\begin{aligned}
\chi(\Gamma, \{1, 5\}) &= w(3, 4)w(2, 6) \\
&= -U_{12}^2
\end{aligned} \tag{3.24}$$

$$\begin{aligned}
\chi(\Gamma, \{2, 5\}) &= w(3, 4)w(1, 6) \\
&= U_{22}^2
\end{aligned} \tag{3.25}$$

and

$$\begin{aligned}
\chi(\Gamma, \{1, 2, 5, 6\}) &= w(3, 4) \\
&= U_{22}^1
\end{aligned} \tag{3.26}$$

Note the two minus signs in the off-diagonal entries of the inner matrix. This is exactly the result of considering the naked character instead of the character. In appendix A, it is derived that only for those choices of Z , a negative character arises and hence, by considering the character including the modifier, we see that this matchgate indeed offers a description of the given unitary. Alternatively, one can skip straight ahead to section 3.3 as we can then consider the 2-qubit gate to be embedded in a *matchcircuit* (which is always the case for computational purposes). The character of a matchcircuit is equal to its naked character since it is not externally connected. \square

The second matchgate theorem does not hold for arbitrary matrices. In fact, [2] refers to a proof of the fact that all matchgates satisfy the so called *matchgate identities* which imply that we cannot find a graph for general unitary 2-qubit operations. Let us now examine the properties of the unitaries that *do* have a matchgate representation. Consider the way these matrices act on the computational basis states of a 2-qubit system:

$$\begin{pmatrix} U_{11}^1 & & & U_{12}^1 \\ & U_{11}^2 & U_{12}^2 & \\ & U_{21}^2 & U_{22}^2 & \\ U_{21}^1 & & & U_{22}^1 \end{pmatrix} \begin{pmatrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{pmatrix}$$

Inspired by the number of 1's in the respective states we call $|00\rangle$ and $|11\rangle$ *even* and $|01\rangle$ and $|10\rangle$ *odd parity* states. The matrices we succeeded to simulate are hence parity preserving.

Combine this with the determinant condition and we arrive at the following conclusion: For every $U \in U(4)$ that we succeeded to simulate with the above construction it holds that $U \in SU(2) \oplus SU(2)$ where the $SU(2)$'s represent the even and odd parity subspaces.

Remark 3.17. We can assume both determinants to be $+1$ since if both are -1 we consider $e^{\frac{\pi i}{4}} U$, the determinant switches from -1 to $+1$ and the overall phase does not affect the resulting probability distribution.

3.3 Circuits are expressible as Character matrices

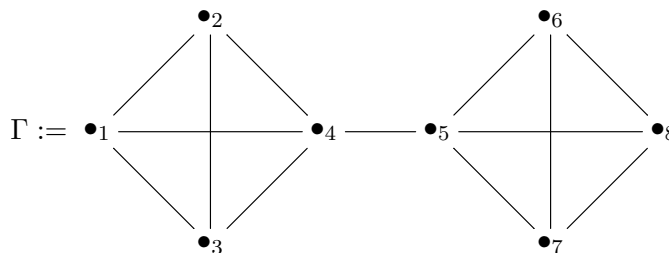
Until now, we have been considering systems of quantum circuits comprising of one gate each only. This is barely of any interest for any computation as we want to be able to perform the unitary transformations discussed in section (3.2) in succession.

Theorem 3.18. (*Matchcircuit Theorem*) Let $\hat{U}_1 \dots \hat{U}_n$ be a set of quantum gates where each \hat{U}_i is either

- a 1-qubit gate fulfilling the requirements of theorem 3.15 and acting on the first qubit line or
- a 2-qubit gate that fulfills the requirements of theorem 3.16 and acts on nearest neighbour qubit lines.

Let $U_i = (\hat{U}_i)_{\text{relevant qubit lines}} \otimes I$ and consider a circuit $U = U_1 \dots U_m$. Then there is a matchgate Γ such that $U = \chi(\Gamma)$. We call this composite matchgate a matchcircuit.

Proof. Again we construct explicitly. For clarity, we give the case of two 1-qubit matchgates and the generalization is straightforward. Let us illustrate the construction by a figure:



Here, all internal weights are the same as for the single gates, the so called *linking edge* is defined to have $w(4, 5) = 1$ and all other weights of edges connecting the separate graphs are zero. The input and output sets are $X = X_1$ and $Y = Y_2$ and the set of omittable vertices is $T = T_1 \cup T_2 = \{2, 6\}$. The corresponding quantum circuit is again a 2×2 unitary matrix and we will now establish how to compute the entries of this matrix (which are the all-important transition amplitudes). For clarity, we do not give a general proof but rather illustrate an example that is easily generalized. Consider $\chi(\Gamma, \{\})$. We can divide all possible perfect matchings into two groups: the first group does not contains the edge $(4, 5)$ and the second does. The resulting character will then just be a sum of all

appropriate Pfaffian sums of the first group plus all Pfaffian Sums of the other group. For all perfect matchings in the first group, the two graphs will not be connected and hence we have to take every perfect matching in graph one and multiply with every perfect matching in graph two. For the perfect matchings of the second group, 4 and 5 are already saturated by the linking edge and must hence be removed before starting to look for matchings (again in the separate graphs). We write

$$\begin{aligned}
\chi(\Gamma, \{\}) &= \chi \left(\begin{array}{c} \bullet 2 \\ \diagup \quad \diagdown \\ \bullet 1 \quad \bullet 4 \\ \diagdown \quad \diagup \\ \bullet 3 \end{array} \right) \chi \left(\begin{array}{c} \bullet 6 \\ \diagup \quad \diagdown \\ \bullet 5 \quad \bullet 8 \\ \diagdown \quad \diagup \\ \bullet 7 \end{array} \right) + \chi \left(\begin{array}{c} \bullet 2 \\ \diagup \quad \diagdown \\ \bullet 1 \quad \bullet 3 \\ \diagdown \quad \diagup \\ \bullet 7 \end{array} \right) \chi \left(\begin{array}{c} \bullet 6 \\ \diagup \quad \diagdown \\ \bullet 8 \\ \diagdown \quad \diagup \\ \bullet 7 \end{array} \right) \underbrace{w(4, 5)}_{=1} \\
&= \chi(\Gamma_1, \{\})\chi(\Gamma_2, \{\}) + \chi(\Gamma_1, \{4\})\chi(\Gamma_2, \{5\}) \\
&\stackrel{(3.15)}{=} \sum_{i=1}^2 \underbrace{\begin{pmatrix} \chi(\Gamma_1, \{\}) & \chi(\Gamma_1, \{4\}) \\ \chi(\Gamma_1, \{1\}) & \chi(\Gamma_1, \{1, 4\}) \end{pmatrix}}_{U_1} \underbrace{\begin{pmatrix} \chi(\Gamma_2, \{\}) & \chi(\Gamma_2, \{8\}) \\ \chi(\Gamma_2, \{5\}) & \chi(\Gamma_2, \{5, 8\}) \end{pmatrix}}_{U_2} \underbrace{i}_{i1} \\
&= [U_1 U_2]_{11}
\end{aligned} \tag{3.27}$$

Similarly, one obtains the other entries of the character matrix to conclude

$$U_1 U_2 = \begin{pmatrix} \chi(\Gamma, \{\}) & \chi(\Gamma, \{8\}) \\ \chi(\Gamma, \{1\}) & \chi(\Gamma, \{1, 8\}) \end{pmatrix} \tag{3.28}$$

The proof for 2-qubit matchgates proceeds analogously if they act on nearest neighbour lines (cf. the end of the proof). Now construct $\Gamma(G(V, E, w), X, Y, T)$ by the following procedure: we begin by introducing n input vertices labeled $(1, \dots, n)$, where qubit line l corresponds to input vertex $n - l$ (*decreasing* order). Define X as the union of these vertices. For each gate U_i we now do the following:

1. Add a number of vertices corresponding to the number of qubit lines that the gate acts on and connect them with a linking edge of weight one.
2. Attach the rest of the gate to these input vertices. If this gate contains omissible vertices T_i set $T \rightarrow T \cup T_i$.
3. For each qubit line, add a vertex where this time the labeling is *increasing* with the qubit line number. Again, connect the vertices with linking edges of weight one.
4. If this was *not* the last gate in the circuit, for each qubit line, add a vertex where this time the labeling is *decreasing* with the qubit line number.

Similar to X , set $Y = \bigcup_{k=1}^n \{|V| - n + k\}$. Now to compute the transition amplitude $\langle \mathbf{x} | U | \mathbf{y} \rangle$, remove all *input* vertices which correspond to qubit lines with input fixed to 0 and do the same for the output vertices. Hence, all remaining input and output vertices *have* to be saturated by the first gate on the respective qubit lines, thus singling out a single row of the first gate U_1 and a single column of the last gate U_m . It is clear that the

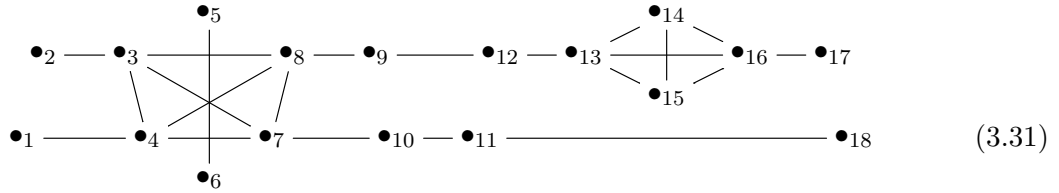
intermediate (or *linking*) edges - all set to 1 - do not influence the Pfaffian of the resulting graph. Therefore, since we have shown above that appending gates is equivalent to matrix multiplication we have that

$$\begin{aligned}\chi(\Gamma \setminus (I_{\mathbf{x}} \cup O_{\mathbf{y}})) &= \sum_{k_1=1}^n \cdots \sum_{k_{m-1}=1}^n (U_1)_{ik_1} (U_2)_{k_1 k_2} \cdots (U_m)_{k_{m-1} j} \\ &= (U_1 U_2 \cdots U_m)_{ij} \\ &= \langle \mathbf{x} | U | \mathbf{y} \rangle\end{aligned}\tag{3.29}$$

where $I_{\mathbf{x}}$ and $O_{\mathbf{y}}$ are the sets of vertices that correspond to a given input and output:

$$I_{\mathbf{x}} = \bigcup_{k=1}^n \{\mathbf{x}_{n-k}(n-k)\} \quad O_{\mathbf{y}} = \bigcup_{k=1}^n \{\mathbf{y}_k(|V| - n + k)\}\tag{3.30}$$

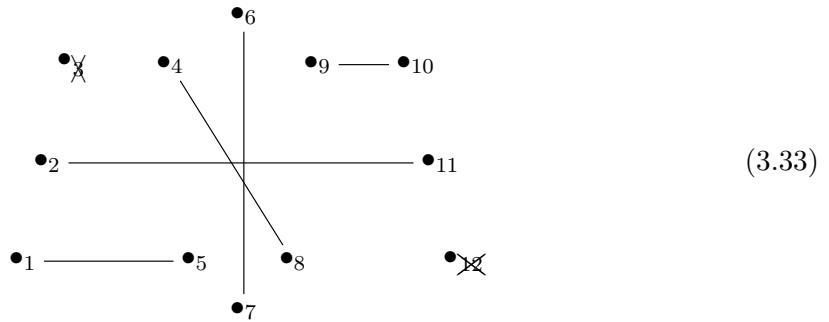
Hence



is the transcription of the circuit $U = U_1 U_2$

$$\begin{array}{c} |1\rangle \text{ --- } \boxed{U_2} \text{ --- } \boxed{U_1} \text{ --- } |1\rangle \\ |1\rangle \text{ --- } \boxed{U_1} \text{ --- } |1\rangle \end{array}\tag{3.32}$$

We will now show why we imposed additional constraints on the gates in the statement of the theorem. Consider first the case of a 2-qubit matchgate acting on non-nearest neighbour qubit lines 1 and 3. We want to compute the transition amplitude $\langle 110 | U | 011 \rangle$ for which we have to remove the input vertex on the first line and the output vertex on the third line. The only perfect matching corresponding to this value is shown below:



It is clear that the absolute value is not influenced by the presence of the intermediate qubit line. Taking a look at the overlaps though

(3.34)

we see that the edge $(2,11)$ introduces a minus sign which would not occur for a next-neighbour qubit gate. Hence, this gate does not act like $\hat{U}_{1,2} \otimes I_3$ but rather like $\hat{U}_{1,2} \otimes Z_3$, introducing a minus sign if the input of qubit 3 is fixed to 1.

Finally, let us investigate why a 1-qubit gate on lines beyond the first gives rise to sign errors. Consider the transition amplitude $\langle 11|X|10\rangle = 1$:

(3.35)

Again, by looking at the overlaps

(3.36)

we find out that the external qubit line introduces an overlap such that the gate does not act trivially on the first gate. Observe that this feature does not occur for 2-qubit gates acting on arbitrary qubit lines. As an example, consider a matchcircuit comprising a single 2-qubit gate acting on lines 2 and 3. For the transition amplitude $|111\rangle \rightarrow |111\rangle$, the overlap of the edge on the first qubit line with the internal edges is $(-1)^4 = 1$. Now for a 2-qubit gate, *we always have to remove an even number of vertices* as these gates are parity preserving. Hence no minus sign can arise.

Note that the additional minus signs only turn up when the intermediate (or in the 1-qubit gate *preceding*) qubit lines are fixed to 1. We can regard this as the gate acting with Z on those lines. We will later see that these overlaps are the matchgate analog for the initial Z gates in the *Jordan-Wigner representation* (cf. section 4.3). \square

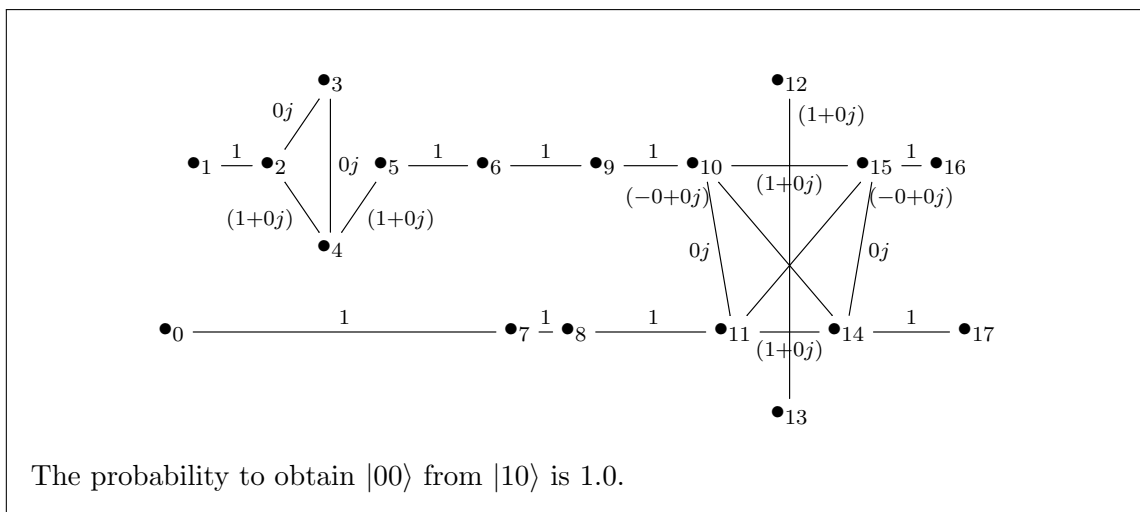
3.4 A PYTHON routine to simulate Matchcircuits

A routine to simulate Matchgate circuits is available at

<https://github.com/henrik-dreyer/MatchgateSimulator>

It relies on the effective computation of the Pfaffian which is realised in the package created in [14] (that is required to run the program and can be obtained from the authors website). MatchgateSimulator.py takes an input file and an input and output vector and creates a Latex-file that contains a picture of the graph, its matrix and the inquired probability.

Setting up the file manually is cumbersome, which is why CircuitCreator.py walks us through the steps of creating a quantum circuit one-by-one. The circuit file could also be generated algorithmically for a given family of quantum circuits. The structure of the input file is as follows. Every number has a line for its own and the ordering is: number of qubits, number of 1-qubit gates, number of 2-qubit gates and then for every gate: 1 or 2 for the type, the first line that it acts on, then in succession the real and imaginary parts of $U_{11}, U_{12}, U_{21}, U_{22}$ for a 1-qubit gate and $U_{11}^1, U_{12}^1, U_{21}^1, U_{22}^1, U_{11}^2, U_{12}^2, U_{21}^2, U_{22}^2 \dots$ for 2-qubit gates. As a proof of principle the program is currently restricted to gates with non-zero “bottom-right”-entries. The file in appendix D creates a circuit consisting of an X and identity 2-qubit gate and creates this output (modulo graph matrix, rescaling and slight rearranging for better legibility):



This concludes our survey of the matchgate formalism for now.

4 Non-interacting fermions

This section is concerned with an alternative way of strongly simulating the quantum circuits of theorem 3.18. We will proceed as follows: in 4.1, it is shown that simulation of the time evolution of a system of identical fermions is hard and how it is related to the simulation of a quantum circuit. In 4.2, we will then investigate a special case of non-interacting particle-preserving systems which are simulatable in polynomial time. The notion of the *generator* of a group will be crucial since their number scales logarithmically with the group size. After that, we will carry this result over to quantum circuits by interpreting the time evolution operator as a quantum gate in 4.3. We will realise that the restriction in 4.2 was too strict in the context of quantum computation and will loosen it to the conservation of particle number mod 2 only. The result of 4.4 will be a strong simulation of 2-qubit quantum gates of the group $SU(2) \oplus SU(2)$ acting on nearest neighbours. Remarkably, this is identical to the result of section 3. Unlike the original derivation by Terhal and

DiVincenzo, we will carry out the computation in the basis of the usual creation and annihilation operators a and a^\dagger . Hence, we will finish the section by giving an overview of the derivation in terms of *Majorana fermions* in 4.5, which gives a more natural description of particle number mod 2-conservation. For the majority of the time, we will follow the derivations in [3] and [4].

4.1 Time evolution in an exponentially scaling space

Time evolution of a system is encoded in the Hamiltonian, which - in the language of second quantisation - is generally a combination of annihilation and creation operators a_i and a_i^\dagger . For a fermionic system these, obey the anticommutation relations

$$\begin{aligned}\{a_i, a_j\} &= \{a_i^\dagger, a_j^\dagger\} = 0 \\ \{a_i, a_j^\dagger\} &= \delta_{ij}\end{aligned}\tag{4.1}$$

For an n -level system, the highest order monomial that one can form from this algebra and that is *normal ordered* (i.e. has all creation operation left of all annihilation operators) has degree $2n$. The reason is that as soon as the expression contains any operator twice, one can commute one of these operators through to form a simpler operator (if the monomial contains the appropriate hermitian conjugate of what we aim to commute through) or the whole expression vanishes (if it does not). The dimension of the vector space A_n of monomials formed by the a s and a^\dagger s is 2^{2n} , since we can go through each of the $2n$ operators and ask "is this contained in the product?"; we will receive either yes or no as an answer ($2n$ times in total) and each of these monomials is an independent basis vector of the vector space. At every point in time, the system is completely described by the generators $a_i(t) = U^\dagger(t)a_i(0)U(t)$ and $a_i^\dagger(t) = U^\dagger(t)a_i^\dagger(0)U(t)$, in terms of their expansion in a basis of the aforementioned vector space. Writing down all 2^{2n} possible monomials into a vector we can write

$$a_i(t) = \begin{pmatrix} & & \\ & M(t) & \\ & & \end{pmatrix} \begin{pmatrix} a_1(0) \\ a_1^\dagger(0) \\ a_1^\dagger(0)a_1(0) \\ \vdots \end{pmatrix}\tag{4.2}$$

where M is now a $2^{2n} \times 2^{2n}$ matrix. If we could compute the coefficients of the generators at any given time in polynomial time, then simulation of this system would be easy. That is in general not the case and the crux lies within the time evolution operator of the system $U(t) = e^{-iHt}$. This is an exponential of operators (which are 2^n by 2^n matrices) and hence in general itself $2^n \times 2^n$ -dimensional. The elements generated by a s and a^\dagger s form a vector space that has 2^{2n} basis vectors into which the exponential can be expanded. We can hence keep track of 2^{2n} coefficients to describe the time evolution of an operator. Note also that in general, even though there are only $2n$ initial generators $a_1 \dots a_n, a_1^\dagger \dots a_n^\dagger$ even *their* time evolution will drift into a space where the coefficients of their expansion into the basis

vectors generated by them will be non-zero. This is remarkably similar to the problem of simulating evolution in a quantum circuit in section 2 and the question is raised whether it is possible to reduce the dimension of the time-advanced operators by restricting the time evolution operator in a meaningful way.

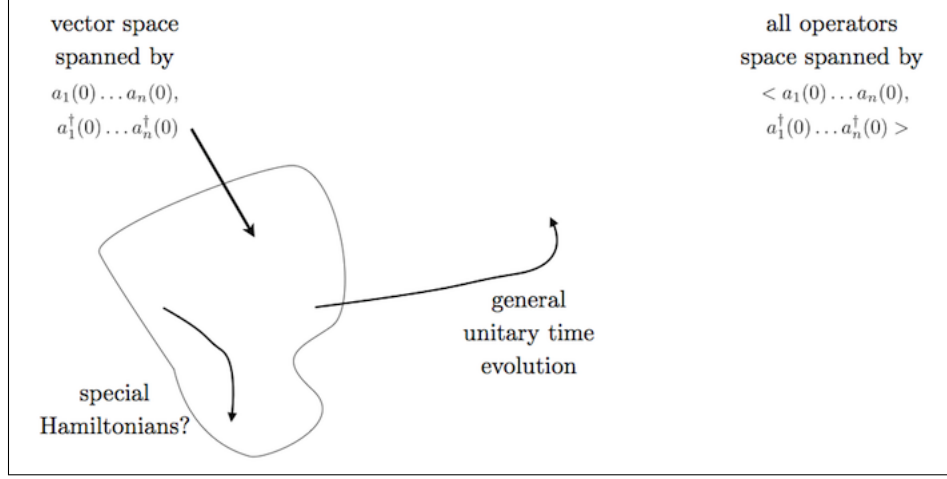


Figure 2: Certain kinds of Hamiltonians might allow us to keep track of the evolution. Note that the exact same picture would explain the effectiveness of simulating Clifford circuits via the stabilizer formalism (cf. [15]). The smaller area would be the Pauli group and the larger space the space of all gates. “Clifford gates” would replace “special Hamiltonians”.

If we look at any given mode of a system of identical fermions, this in a way resembles a qubit, since it can - due to the *Pauli* exclusion principle - only be occupied or unoccupied.

$$|01\rangle = \text{no fermion in the first mode and one fermion in second mode}$$

If the qubit basis states are represented by occupation numbers, what is the counterpart of a gate in a fermion system? In the latter, the state of the system evolves with the time evolution operator $U = e^{-iHt}$ where H is the Hamiltonian of the system. As the exponential of an antihermitian matrix (iH) is unitary, this operator fulfills all requirements to be interpreted as a qubit gate. What does t stand for in a quantum circuit? In complexity theory, we are concerned with polynomial vs. exponential *time* but what we really mean is the *number of gates* that are required for the computation. This observation suggests that a quantum gate in a circuit can be regarded as time evolution with constant timestep (say, 1, $U(1) = e^{-iH}$).

The question is raised whether there is a restricted fermionic system that *can* be classically simulated. We will follow [3]; for the original derivation which also relies on the computability of the Pfaffian see [4].

4.2 Certain systems evolve in a linearly scaling subspace

The general (particle-preserving) Hamiltonian of a fermionic system is given by

$$H = \sum_{ij} \langle i | A | j \rangle a_i^\dagger a_j + \sum_{ijkl} \langle i, j | V | k, m \rangle a_i^\dagger a_j^\dagger a_m a_k \quad (4.3)$$

It will become clear shortly why we consider first-order interactions only. We want to know whether we can track the evolution of such a system. It will be convenient to work in the *Heisenberg picture* since we will be concerned with the operators that the Hamiltonian consists of rather than arbitrary state vectors. We note that time evolution of an operator $a_k(t) = U^\dagger(t) a_k(0) U(t)$ with $U(t) = e^{-iHt}$ in this picture can be easily derived:

$$\frac{da_k(t)}{dt} = i[H, a_k(t)] \quad (4.4)$$

We observe that this differential equation has a closed analytical solution (namely just the exponential ansatz) if $[H, a_k(t)]$ is a multiple of $a_k(t)$. Is this the case? As the commutator seems of central importance in what is to follow, it is useful to note that.

$$\begin{aligned} [a_i, a_j] &= 2a_i a_j - \{a_i, a_j\} = 2a_i a_j \\ [a_i^\dagger, a_j^\dagger] &= 2a_i^\dagger a_j^\dagger - \{a_i^\dagger, a_j^\dagger\} = 2a_i^\dagger a_j^\dagger \\ [a_i^\dagger, a_j] &= 2a_i^\dagger a_j - \{a_i^\dagger, a_j\} = 2a_i^\dagger a_j - \delta_{ij} \end{aligned} \quad (4.5)$$

Consider first the non-interacting (NI) term.

$$\begin{aligned} e^{iHt} i[H_{\text{NI}}, a_k(0)] e^{-iHt} &= e^{iHt} \left(\sum_{ij} A_{ij} [a_i^\dagger a_j, a_k] \right) e^{-iHt} \\ &= i e^{iHt} \left(\sum_{ij} A_{ij} \left(a_i^\dagger [a_j, a_k] + [a_i^\dagger, a_k] a_j \right) \right) e^{-iHt} \\ &= i e^{iHt} \left(\sum_{ij} A_{ij} \left(a_i^\dagger 2a_j a_k + 2a_i^\dagger a_k a_j - \delta_{ik} a_j \right) \right) e^{-iHt} \\ &= i e^{iHt} \left(2 \sum_{ij} A_{ij} a_i^\dagger \underbrace{\{a_k, a_j\}}_{=0} - \sum_{ij} A_{ij} \delta_{ik} a_j \right) e^{-iHt} \\ &= -i e^{iHt} \left(\sum_j A_{kj} a_j \right) e^{-iHt} \\ &= -i e^{iHt} (\mathbf{A} \vec{a}(0))_k e^{-iHt} \\ &= -i [\mathbf{A} \vec{a}(t)]_k \end{aligned} \quad (4.6)$$

where $[\mathbf{A}\vec{a}(t)]_k$ denotes the k -th entry of the vector that results if we multiply the original matrix of coefficients in the Hamiltonian with the vector $(a_1 \cdots a_n)^T$. Similarly

$$i[H_{\text{NI}}, a_k^\dagger(t)] = -i \left[\mathbf{A}^T \vec{a}^\dagger(t) \right]_k \quad (4.7)$$

Note that thus far no mixing of the a s and a^\dagger s occurs in the time evolution. The commutator for the interaction (I) term is evaluated in appendix B:

$$[H_I, a_k(0)] = 2 \sum_{ghij} V_{ghij} a_g^\dagger a_k a_h^\dagger a_i a_j + \sum_{gij} V_{gkij} a_g^\dagger a_i a_j + \sum_{hij} V_{khij} a_h^\dagger a_i a_j \quad (4.8)$$

$$(4.9)$$

and it is hence highly non-trivial to evaluate such a non-linear differential equation. We will henceforth drop all interaction terms in the Hamiltonian and consider *quadratic* Hamiltonians only. The time evolution of the creation and annihilation operators under those Hamiltonians can now be written down by solving the ordinary matrix differential equations

$$\begin{aligned} \frac{d\vec{a}(t)}{dt} &= -i\mathbf{A}\vec{a}(t) \\ \rightarrow \vec{a}(t) &= e^{-i\mathbf{A}t}\vec{a}(0) \end{aligned} \quad (4.10a)$$

and

$$\begin{aligned} \frac{d\vec{a}^\dagger(t)}{dt} &= -i\mathbf{A}^T \vec{a}^\dagger(t) \\ \rightarrow \vec{a}^\dagger(t) &= e^{-i\mathbf{A}^T t} \vec{a}^\dagger(0) \end{aligned} \quad (4.10b)$$

where the exponential is now not an infinite sum of products of generators but a mere $2n \times 2n$ matrix exponential for which polynomial-time algorithms are well-known. We have hence achieved exactly what we were looking for. The time evolution of the a s and a^\dagger s does not throw them somewhere into the gigantic space of monomials that they generate but they are at all times given by a linear combination of the generators themselves. It is now time to apply these preliminaries to quantum computation.

4.3 The time evolution operator as a quantum gate

Consider the following 2-qubit quantum circuit.

$$\begin{array}{c} |0\rangle \text{ --- } \boxed{\phantom{e^{-iH_2}}} \text{ --- } ? \\ |0\rangle \text{ --- } \boxed{e^{-iH_1}} \text{ --- } \boxed{e^{-iH_2}} \end{array} \quad (4.11)$$

Now if that was the 2-qubit instance of a uniform family of quantum circuits designed to solve, say, a decision problem by measuring the first output line, then we would only need

to compute

$$\begin{aligned}
\langle \psi_{out} | Z_1 | \psi_{out} \rangle &= \langle \psi_{out} | a_1 a_1^\dagger - a_1^\dagger a_1 | \psi_{out} \rangle \\
&= \langle \psi_{in} | e^{iH_1} e^{iH_2} \left(a_1 a_1^\dagger - a_1^\dagger a_1 \right) e^{-iH_2} e^{-iH_1} | \psi_{in} \rangle \\
&= \langle \psi_{in} | e^{iH_1} e^{iH_2} a_1 e^{-iH_2} e^{-iH_1} e^{iH_1} e^{iH_2} a_1^\dagger e^{-iH_2} e^{-iH_1} | \psi_{in} \rangle \\
&\quad - \langle \psi_{in} | e^{iH_1} e^{iH_2} a_1^\dagger e^{-iH_2} e^{-iH_1} e^{iH_1} e^{iH_2} a_1 e^{-iH_2} e^{-iH_1} | \psi_{in} \rangle \\
&= \langle \psi_{in} | \left[\left(e^{-i\mathbf{A}_1} e^{-i\mathbf{A}_2} \vec{a} \right)_1, \left(e^{-i\mathbf{A}_1^T} e^{-i\mathbf{A}_2^T} \vec{a}^\dagger \right)_1 \right] | \psi_{in} \rangle \tag{4.12}
\end{aligned}$$

where $[\cdot, \cdot]$ denotes the commutator and we end up with a sum over expectation values of generators $a_1, a_1^\dagger, a_2, a_2^\dagger$. Under appropriate circumstances (it is sufficient for the generators to be product operators and the input states to be product states respectively [3]), those initial expectation values are poly-time computable. As the number of gates is polynomial by assumption, we have arrived at a polynomially scaling simulation of a quantum circuit. The condition that the a_i, a_i^\dagger are product operators is fulfilled by the *Jordan-Wigner representation* ([16]):

$$\begin{aligned}
a_1 &= \frac{X + iY}{2} I \cdots I & a_2 &= Z \frac{X + iY}{2} I \cdots I & a_n &= Z \cdots Z \frac{X + iY}{2} I \cdots I \\
a_1^\dagger &= \frac{X - iY}{2} I \cdots I & a_2^\dagger &= Z \frac{X - iY}{2} I \cdots I & a_n^\dagger &= Z \cdots Z \frac{X - iY}{2} I \cdots I
\end{aligned} \tag{4.13}$$

where the tensor product between the operators is understood. This representation also fulfills the canonical anti-commutation relations.

Remark 4.1. Notice that in this representation, the operators do not act trivially on preceding qubit lines. The Pauli-Z's are needed to fulfill the condition $\{a_i, a_{j \neq i}\} = 0$ as they anti-commute with both a and a^\dagger . Consider a quadratic operation on two non-nearest neighbour (n.n.) states, like $a_1^\dagger a_3 = (X - iY)Z(X + iY)$. In addition to acting as desired on states 1 and 3 it exerts a non-trivial action on state 2. We will henceforth restrict our discussion to action on nearest neighbours only.

The natural question to ask is: Which gates can be written as exponentials of these quadratic Hamiltonians? How far does this approach take us?

Let us look at the case of 2 qubits, i.e a fermionic system with 2 modes. The four possible quadratic monomials in this case are:

$$a_1^\dagger a_1 = E_{33} + E_{44} \tag{4.14a}$$

$$a_1^\dagger a_2 = E_{23} \tag{4.14b}$$

$$a_2^\dagger a_1 = E_{32} \tag{4.14c}$$

$$a_2^\dagger a_2 = E_{22} + E_{44} \tag{4.14d}$$

where the matrix representations are in the usual $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)^T$ -basis and E_{ij} is a 4×4 matrix with $(E_{ij})_{kl} = \delta_{ik} \delta_{jl}$. Since the Hamiltonian is hermitian, the general matrix

of coefficients must be

$$A = \begin{pmatrix} a & b \\ \bar{b} & d \end{pmatrix} \quad (4.15)$$

and hence the general quadratic operator is

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & a & b & 0 \\ 0 & \bar{b} & d & 0 \\ 0 & 0 & 0 & a+d \end{pmatrix} \quad (4.16)$$

Let us look at some properties of this matrix. First of all, it is singular; in fact the first row and column are zero. This singularity is physically intuitive, since - barring identity operations which simply add omittable constants to the Hamiltonian - *these operators leave us with no way of producing fermions without destroying others and vice-versa*. Hence, the $|00\rangle$ -state cannot be obtained from any other state and it is not possible for the system to leave the $|00\rangle$ -state either.

Secondly, the matrix preserves even and odd parity subspaces. In fact, preserving the number of fermions is much stronger than just preserving parity since *it deprives us of the possibility to add or remove a pair of fermions at a time*.

Furthermore, consider the case $a = -d$. In that case, iH is antihermitian and traceless. It follows that $iH \in \mathfrak{su}(2)$ where $\mathfrak{su}(2)$ is the Lie algebra of the group of special unitary matrices *acting on the odd parity states only*. Since the resulting matrix has three real degrees of freedom (a is real, b complex), this spans all of $\mathfrak{su}(2)$. By remembering that $e^{\mathfrak{su}(2)} = SU(2)$, we arrive at the conclusion that we can classically simulate the probability of obtaining 1 on the first qubit line for a circuit that comprises of two-qubit gates, each acting with a special unitary matrix on the odd parity subspace for product state input.

The use of this type of gates for quantum computation is very limited because we restricted ourselves too severely by enforcing the analogy to fermion physics. Constraining the quantum circuit to preserve the number of 0s and 1s during a computation is not a sensible constraint. It is hence natural to seek generalisations of the Hamiltonian above (which should however still allow classical simulation). The first option would be to add terms like a_1^\dagger or $a_1^\dagger a_1 a_2$ to give up fermion number conservation. From the discussion in section 3 we suspect conservation of particle number mod 2 to be key. The natural progression is to incorporate *pair creation and annihilation* $a_i^\dagger a_j^\dagger$ and $a_i a_j$ (with $i \neq j$).

4.4 Particle number mod 2-conserving Hamiltonians span $\mathfrak{su}(2) \oplus \mathfrak{su}(2)$

The general parity preserving quadratic Hamiltonian is

$$H = \sum_{ij} A_{ij} a_i^\dagger a_j + \sum_{ij} B_{ij} a_i^\dagger a_j^\dagger + \sum_{ij} C_{ij} a_i a_j \quad (4.17)$$

To assure that we can still express the time evolution of these new operators $a_i^\dagger a_j^\dagger$ and $a_i a_j$ in terms of the coefficients of their expansion into a linear combination of initial generators,

we compute

$$\begin{aligned}
\sum_{ij} B_{ij} [a_i^\dagger a_j^\dagger, a_k] &= \sum_{ij} B_{ij} \left(a_i^\dagger \underbrace{[a_j^\dagger, a_k]}_{=\delta_{jk}-2a_k a_j^\dagger} + \underbrace{[a_i^\dagger, a_k]}_{=\delta_{ik}-2a_k a_i^\dagger} a_j^\dagger \right) \\
&= \sum_{ij} B_{ij} (a_i^\dagger \delta_{jk}) + \sum_{ij} B_{ij} (a_j^\dagger \delta_{ik}) \\
&\quad - 2 \sum_{ij} B_{ij} \underbrace{(a_i^\dagger a_k a_j^\dagger + a_k a_i^\dagger a_j^\dagger)}_{=\{a_i^\dagger, a_k\} a_j^\dagger} \\
&= \left[(\mathbf{B}^T + \mathbf{B}) \vec{a}^\dagger \right]_k - 2 \sum_{ij} B_{ij} \delta_{ik} a_j^\dagger \\
&\quad \underbrace{\hspace{10em}}_{=[\mathbf{B} \vec{a}^\dagger]_k} \\
&= \left[(\mathbf{B}^T - \mathbf{B}) \vec{a}^\dagger \right]_k
\end{aligned} \tag{4.18}$$

and similarly

$$\sum_{ij} B_{ij} [a_i a_j, a_k^\dagger] = [(\mathbf{C}^T - \mathbf{C}) \vec{a}]_k \tag{4.19}$$

The commutator $[a_i^\dagger a_j^\dagger, a_k^\dagger]$ is naturally zero since all these creation operators anti-commute and hence commuting a_k^\dagger through an even number of operators (two) will result in the initial expression. The same holds for $[a_i a_j, a_k]$. Evolution under the general Hamiltonian is hence still constrained to the $2n$ dimensional subspace spanned by the a s and a^\dagger s and therefore classically simulable. Let us again look at what these gates are. Imposing hermiticity on the Hamiltonian we find

$$\begin{aligned}
H &= \sum_{ij} A_{ij} a_i^\dagger a_j + \sum_{ij} B_{ij} a_i^\dagger a_j^\dagger + \sum_{ij} C_{ij} a_i a_j \\
&\stackrel{!}{=} H^\dagger \\
&= \sum_{ij} \overline{A_{ij}} a_j^\dagger a_i + \sum_{ij} \overline{B_{ij}} a_j a_i + \sum_{ij} \overline{C_{ij}} a_j^\dagger a_i^\dagger \\
&\stackrel{i \leftrightarrow j}{=} \sum_{ij} \overline{A_{ji}} a_i^\dagger a_j + \sum_{ij} \overline{B_{ji}} a_i a_j + \sum_{ij} \overline{C_{ji}} a_i^\dagger a_j^\dagger
\end{aligned} \tag{4.20}$$

and since these operators are independent we can compare coefficients to obtain:

$$\begin{aligned}
\mathbf{A} &= \mathbf{A}^\dagger \quad (\text{as before}) \\
\mathbf{B} &= \mathbf{C}^\dagger
\end{aligned} \tag{4.21}$$

Moreover, since $a_i^2 = a_i^{\dagger 2} = 0$, the diagonal entries of the matrices \mathbf{B} and \mathbf{C} will not enter in the Hamiltonian and can hence without loss of generality be assumed to be zero. As a last observation, note that terms like $a_1 a_2$ anti-commute, so \mathbf{B} and \mathbf{C} are - again w.l.o.g. - antisymmetric. For clarity, consider the $n = 2$ case. The most general Hamiltonian is then

$$\begin{aligned}
H = & \begin{pmatrix} a_1^\dagger & a_2^\dagger \end{pmatrix} \begin{pmatrix} a & b \\ \bar{b} & d \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \\
& + \begin{pmatrix} a_1^\dagger & a_2^\dagger \end{pmatrix} \begin{pmatrix} 0 & \frac{c}{2} \\ -\frac{\bar{c}}{2} & 0 \end{pmatrix} \begin{pmatrix} a_1^\dagger \\ a_2^\dagger \end{pmatrix} \\
& + \begin{pmatrix} a_1 & a_2 \end{pmatrix} \begin{pmatrix} 0 & \frac{\bar{c}}{2} \\ -\frac{c}{2} & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}
\end{aligned} \tag{4.22}$$

Recalling the form of the number-preserving operators (4.14) and working out the two new "pair"-operators in the $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)^T$ -basis

$$\begin{aligned}
a_1^\dagger a_2^\dagger &= E_{41} \\
a_1 a_2 &= E_{14}
\end{aligned} \tag{4.23}$$

the general Hamiltonian in the computational basis is

$$H = \begin{pmatrix} \textcolor{blue}{0} & 0 & 0 & \textcolor{blue}{\bar{c}} \\ 0 & \textcolor{brown}{a} & \textcolor{brown}{b} & 0 \\ 0 & \textcolor{brown}{\bar{b}} & \textcolor{brown}{d} & 0 \\ \textcolor{blue}{c} & 0 & 0 & \textcolor{blue}{a+d} \end{pmatrix} \tag{4.24}$$

As this matrix is hermitian and still parity-preserving it holds that $iH \in \mathfrak{u}(2) \oplus \mathfrak{u}(2)$, the Lie-Algebra of the unitary group. In fact, as the coefficient of the number operator-like terms $a_i a_i^\dagger$ (the "energy of a particle in the i -th mode" so to speak) varies, so does the trace of the matrix and since $\det(e^{iH}) = e^{\text{Tr}(iH)}$, the determinant of the resulting gate attains every possible complex value with absolute value one. The real dimension of $\mathfrak{u}(2) \oplus \mathfrak{u}(2)$ is 8 and we are only spanning a 6-dimensional subspace (a and d each provide one real degree of freedom, whereas b and c - being complex numbers - give two). We conclude that a general 2-qubit parity-preserving gate cannot be simulated. But $\mathfrak{u}(2) \oplus \mathfrak{u}(2)$ has a 6-dimensional sub-Lie algebra, namely $\mathfrak{su}(2) \oplus \mathfrak{su}(2)$. Is it possible to simulate gates of this class?

We would have to construct a *traceless* Hamiltonian. The general Hamiltonian we wrote down is not traceless. However, this can be remedied. Note that (4.24) fulfills $\langle 00 | H | 00 \rangle = 0$. Physically this makes sense, since we used our gauge freedom to add constants to the Hamiltonian (this is exactly what normal ordering does: it sets the vacuum expectation value to zero). But in quantum computation $|00\rangle$ does not mean vacuum! In fact $\langle 00 | Z_1 | 00 \rangle = 1$. Hence, it is reasonable to add a term λI to our Hamiltonian and time evolution of the operators will not be affected. Therefore, if we are capable of simulating a circuit comprising of e^{iH_j} -gates we will most certainly be capable of simulating the same circuit with $H_j \rightarrow H_j + \lambda I$. Hence, fix $\lambda = -\frac{a+d}{2}$ (if we think about this parameter as

the "average energy of a state", it becomes evident how to generalize this to more than 2 dimensions). The modified Hamiltonian becomes

$$H = \begin{pmatrix} -\frac{a+d}{2} & 0 & 0 & \bar{c} \\ 0 & \frac{a-d}{2} & b & 0 \\ 0 & \bar{b} & \frac{d-a}{2} & 0 \\ c & 0 & 0 & \frac{a+d}{2} \end{pmatrix} \quad (4.25)$$

It is traceless, iH is still antihermitian and $e^{iH} \in SU(2) \oplus SU(2)$. We have proved that time evolution through *Gaussian gates* e^{iH} generated by Hamiltonians of the form (4.17) can be computed in polynomial time.

We have consequently managed to classically simulate the probability of measuring 1 on the first qubit of any quantum circuit comprising of n.n. parity preserving 2-qubit gates that fulfill the determinant condition (it need not be +1 as discussed in remark 3.17) with product state input.

4.5 Majorana fermions & particle number mod 2 conservation

This section will conclude by giving an overview of the original derivation ([3, 4]) of the results in section 4.4 and evoking some physical intuition behind it. Recall that we had to artificially extend the Hamiltonian from (4.3) to (4.17) by adding pair creation and annihilation operators. The arising formalism was characterised by an unaesthetic asymmetry between those operators and number preserving terms. Can all the parity-preserving operators be put on equal footing? A physical intuition helps us proceed: *Majorana fermions* are theoretical fermions, that are their own antiparticle. A process like e^+e^- or electron-hole-pair creation for their case means creating or annihilating two *identical* particles. We began our discussion with Hamiltonian (4.3) since a term like $a^\dagger a^\dagger$ is physically unthinkable as it increases the amount of charge in the universe. Majorana fermions - being their own antiparticles - are neutral and in a hypothetical system of Majorana fermions, *there is no reason why particle number should be conserved* (modulo 2) and the Hamiltonian should naturally involve pair creation and annihilation terms. How is such a system usually described? We make use of groundwork done by theoretical physicists and define for an n -level system:

$$c_{2k-1} := a_k^\dagger + a_k \quad c_{2k} := i(a_k^\dagger - a_k) \quad (k = 1 \dots n) \quad (4.26)$$

Since these are *linear* combination of a and a^\dagger , it is clear that the previously quadratic Hamiltonian (4.17) will also be quadratic in these operators.

$$H = i \sum_{ij}^{2n} h_{ij} c_i c_j \quad (4.27)$$

Considering the "scattering" and "pair" terms on equal footing facilitates the following computations tremendously. Note first that the c -operators fulfill (by definition and due

to the canonical anti-commutation relations of Dirac fermions)

$$\begin{aligned}\{c_i, c_j\} &= 2\delta_{ij} \\ \rightarrow [c_i, c_j] &= \{c_i, c_j\} - 2c_j c_i = 2(\delta_{ij} - c_j c_i)\end{aligned}\tag{4.28}$$

which is called the *Clifford algebra* and resembles the defining property of the γ -matrices in the Dirac equation. Then, similar to (4.22), $h_{ij} = -h_{ji}$, since the off-diagonal operators anticommute and they hence are not independent. Then

$$\begin{aligned}[H, c_k] &= \sum_{ij} h_{ij} [c_i c_j, c_k] \\ &= \sum_{ij} h_{ij} \left(c_i \underbrace{[c_j, c_k]}_{2(\delta_{jk} - c_k c_j)} + \underbrace{[c_i, c_k]}_{2(\delta_{ik} - c_k c_i)} c_j \right) \\ &= 2 \left(\sum_i h_{ik} c_i + \sum_j h_{kj} c_j - \sum_{ij} h_{ij} \{c_i, c_k\} c_j \right) \\ &= 2 \left(\sum_i \underbrace{(h_{ik} + h_{ki})}_0 c_i - 2 \sum_{kj} h_{kj} c_j \right) \\ &= -4 [\mathbf{h}\vec{c}]_k\end{aligned}\tag{4.29}$$

Also, by hermiticity, the h_{ij} 's must be real and we note that $h \in \mathfrak{so}(2n)$. This result is again similar to (4.10) but the more natural approach offers a geometrical interpretation. Indeed, time evolution through a Gaussian gate e^{-4h} is a rotation in the $2n$ -dimensional subspace spanned by the generators.

5 Establishing ties

In this section, we will give some thoughts about the analogy between the seemingly different models presented in sections 3 and 4.

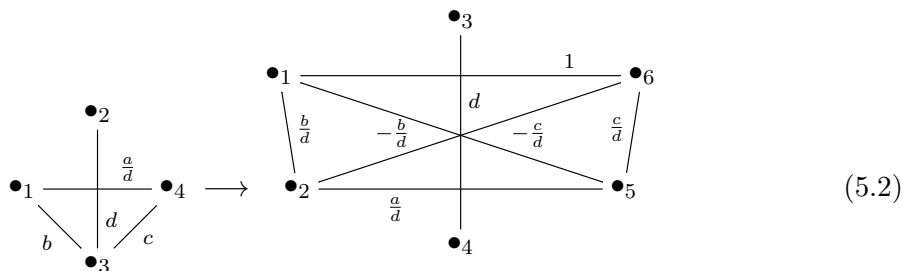
Curiously, the approach with non-interacting fermions extended as far as the Matchgate formalism. In fact, Terhal and DiVincenzo show that measurement probability (even beyond the first qubit line) takes the form of a Pfaffian of a polynomial-sized matrix for both number and parity conserving Hamiltonians [4]. The efficient computation of the Pfaffian is therefore crucial for both approaches.

An interesting analogy between the two formalisms reveals itself if we consider the fact that there was no equivalent of the 1-qubit Matchgates in the fermionic formalism. The

two qubit gates exert a connection to fermions:

$$\begin{aligned}
& \text{remove vertices 1 and 6} \rightarrow a_1^\dagger a_1 \\
& \text{remove vertices 1 and 5} \rightarrow a_1^\dagger a_2 \\
& \text{remove vertices 2 and 5} \rightarrow a_2^\dagger a_2 \\
& \text{remove vertices 2 and 6} \rightarrow a_2^\dagger a_1 \\
& \text{remove both input vertices} \rightarrow a_1^\dagger a_2^\dagger \\
& \text{remove both output vertices} \rightarrow a_1 a_2 \\
& \text{remove all or no vertices} \rightarrow I
\end{aligned} \tag{5.1}$$

There is no such analogue for 1-qubit gates. From the perspective of Matchgates, this makes sense since the representation of 1-qubit gates relies heavily on taking out single vertices for which there is no analogue in the quadratic Hamiltonian formalism. Note though, that by *encoding* a logical qubit into multiple physical qubits, we can still simulate the action of arbitrary 1-qubit gates (cf. [3]). Consider further an extension of the 1-qubit matchgate:



where fixing the input/output to 1 is done by removing *both* input/output vertices of the according 2-qubit graph. In the Hamiltonian formalism, a linear term like a_1 suddenly becomes quadratic (and hence allowable) if we introduce a n.n. ancilla to form $a_{\text{ancilla}}^\dagger a_1^\dagger$ which we discard immediately after the application of the gate.

It is also of interest to see what gives rise to the nearest-neighbour condition in both formalisms. Matchgates acting on non n.n. qubit lines acquire a sign flip due to the overlap of the intermediate linking edge with the external edges of the gate if the intermediate qubit is set to one (cf. theorem 3.18). The very same sign flip occurs in the Hamiltonian formalism due to the necessity of having Z -gates in the preceding lines in the Jordan-Wigner representation to establish the canonical anticommutation relations (cf. 4.13).

6 Minimal quantum aids & Universal computation

Our original aim was to investigate the boundaries between classical and quantum computation and to find out which ingredients are crucial to the conjectured additional power of quantum computers. The natural way to answer this question with the background we

have reviewed is to try and reestablish universal quantum computation and see *what* and *how much* one has to add. We will give a brief analysis based on arguments presented in [3], [17] and [18].

It is a known result that any family of quantum circuits with a polynomially scaling number of gates can be mapped to an equivalent circuit family comprising only a polynomial number of gates of the set $\bigcup_{i=1}^n \{U_i \in U(2)\} \cup \{CZ_{i,i+1,i \neq n} = \text{diag}(1, 1, 1, -1)\}$ where U_i is a 1-qubit unitary on qubit i acting with the identity on all other qubits and $CZ_{i,i+1}$ is the controlled- Z gate on n.n. qubits. It is established in [3] that any circuit family comprising polynomially many gates of the set $\bigcup_{i=1}^n \{U_i \in U(2)\} \cup \{G_{i,i+1,i \neq n}(A, B) \in SU(2) \oplus SU(2)\}$ is strongly simulable in sense of the matchgate formalism by encoding a logical qubit into four physical qubits. Because of the determinant condition (cf. theorem 3.16), we have $CZ \notin SU(2) \oplus SU(2)$. Note that

$$CZ = G(H, H)G(X, X) \underbrace{\tilde{G}(I, X)}_{\text{SWAP}} G(H, H) \quad (6.1)$$

where $G(A, B)$ denotes the 2-qubit gate acting with A on the even and with B on the odd parity space. The consequence is that universal quantum computation can be reestablished if one has access to an unlimited number of SWAP-gates. Note that the information in a given qubit line *can* travel throughout the circuit, *but not without affecting all intermediate qubit lines* (cf. 4.13). As an example, consider the fermionic SWAP gate $G(Z, X)$ and a quantum circuit operating with computational basis states only. The state of a qubit can be transmitted to distant qubit lines via this gate while applying the CZ gate to all intermediate qubits. In this sense, one can say that the gates that are simulable in the matchgate formalism are universal minus *long-range interaction* (in terms of qubit lines). Here, we are not talking about spacial separation but rather about the number of qubit lines that a given qubit line is allowed to interact with directly during the computation.

We can look at this restriction from a different angle: usually, one considers time restrictions (i.e. the number of gates should scale e.g. polynomially with the input). There is also the notion of *space-bounded* computation. For example we might restrict the circuit to perform a polynomial number of gates on a number of qubit lines that scales, say, logarithmically with the input. It has been shown that matchgate computation is in fact equivalent to universal but logarithmically space-bounded computation [17].

It is also interesting to see the subtlety of the transition from classically simulable to universal quantum computation. It has more recently been shown that adding *any* $G(A, B)$ that does not fulfil the determinant condition will reestablish universal quantum computing [18]. This result raises the question of whether one can actually form *intermediate* gates that are neither classically simulable nor universal for quantum computation.

It is notable that these simulatable circuits are capable of creating entangled states. This fact disproves the conjecture that entanglement is the one crucial ingredient of quantum computation.

7 Can the formalism be generalised to qudits?

The bit is the natural unit of computation because the natural way of representing information in a classical computer is by electric current which can either be flowing (1) or not (0). Other systems are being in use (for example the hexadecimal system), mainly to reduce the number of digits to represent a given number. We effectively gain nothing by changing the basis because there exists a map between circuits based on different bases that can be performed with polynomial overhead. The hexadecimal system just aggregates a couple of bits together but there are non-trivial (i.e. not powers of 2) bases that are of certain interest for quantum computation. While there is no need in a classical computation to, say, have the three states of "no current", "a little bit of current" and "loads of current", physical implementation of quantum computation is sometimes attempted in a larger basis (e.g. in [19]). One advantage is that if number of basis states is larger, less particles need to be controlled (e.g. 65 qubits have the same computational power as 41 qutrits (3-level-systems)). Also, some computational tasks are more natural in a different basis [20]. Lastly, properties of error correction and decoherence on qudit systems may diverge from what is known for qubits [21].

While we expect that a certain subset of qudit gates *is* classically simulable by mapping it to a qubit circuit in classical polynomial time and then simulate it with the discussed methods, we will attempt to find a classical simulatable qudit system directly.

We want to generalise the two equivalent formalisms developed before and we will for the sake of simplicity consider qutrits. The graph-theoretic matchgate formalism lacks an obvious generalization. Recall that we removed the vertices that corresponded to 0's in the computation and retained the 1's. It is hard to imagine a third state of a vertex beyond "there" and "not there".

The algebraic approach in section 4 promises to be more fruitful. The procedure we will follow is:

1. Look for operators on qutrits that generate a group that provides the terms of the Hamiltonian like the a and a^\dagger s or the X and Z s.
2. Consider gates that can be written as e^{iH} where H is now a $3^n \times 3^n$ matrix.
3. Assume a sensible form of the Hamiltonian and impose hermiticity in order to make the gate unitary.
4. Compute the time evolution by evaluating the commutators of the generators with the Hamiltonian. Does it map the generators to a linear combination of generators? Or do we get terms in the exponentially scaling group?

7.1 Pauli-operators on qutrits

In [22] we find a natural generalization of the X and Z matrices to three-level systems.

$$X = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \omega & 0 \\ 0 & 0 & \omega^2 \end{pmatrix} \quad Y = ZX \quad (7.1)$$

where $\omega = e^{\frac{2\pi i}{3}} = e^{\frac{2\pi i}{3}}$. These reduce to the usual Pauli matrices for $\omega \rightarrow -1$. We note a couple of remarkable properties for these matrices. For any $V, W \in \{X, Y^\dagger, Z\}$

$$\text{Tr}(V) = 0 \quad (7.2a)$$

$$V^2 = V^\dagger \quad (7.2b)$$

$$V^3 = I \quad (7.2c)$$

$$ZX = \omega XZ, \quad XZ = \bar{\omega} ZX \quad (7.2d)$$

$$\omega^2 = \bar{\omega} \quad (7.2e)$$

Note that there are several possible generalisations to more than two dimensions. For example, one could think of the Pauli matrices as generators of $\text{SU}(2)$. Then, the natural generalisation to three dimensions would be given by the *Gell-Mann matrices*. Due to their more complex commutation relations, it seems to be difficult use them to describe qutrit circuits (although generating $\mathfrak{su}(3)$, they are the perfect candidates for a Hamiltonian that gives rise to Gaussian gates in $\text{SU}(3)$). It is conceivable that two basic operators X and Z will not be sufficient to span an interesting class of gates but we could also try to form more complex *combinations* of these operators.

7.2 One-qutrit gates

To gain insights in the required form of the Hamiltonian in this formalism, let us look at the simple case of the 1-qutrit Hamiltonian. We might first attempt it to be quadratic in the X and Y -operators as before. Since X and Y commute up to a phase it would then have three free parameters:

$$H = aXX + bXY + cYY \quad (7.3)$$

$$= \begin{pmatrix} 0 & a + b + c\omega^2 & 0 \\ 0 & 0 & a + b\omega + c\omega \\ a + b\omega^2 + c & 0 & 0 \end{pmatrix} \quad (7.4)$$

and the whole matrix would have to vanish in order to be hermitian. This is because we have seen above that hermitian conjugation of any quadratic operator *doubles* the degree of the monomial and the result is *not a quadratic operator anymore!* We can hence never construct a hermitian Hamiltonian with quadratic terms. By virtue of (7.2c) the maximal

degree of an operator is 4 ($XXZZ$) and we see that in general

$$\begin{aligned}
& \text{linear} \rightarrow \text{quadratic} \\
& \text{quadratic} \rightarrow \text{quartic} \\
& \text{cubic} \rightarrow \text{degree 6} \rightarrow \text{cubic} \\
& \text{quartic} \rightarrow \text{degree 8} \rightarrow \text{degree 5} \rightarrow \text{quadratic}
\end{aligned} \tag{7.5}$$

This leads us to the conjecture that the appropriate Hamiltonian will be *cubic*. Again we look at the 1-qutrit case first. There are two independent cubic operators: XXZ and ZZX (all other operators can be created from those by permutation up to phase or are equal to the identity). Since $(XXZ)^\dagger = Z^\dagger X^\dagger X^\dagger = ZZXXXX = ZZX$ general hermitian Hamiltonian then looks like

$$\begin{aligned}
H &= aXXZ + \bar{a}ZZX \\
&= \begin{pmatrix} 0 & a\omega & \bar{a} \\ \bar{a}\omega^2 & 0 & a\omega^2 \\ a & \bar{a}\omega & 0 \end{pmatrix}
\end{aligned} \tag{7.6}$$

We note, that we have two real degrees of freedom (in (4.17) we had one real degree of freedom for a single qubit). There is a simple argument that allows us to prove

Lemma 7.1. Any commutator is a linear combination of generators, i.e. for every $V \in \{X, Y^\dagger, Z\}$, $[H, V] = \sum_i \kappa_i W_i$, where $\kappa_i \in \mathbb{C}$ and $W_i \in \{X, Y^\dagger, Z\}$

Proof. The Hamiltonian consists of terms that are cubic in X and Z . Due to (7.2e), every single commutator is proportional to a single quartic operator. Now there are two options: Either one of X or Z appears three times in the expression and the term is then proportional to the remaining operator. Or both appear twice and then the term is proportional to Y^\dagger . \square

We can explicitly calculate the commutators for the 1-qutrit Hamiltonian.

Example 7.2. We compute

$$\begin{aligned}
[H, X] &= a[XXZ, X] + \bar{a}[ZZX, X] \\
&= a(XXZX - XXXZ) + \bar{a}(ZZXX - XZZX) \\
&\stackrel{(7.2e)}{=} a(\omega XXXZ - XXXZ) + \bar{a}(\omega^4 XXZZ - \omega^2 XXZZ) \\
&= a(\omega - 1)Z + \bar{a}(\omega - \omega^2)Y^\dagger
\end{aligned} \tag{7.7}$$

and the other relations follow similarly. We summarize them into the M -matrix such that

$$\begin{aligned} \begin{pmatrix} \dot{X}(t) \\ \dot{Z}(t) \\ \dot{Y}^\dagger(t) \end{pmatrix} &=: \dot{\vec{s}}(t) = i \begin{pmatrix} [H, X(t)](t) \\ [H, Z(t)](t) \\ [H, Y^\dagger(t)] \end{pmatrix} \\ &= i \underbrace{\begin{pmatrix} 0 & a(w-1) & \bar{a}(w-\bar{w}) \\ \frac{a(w-1)}{a(\bar{w}-\omega)} & 0 & a(1-\bar{w}) \\ a(\bar{w}-\omega) & \bar{a}(1-\omega) & 0 \end{pmatrix}}_{=:M} \begin{pmatrix} X(t) \\ Z(t) \\ Y^\dagger(t) \end{pmatrix} \end{aligned} \quad (7.8)$$

so

$$\vec{s}(t) = e^{iMt} \vec{s}(0) \quad (7.9)$$

It is interesting to see how this differs from section (4.2). There, one could cast the equivalent of the M -matrix into a form where it was real and antisymmetric which allowed us to interpret the time evolution through a gate by a rotation in a $2n$ -dimensional subspace. For the case $n = 1$, one free parameter would span all of $SO(2)$. In $SO(3)$, there are not two, but *three* (Euler) angles to specify but here we are only spanning a 2-dimensional space. Combining this with the fact that Y^\dagger is so dissimilar from X and Y , we come to the conclusion that something crucial is missing in this theory. Is there an operator that would allow for a more general Hamiltonian but still allow to express time evolution in a linear subspace? Another minor issue is that the iM -matrix is antihermitian as anticipated but not real.

7.3 Further questions and outlook

We discovered that there was an unaesthetic asymmetry between the X, Z and Y^\dagger -operators. That was not the case in two dimensions. Furthermore, we could not span all of $SO(3n)$ with the proposed M -matrix.

Further research in this area could be concerned with the following questions: Is there a more natural set of operators to construct a qutrit Hamiltonian? How would a simulable 2-qutrit gate look like? Which insights can one obtain by trying to generalize the notion of parity to 2-qutrit states (i.e. $x_1 + x_2 \bmod 3$)? How does the nearest neighbour constraint arise in this formalism?

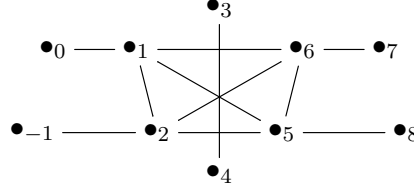
References

- [1] Jozsa, R., Quantum Computation, Lecture Notes (2014)
- [2] Valiant, L., Quantum Computers that can be Simulated Classically in Polynomial Time, *Proceeding STOC '01 Proceedings of the thirty-third annual ACM symposium on Theory of computing* pp.114 - 123 (2001)
- [3] Jozsa, R. and Miyake, A., Matchgates and classical simulation of quantum circuits, *Proc. R. Soc. A* 8 pp. 2089-3106 (2008)
- [4] Terhal, B. and DiVincenzo, D., Classical simulation of noninteracting-fermion quantum circuits, *Phys. Rev. A* 65 (2002)
- [5] Cubitt, T. Advanced Quantum Information Theory, Lecture Notes (2014)
- [6] Arora, S. and Barak, B., Computational Complexity - A Modern Approach, *Cambridge University Press* (2009)
- [7] Aharonov, D. and Naveh, T., Quantum NP - A Survey, *arXiv:quant-ph/0210077* (2002)
- [8] Cayley, A., Sur les déterminantes gauches, *Crelle's J.* 38 p. 93 (1854)
- [9] Kasteleyn, P., The statistics of dimers on a lattice, *Physica* 27, pp.1209-1225 (1961)
- [10] Temperley, H. and Fisher, M., M. E. dimer problems in statistical mechanics n exact result, *Philosophical Magazine* 6 pp.1061-1063 (1961)
- [11] Jerrum, M., Counting, Sampling and Integrating: Algorithms and Complexity, Birkhauser, Basel, Switzerland (2003)
- [12] Strassen, V., Gaussian elimination is not optimal, *Numerische Mathematik* Volume 13, Issue 4, pp. 354-356, 1969
- [13] Cai, J. and Gorenstein, A., Matchgates Revisited, *arXiv:1303.6729 [cs.CC]* (2013)
- [14] Wimmer, M., Efficient numerical computation of the Pfaffian for dense and banded skew-symmetric matrices, *ACM Trans. Math. Software.* 38 p. 30 (2012)
- [15] Gottesman, D., The Heisenberg Representation of Quantum Computers, *Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics* pp. 32-43 (1998).
- [16] Jordan, P. and Wigner, E. Über das Paulische Äquivalenzverbot, *Zeitschrift für Physik* 47, p. 631 (1928).
- [17] Jozsa, R. et al., Matchgate and space-bounded quantum computations are equivalent, *Proc. R. Soc. A* 466 pp. 809-830 (2010)

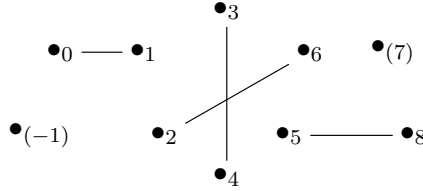
- [18] Brod, D and Galvao, E., Extending matchgates into universal quantum computation, *Phys. Rev. A* 84 (2011)
- [19] Neeley, M. et al., Emulation of a Quantum Spin with a Superconducting Phase Qudit, *Science* vol. 325 no 5941, pp. 722-725 (2009)
- [20] Hoyer, P., Efficient Quantum Transforms, *arXiv:quant-ph/9702028* (1997)
- [21] Gottesman, D., Fault-Tolerant Quantum Computation with Higher-Dimensional Systems, *Lecture Notes in Computer Science* Volume 1509 pp. 302-313 (1999)
- [22] Clark, S., Valence bond solid formalism for d-level one-way quantum computation *J. Phys. A: Math. Gen* 39, pp. 2701-2721 (2006)
- [23] Brennen, G. et al. Efficient Circuits for Exact-Universal Computations with Qudits, *Quantum Information and Computation* Vol. 6 p. 436 (2006)

A Modifiers

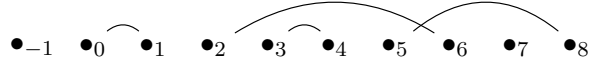
The purpose of this section is to prove that only for the $\chi(\Gamma, \{1, 5\})$ and $\chi(\Gamma, \{2, 6\})$ terms a negative modifier arises. Indeed consider the externally connected 2-qubit gate



We take vertices out to ensure that they are saturated *by external edges*. So in the example of $\chi(\Gamma, \{1, 5\})$, 1 is surely connected to 0 and 5 to 8. The only possible perfect matching will be



which indeed - counting the overlaps -



has a modifier of -1 . Same holds symmetrically for $\chi(\Gamma, \{2, 6\})$. From the above graph it is also clear that if none or all vertices corresponding to one qubit line are retained or removed no odd overlap will arise. Similarly, for 1-qubit gates with single input and output vertices, the modifier is $+1$. This explains the minus signs in the derivation of theorem 3.16.

B Calculation of the interaction commutator

The commutator for the interacting term is

$$\begin{aligned}
[H_I, a_k(0)] &= \sum_{ghij} V_{ghij} \left[a_g^\dagger (a_h^\dagger a_i a_j), a_k \right] \\
&= \sum_{ghij} V_{ghij} \left(a_g^\dagger \left[a_h^\dagger (a_i a_j), a_k \right] + \underbrace{\left[a_g^\dagger, a_k \right]}_{=\delta_{gk} - 2a_k a_g^\dagger} a_h^\dagger a_i a_j \right) \\
&= \sum_{ghij} V_{ghij} \left(a_g^\dagger \left(a_h^\dagger [a_i a_j, a_k] + \underbrace{\left[a_h^\dagger, a_k \right]}_{\delta_{hk} - 2a_k a_h^\dagger} a_i a_j \right) - 2a_k a_g^\dagger a_h^\dagger a_i a_j \right) \\
&\quad + \sum_{hij} V_{khij} a_h^\dagger a_i a_j \\
&= \sum_{ghij} V_{ghij} \left(a_g^\dagger a_h^\dagger \left(a_i \underbrace{[a_j, a_k]}_{2a_j a_k} + \underbrace{[a_i, a_k]}_{2a_i a_k} a_j \right) - 2a_g^\dagger a_k a_h^\dagger a_i a_j \right) \\
&\quad + \sum_{gij} V_{gkij} a_g^\dagger a_i a_j + \sum_{hij} V_{khij} a_h^\dagger a_i a_j \\
&= 2 \sum_{ghij} V_{ghij} \left(a_g^\dagger a_h^\dagger a_i \underbrace{(a_j a_k + a_k a_j)}_{\{a_j, a_k\}=0} - a_g^\dagger a_k a_h^\dagger a_i a_j \right) \\
&\quad + \sum_{gij} V_{gkij} a_g^\dagger a_i a_j + \sum_{hij} V_{khij} a_h^\dagger a_i a_j \\
&= 2 \sum_{ghij} V_{ghij} a_g^\dagger a_k a_h^\dagger a_i a_j + \sum_{gij} V_{gkij} a_g^\dagger a_i a_j + \sum_{hij} V_{khij} a_h^\dagger a_i a_j
\end{aligned}$$

as desired.

C Planarity

The efficient computability of the Pfaffian of a graph requires is *planarity* i.e. that one can draw the graph on a piece of paper without any vertices crossing. Consider the following example of a complete 4-graph, acting on the second qubit line of a circuit of three qubits. We see that for a complete 4-graph, planarity is always given even when embedded on arbitrary positions in a matchcircuit. Consider further the 2-qubit gates from section 3.

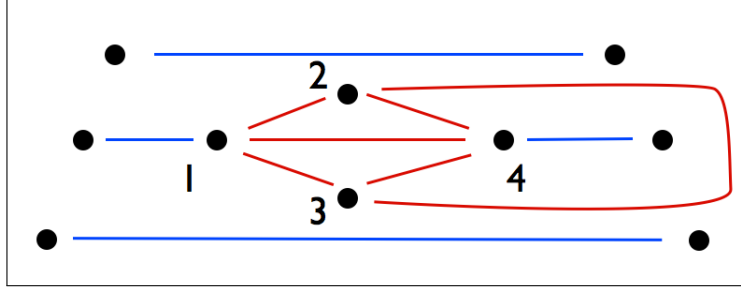


Figure 3: A complete graph with 4 vertices acting on the intermediate qubit of a 3 qubit matchcircuit. The labeling is taken from theorem 3.15. The internal edges are coloured in red whereas external and linking edges are blue.

The complete 6-graph is not planar, but the gates we considered were not complete in the sense that vertices 3 and 4 did only have an edge connecting them with each other but no connection to the rest of the graph. We see that the matchcircuits one can construct by letting 2-qubit gates act on arbitrary qubit lines will be planar.

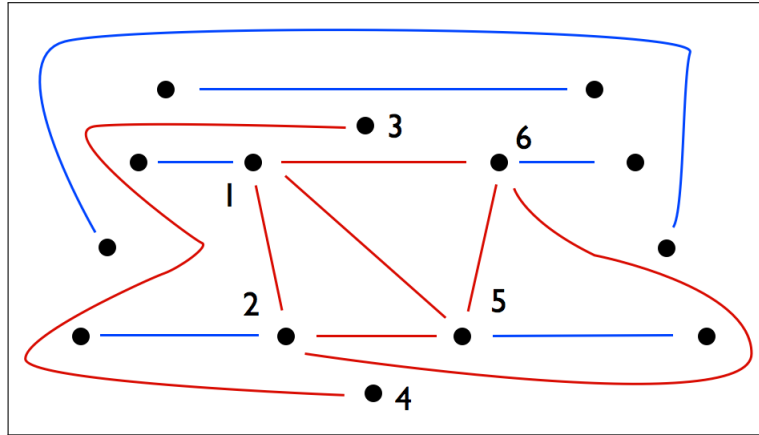


Figure 4: A graph with 6 vertices acting on qubits 2 and 4 of a 4 qubit matchcircuit. The labeling is taken from theorem 3.16. The internal edges are coloured in red whereas external and linking edges are blue.

D Source File for the Matchgate Simulator

The output in section 3.4 is generated by a plain text file with the following numbers all in separate lines:

```
2111000101000201000001010000010
```

```
color
The code is
```

```

1 #MATCHGATESIMULATOR.PY
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon Mar 24 22:22:44 2014
5 @author: henrikdreyer
6 """
7 import numpy as np
8 # We use the module in [14]
9 import pfaffian as pf
10
11 #The idea is to add the verices and edges one-by-one
12 #while scanning through the file:
13
14 def add_vertex(atX,atY,number):
15     #Write the new vertex into the LaTeX output
16     global f
17     f.write("!{")
18     f.write(str(atX))
19     f.write(",")
20     f.write(str(atY))
21     f.write(" }*+{\\bullet-{" + str(number)+\
22     "}}=\\ " + str(number) + "\\n")
23
24 def add_edge(from_vertex,to_vertex,value,\
25             over_or_under=1):
26     #Write the the new edge into the LaTeX output
27     global f
28     if over_or_under==1:
29         f.write("\\")
30         f.write(str(int(from_vertex)))
31         f.write("\\-\\")
32         f.write(str(int(to_vertex)))
33         f.write("\\ ^{")
34         f.write(str(value))
35         f.write("}")
36         f.write("\\n")
37     else:
38         f.write("\\")
39         f.write(str(int(from_vertex)))
40         f.write("\\-\\")
41         f.write(str(int(to_vertex)))
42         f.write("\\ _{")
43         f.write(str(value))
44         f.write("}")
45         f.write("\\n")
46
47 def add_one_qubit_gate(acts_on,a,b,c,d,final_gate):
48     #For a 1-qubit gate:
49     #This adds both edges and vertices to the
50     #LaTeX output and fills the upper part
51     #of the skew-symmetric graph matrix
52     #at the same time
53

```

```

55     global p,q,A,T,g,k,X,Y
56
57     #Add initial vertex and edge to both
58     #graph matrix and LaTeX output
59     A[q[acts_on],k+1] = 1
60     add_vertex(X+1,n-1-acts_on,k+1)
61     add_edge(q[acts_on],k+1,1,1)
62
63     #Move geometric position
64     X=X+1
65     Y=n-1-acts_on
66
67     #Add the omissible vertex to the set of all
68     #omissible vertices
69
70     #Add internal vertex and edge to both
71     #graph matrix and LaTeX output
72     T=np.append(T,[k+2])
73     A[k+1,k+2] = a*np.conjugate(c)/abs(c)**2
74     #We assume c neq 0 for now (cf. main text)
75     A[k+1,k+3] = b
76     A[k+2,k+3] = d
77     A[k+3,k+4] = c
78     add_vertex(X+0.5,n-1-acts_on+0.5,k+2)
79     add_vertex(X+0.5,n-1-acts_on-0.5,k+3)
80     add_vertex(X+1.0,n-1-acts_on,k+4)
81     add_edge(k+1,k+2,a/c,1)
82     add_edge(k+1,k+3,b,0)
83     add_edge(k+2,k+3,d,1)
84     add_edge(k+3,k+4,c,0)
85
86     #Update position and current "active" qubit
87     k=k+4
88     X=X+2
89     q[acts_on] = k
90
91     #Generate external vertices and edges
92     for i in range(0,n):
93         p[i] = q[i]
94         q[i] = k+i+1
95         A[p[i],q[i]]=1
96         add_vertex(X+dist*i,n-1-i,k+i+1)
97         add_edge(p[i],q[i],1,1)
98
99     #Update positions
100     k=k+n
101     X=X+dist*(n-1)+1
102
103     #If this is the final gate,
104     #add a layer of output vertices
105     if final_gate==0:
106         for i in range(n-1,-1,-1):
107             p[i] = q[i]

```

```

107         q[i] = k+1+n-1-i
108         add_vertex(X+dist*(n-1-i), n-1-i, k+1+n-1-i)
109         add_edge(p[i], q[i], 1, 1)
110         A[p[i], q[i]] = 1
111         k=k+1+n-1
112         X=X+dist*(n-1)
113
114 def add_two_qubit_gate(\
115     acts_on, a, b, c, d, e, f, g, h, final_gate):
116     #For a 2-qubit gate:
117     #This adds both edges and
118     #vertices to the LaTeX output
119     #and fills the upper
120     global p, q, A, T, k, X, Y
121
122     #Add initial vertex and edge to both
123     #graph matrix and LaTeX output
124     A[q[acts_on], k+1] = 1
125     A[q[acts_on+1], k+2] = 1
126     add_vertex(X+1, n-1-acts_on, k+1)
127     add_vertex(X+1+dist, n-1-(acts_on+1), k+2)
128     add_edge(q[acts_on], k+1, 1, 1)
129     add_edge(q[acts_on+1], k+2, 1, 1)
130
131     #Move geometric position
132     X=X+1+dist
133
134     #Add internal vertex and edge to both
135     #graph matrix and LaTeX output
136     t = np.conjugate(d)/abs(d)**2
137     A[k+1, k+2] = t*b #assume c neq 0 for now
138     A[k+1, k+5] = -t*g
139     A[k+1, k+6] = t*h
140     A[k+2, k+5] = t*e
141     A[k+2, k+6] = -t*f
142     A[k+3, k+4] = d
143     A[k+5, k+6] = t*c
144
145     add_vertex(X+0.5, n-1-acts_on+0.5, k+3)
146     add_vertex(X+0.5, n-1-(acts_on+1)-0.5, k+4)
147     add_vertex(X+1.0, n-1-(acts_on+1), k+5)
148     add_vertex(X+1.0+dist, n-1-acts_on, k+6)
149
150     add_edge(k+1, k+2, t*b, 1)
151     add_edge(k+1, k+5, -t*g, 0)
152     add_edge(k+1, k+6, t*h, 0)
153     add_edge(k+2, k+5, t*e, 0)
154     add_edge(k+2, k+6, -t*f, 0)
155     add_edge(k+3, k+4, 1./t, 0)
156     add_edge(k+5, k+6, t*c, 0)
157
158     #Update position and current "active" qubit
159     k=k+6

```

```

161     X=X+2+dist
162     q[acts_on] = k
163     q[acts_on+1] = k-1
164
165     #Generate external vertices and edges
166     for i in range(0,n):
167         p[i] = q[i]
168         q[i] = k+i+1
169         A[p[i],q[i]]=1
170         add_vertex(X+dist*i,n-1-i,k+i+1)
171         add_edge(p[i],q[i],1,1)
172
173     #Update positions
174     k=k+n
175     X=X+dist*(n-1)+1
176
177     #If this is the final gate,
178     #add a layer of output vertices
179     if final_gate==0:
180         for i in range(n-1,-1,-1):
181             p[i] = q[i]
182             q[i] = k+1+n-1-i
183             add_vertex(X+dist*(n-1-i),n-1-i,k+1+n-1-i)
184             add_edge(p[i],q[i],1,1)
185             A[p[i],q[i]] = 1
186             k=k+1+n-1
187             X=X+dist*(n-1)
188
189 def compute_pfaffian_sum(B,T):
190     #Here we compute the PfaffianSum effectively ,
191     #following Valiant's original paper on
192     #matchgates [7]. In particular, we use
193     #use the fact that the PfaffianSum of
194     #a matrix is the Pfaffian of a modified matrix.
195     #This allows us to compute the Pfaffian
196     #without needing to consider an exponential
197     #number of terms.
198
199     #Create modified matrix for both even and odd graphs
200     #and return Pfaffian
201     sizeB=np.shape(B)[0]
202     if sizeB % 2 == 0:
203         Lambda=np.zeros((sizeB,sizeB))
204         l=np.zeros(sizeB)
205         for i in T:
206             l[i]=1
207         for i in range(sizeB):
208             for j in range(sizeB):
209                 if i<j:
210                     Lambda[i,j]=(-1)**(j-i+1)*l[i]*l[j]
211                 elif i>j:
212                     Lambda[i,j]=(-1)**(i-j)*l[i]*l[j]

```



```

213     return pf.pfaffian(B+Lambda)
214 elif sizeB % 2 != 0:
215     temp = np.vstack((B,np.zeros(sizeB)))
216     B_plus = np.column_stack((temp,np.zeros\
217                               (sizeB+1)))
218     Lambda_plus = np.zeros((sizeB+1,sizeB+1))
219     l=np.zeros(sizeB+1)
220     l[sizeB]=1
221     for i in T:
222         l[i]=1
223     for i in range(sizeB+1):
224         for j in range(sizeB+1):
225             if i<j:
226                 Lambda_plus[i,j]=(-1)**(j-i+1)\
227                                     *l[i]*l[j]
228             elif i>j:
229                 Lambda_plus[i,j]=(-1)**(i-j)\
230                                     *l[i]*l[j]
231     return pf.pfaffian(B_plus+Lambda_plus)
232
233 def output(A,input_state , output_state , omissables):
234     #Remove the vertices corresponding to 0 in input
235     #and output and calculate the according PfaffianSum
236     #and take square of modulus.
237     B=A
238     f.write("The probability to obtain $\\ket{")
239     for i in range(n):
240         f.write(str(output_state[i]))
241     f.write("}$ from $\\ket{")
242     for i in range(n):
243         f.write(str(input_state[i]))
244     f.write("}$ is ")
245     omit = omissables
246
247     for i in range(n-1,-1,-1):
248         if output_state[i]==0:
249             B=np.delete(B,V-(n-i),0)
250             B=np.delete(B,V-(n-i),1)
251     for i in range(0,n):
252         if input_state[i]==0:
253             B=np.delete(B,n-i-1,0)
254             B=np.delete(B,n-i-1,1)
255     omit=omit-1
256     PfS= compute_pfaffian_sum(B,omit)
257     f.write(str(abs(PfS)**2))
258     print "The probability to obtain ", \
259           str(output_state), " from ", \
260           str(input_state), " is ", \
261           str(abs(PfS)**2)
262     f.write(". \\\\ \\n")
263
264 if __name__ == "__main__":

```

```

267 #Initialize LateX-file
inputfilepath = raw_input("Specify file: ")
269 print "Looking through ", inputfilepath
inputfile = open(str(inputfilepath))

271 #Initialize output and input vertices and
#assign global variables
273 f=open('output.tex','w')
n = int(inputfile.readline())
275 O = int(inputfile.readline())
Two = int(inputfile.readline())
277 number_of_vertices = n + O*(2*n+4) + Two*(2*n+6) - n
V = number_of_vertices
279 k = 0 #last vertex counter
q = np.zeros(n) #last vertex on line i
281 p = np.zeros(n) #last-last vertex on line i
T = np.zeros(0) #omittable nodes
283 A = np.zeros((V,V),'complex') #Graph matrix
X = 0 #store latest X-position
285 Y = 0 #store latest Y-position
dist = 1./n #shift between parallel vertices in slice
287

#Write initial clause into LateX file
289 f.write("\\documentclass[a4paper,12pt,landscape]")
f.write("{article} \n ")
291 f.write(\\
"\usepackage{array} \n \usepackage{braket} \n ")
293 f.write(\\
"\usepackage[pass]{geometry} \n ")
295 f.write(\\
"\usepackage[landscape]{geometry} \n")
297 f.write(\\
"\usepackage{amsmath} \n \\\pagestyle{empty} ")
299 f.write(\\
"\usepackage[all]{xy} \n \\\input{Qcircuit} \n ")
301 f.write(\\
"\\\begin{document} \n")
303 f.write("\\\\begin{align*} \n")
f.write("\\\\xygraph{")
305 f.write("!{<0cm,0cm>;<")
f.write(str(4./(O+Two)))
307 f.write("cm,0cm>;<0cm,")
f.write(str(6./n))
309 f.write("cm>::} \n")

311 #Add input vertices
for i in range(n-1,-1,-1):
313     q[i] = n-1-i
    add.vertex(dist*(n-1-i),n-1-i,n-i-1 )
315 k = n-1
X = dist*(n-1)
317 Y = n-1

```

```

319 #Loop through all gates, read the next bit in the
321 #source file. If it is 1, add a one qubit gate on
323 #the line that is specified thereafter with the
325 #values that are specified thereafter.
327 #If it is 2, add a two qubit gate on the line
329 #that is specified thereafter with the
331 #values that are specified thereafter.
333 for i in range(O+Two):
335     final_gate=0
337     if i==O+Two-1:
339         final_gate=1
341     which_gate=int(inputfile.readline())
343     if which_gate==1:
345         qubit_line=int(inputfile.readline())
347         U_11 = complex(float(inputfile.readline()),\
349             float(inputfile.readline()))
351         U_12 = complex(float(inputfile.readline()),\
353             float(inputfile.readline()))
355         U_21 = complex(float(inputfile.readline()),\
357             float(inputfile.readline()))
359         U_22 = complex(float(inputfile.readline()),\
361             float(inputfile.readline()))
363         add_one_qubit_gate\
            (qubit_line ,U_11,U_12,U_21,U_22,final_gate)
365     elif which_gate==2:
367         qubit_line=int(inputfile.readline())
369         U1_11 = complex(float(inputfile.readline()),\
371             float(inputfile.readline()))
373         U1_12 = complex(float(inputfile.readline()),\
375             float(inputfile.readline()))
377         U1_21 = complex(float(inputfile.readline()),\
379             float(inputfile.readline()))
381         U1_22 = complex(float(inputfile.readline()),\
383             float(inputfile.readline()))
385         U2_11 = complex(float(inputfile.readline()),\
387             float(inputfile.readline()))
389         U2_12 = complex(float(inputfile.readline()),\
391             float(inputfile.readline()))
393         U2_21 = complex(float(inputfile.readline()),\
395             float(inputfile.readline()))
397         U2_22 = complex(float(inputfile.readline()),\
399             float(inputfile.readline()))
401         add_two_qubit_gate\
            (qubit_line ,U1_11,U1_12,U1_21,\
            U1_22,U2_11,U2_12,U2_21,U2_22,final_gate)
403
405 #Antisymmetrize
407 for i in range(V):
409     for j in range(V):
411         if j>i:
413             A[j,i] = - A[i,j]
415
417 #Write graph matrix

```

```

373     f.write("} \n \\end{align*} \n")
374     f.write("\\newpage \n")
375     f.write("The graph matrix is: i \n")
376     f.write("\\begin{align*} \n")
377     f.write("\\left( \n")
378     f.write(" \\setlength{\\extrarowheight}{-6} \n")
379     f.write("\\begin{array}")
380     f.write("{")
381     for i in range(V+1):
382         f.write("c")
383     f.write("} \n")
384     for i in range(V):
385         f.write("& ")
386         f.write("\\mathbf{")
387         f.write(str(i))
388         f.write("}")
389     f.write("\\\\")
390     for i in range(V):
391         f.write("\\mathbf{")
392         f.write(str(i))
393         f.write("} & ")
394         for j in range(V):
395             if A[i,j] != 0:
396                 f.write(str(A[i,j]))
397                 f.write(" & ")
398             else:
399                 f.write("0")
400                 f.write(" & ")
401         f.write("\\\\")
402     f.write("\\end{array} \\right) \n")
403     f.write("\\end{align*} \n")
404
405     #Prompt user to input desired transistion probability
406     print "Specity ", str(n), "-qubit input and output"
407     inputstate_raw = raw_input\
408     ("input state (seperate qubits by comma(,)) ?")
409     outputstate_raw = raw_input\
410     ("output state (seperate qubits by comma(,))?" )
411     output(A, map(int, inputstate_raw.split(',')), \
412            map(int, outputstate_raw.split(',')), T)
413     print "Please see output.tex for the results."
414
415     f.write("\\end{document}")
416     f.close()

```

MatchgateSimulator.py

```

1  #CREATCIRCUIT.PY
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Mar 28 18:17:55 2014
5  @author: henrikdreyer
6  """

```

```

7 | if __name__ == "__main__":
   |     #This routine walks the user
   |     #through the creation of a
   |     #quantum circuit in the terminal.
11 |     #The output file will be in
   |     #a format that can be used
13 |     #by MatchCircuitSimulator.py
   |     #Alternatively, the circuit
15 |     #can be algorithmically generated.
   |     f=open('circuit','w')
17 |
   |     #Global variables at
19 |     #the beginning:
   |     #Number of qudits,
21 |     #number of 1-qubit gates
   |     #number of 2-qubit gates
23 |     number_of_qudits = raw_input("Number of qubits? ")
   |     f.write(number_of_qudits)
25 |     f.write("\n")
   |     number_of_one_qubit_gates = raw_input\
27 |         ("Number of 1-qubit gates? ")
   |     f.write(number_of_one_qubit_gates)
29 |     f.write("\n")
   |     number_of_two_qubit_gates = raw_input\
31 |         ("Number of 2-qubit gates? ")
   |     f.write(number_of_two_qubit_gates)
33 |     f.write("\n")
35 |
   |     #For each gate
   |     #1) specify 1 or 2-qubit gate
37 |     #2) specify target 1st qubit line (n.n.)
   |     #3) specify matrix elements of gate
39 |     for i in range(int(number_of_one_qubit_gates)\
   | +int(number_of_two_qubit_gates)):
41 |         one_or_two = raw_input\
   |             ("1 or 2-qubit gate (type 1 or 2)? ")
43 |         f.write(one_or_two)
   |         f.write("\n")
45 |         acts_on = raw_input\
   |             ("First qubit line it acts on (first one is 0) ")
47 |         f.write(acts_on)
   |         f.write("\n")
49 |         if one_or_two == "1":
   |             f.write(raw_input("Real part of U11? "))
51 |             f.write("\n")
   |             f.write(raw_input("Imaginary part of U11? "))
53 |             f.write("\n")
55 |
   |             f.write(raw_input("Real part of U12? "))
   |             f.write("\n")
57 |             f.write(raw_input("Imaginary part of U12? "))
   |             f.write("\n")
59 |

```

```

61     f.write(raw_input("Real part of U21? "))
62     f.write("\n")
63     f.write(raw_input("Imaginary part of U21? "))
64     f.write("\n")
65
66     f.write(raw_input("Real part of U22? "))
67     f.write("\n")
68     f.write(raw_input("Imaginary part of U22? "))
69     f.write("\n")
70
71     elif one_or_two == "2":
72         f.write(raw_input("Real part of U1.11? "))
73         f.write("\n")
74         f.write\
75             (raw_input("Imaginary part of U1.11? "))
76         f.write("\n")
77
78         f.write(raw_input("Real part of U1.12? "))
79         f.write("\n")
80         f.write\
81             (raw_input("Imaginary part of U1.12? "))
82         f.write("\n")
83
84         f.write(raw_input("Real part of U1.21? "))
85         f.write("\n")
86         f.write\
87             (raw_input("Imaginary part of U1.21? "))
88         f.write("\n")
89
90         f.write(raw_input("Real part of U1.22? "))
91         f.write("\n")
92         f.write\
93             (raw_input("Imaginary part of U1.22? "))
94         f.write("\n")
95
96         f.write(raw_input("Real part of U2.11? "))
97         f.write("\n")
98         f.write\
99             (raw_input("Imaginary part of U2.11? "))
100        f.write("\n")
101
102        f.write(raw_input("Real part of U2.12? "))
103        f.write("\n")
104        f.write\
105            (raw_input("Imaginary part of U2.12? "))
106        f.write("\n")
107
108        f.write(raw_input("Real part of U2.21? "))
109        f.write("\n")
110        f.write\
111            (raw_input("Imaginary part of U2.21? "))
112        f.write("\n")

```

```
113         f.write(raw_input("Real part of U2.22? "))
114         f.write("\n")
115         f.write\
116             (raw_input("Imaginary part of U2.22? "))
117         f.write("\n")
118
119     print "Done. Output written in \"circuit\"."
```

CreateCircuit.py

E Declaration

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

.....