Università degli studi di Pisa

Corso di Laurea Specialistica in Informatica

On some combinatorial properties of graph states

Tesi di

Alessandro Cosentino

Febbraio 2009

Relatori:

Anna Bernasconi

Simone Severini

Sommario

I *graph state* sono particolari stati quantistici, rappresentabili tramite grafi indiretti semplici, che giocano un ruolo fondamentale in informatica quantistica, in particolare nell'ambito dei codici a correzione di errore e nel modello di computazione "one-way". Lo scopo di questa tesi è studiare alcune proprietà dei graph state attraverso un approccio combinatorio. Innanzitutto si sono analizzate le proprietà di un'invariante dei grafi: il numero di sottografi indotti con numero dispari di archi. Questo numero è stato valutato per famiglie importanti di grafi ed è stato trovato un algoritmo efficiente per calcolarlo. Alcune proprietà delle funzioni booleane associate ai graph state sono state studiate analizzando la struttura dei grafi di Cayley delle suddette funzioni. A tale scopo sono stati definiti, e ne è stata analizzata la struttura, grafi che permettono di tracciare le trasformazioni di *local complementation* e *switching* fra graph state. Un congettura è stata fatta sulla struttura di questi grafi. A margine del lavoro, sono state introdotte alcune estensioni alla definizione classica di graph state. In particolare sono stati definiti gli "edge graph state" e i "3-hypergraph state".

Abstract

Graph states are particular quantum states that can be represented by mathematical graphs and that play a fundamental role in quantum information science. In particular, they are important in "one-way" computation model and as codewords in quantum error correction. The aim of this thesis is to investigate some properties of graph states by a combinatorial approach. In order to classify graph states, we study an invariant of graphs: the number of induced subgraphs with odd number of edges. We describe an efficient algorithm to calculate this number and we prove formulas to calculate it for familiar classes of graphs. We also investigate some properties of the boolean functions corresponding to graph states by analizing the structure of Cayley graphs that can be associated to these functions. It is important to understand how the amount of entanglement in graph states changes with regards to graph transformations. In order to approach this problem, we introduce some graphs whose paths keep tracks of the transformations of *local complementation* and *switching* between graph states. Finally, two extensions of the classical definition of graph states are introduced. In particular we define "edge graph states" and "3-hypergraph state".

I would like to address my special acknowledgments to my supervisors Anna Bernasconi and Simone Severini for their guidance, their ideas and their encouragement. I am grateful to my cousin Giuseppe Policastro for our Xmas conversations about physics and computer science. Keith Briggs (BT Research) helped with his ideas about LCS-graphs. From Marc Thurley (Humboldt-Universität zu Berlin) I have got the right answer at the right time. Special thanks also to Matthew Parker, Chris Godsil, Gordon Royle, Niel de Beaudrap, Zhengfeng Ji and everyone who bothered replying to my annoying emails.

Contents

1 Introduction **1**
1.1 Overview . . . 3
1.2 Notation . . . 4

2 Preliminaries **5**
2.1 Quantum information processing . . . 5
2.1.1 Basic concepts of quantum computing . . . 5
2.1.2 Stabilizer formalism . . . 9
2.1.3 Fast quantum algorithms . . . 10
2.1.4 Quantum computational complexity . . . 11
2.1.5 Quantum error-correction codes . . . 12
2.2 Graph Theory . . . 13
2.2.1 Notation and terminology . . . 13
2.2.2 Familiar classes of graphs . . . 14
2.2.3 Hypergraphs . . . 14

3 Graph States **15**
3.1 Definitions . . . 15
3.2 Local unitary and local Clifford equivalence . . . 18
3.3 One-way quantum computing . . . 20

4 The *MS-number* **23**
4.1 Definitions . . . 23

4.2 An explicit formula . . . 24
4.3 Algorithm . . . 26
4.4 Binary rank . . . 27
4.5 Properties of the MS-number . . . 28
4.6 MS-number for familiar classes of graphs . . . 28
4.7 MS-number and graph isomorphism . . . 30
4.8 MS-number as a partition function . . . 32

5 Cubelike graphs **35**
5.1 Definition . . . 35
5.2 Isomorphism . . . 36
5.3 Connectedness . . . 38
5.4 Eingesystem . . . 39

6 Local-Complementation-and-Switching Graph **41**
6.1 Introduction . . . 41
6.2 Switching . . . 41
6.3 LCS graphs . . . 42
6.4 Graph G3 and G4 . . . 43
6.5 The conjecture . . . 45

7 Open Problems **47**

A Extensions **49**
A.1 Edge graph states . . . 49
A.2 3-hypergraph states . . . 50

B Code **53**

List of Figures

2.1 Example of quantum circuit 9

3.1 The complete graph K_3 17
3.2 Quantum circuit for the preparation of $|K_3\rangle$ 17
3.3 Example of local complementation 19
3.4 LU-LC counterexample. 20

4.1 C_4 and S_5 30
4.2 $m(P_3) = m(\text{co-}P_3) = 4$, but $P_3 \ncong \text{co-}P_3$ 31
4.3 $m(G) = m(H) \wedge |E(G)| = |E(H)|$, but $G \ncong H$ 31

5.1 The Cayley graph $X(\mathbb{F}_2^n, \Omega_{K_3})$. 36
5.2 $G \ncong H$, but $|G\rangle \sim_{Cayley} |H\rangle$ 38

6.1 G^σ is a switch of G 42
6.2 The LCS-graph G3 43
6.3 The LCS-graph G4 44

A.1 An example of 3-hypergraph. 50
A.2 Quantum circuit for the preparation of $|H\rangle$ 51

List of Tables

4.1 Number of equivalence classes of graphs for $n = 1 \ldots 9$ on ms-number. 31
4.2 Number of equivalence classes of graphs for $n = 1 \ldots 9$ on (ms-number, size). . 32
4.3 Equivalence classes of graphs based on (order, size, MS-number). 33

1 Introduction

The aim of this thesis is to investigate some properties of *graph states*. These are particular *quantum states* that can be represented by *simple undirected graphs*.

Quantum computation and quantum information theory are fields involved with the study of information processing tasks that can be accomplished using quantum mechanical systems. Quantum computers were first proposed in 1981 by the Nobel laureate physicist Richard Feynman. He showed how a quantum system could be used to do computations with the initial intent of simulating experiments in quantum physics [Fey81].

The theoretical research developed interesting models of quantum computation and surprising algorithms, such as Peter Shor's factoring algorithm (1994) [Sho97] or Lov Grover's search algorithm (1996) [Gro97]. They exploit features of quantum systems to speed up tasks, like factoring an n-digit number and locating an entry in a database of n entries, respectively. These two tasks can be accomplished by classical computers exponentially and quadratically slower.

The potential of these algorithms stimulated many research groups around the world to work in the direction of realizing a physical implementation of the quantum computer. In 1998, IBM scientists led by Isaac Chuang implemented the first 2-qubit quantum computer and in 2001 (see [IR01]) the same team built a seven-qubit quantum computer to run Shor's Algorithm and they correctly identified 3 and 5 as the factors of 15. Clearly these are still toy-models, but still useful to uncover properties of quantum evolution. Experiments are now aimed at developing quantum computers that can easily "scale" to a large number of qubits. Unfortunately, interactions between a quantum system and its environment, a phenomenon called *decoherence*, make useful quantum computation still impossible. A theoretical approach to protect quantum computation from the effects of decoherence is quantum error-correcting coding, the quantum analog of error correction coding fundamental for the transmission of

classical information over noisy channel. In 1995 Shor proved that quantum error-correcting codes exist and may indeed be the solution to reduce the rate of decoherence in quantum memory ([Sho95]). The fact that interactions with a quantum system perturbate the system has been exploited in the design of *quantum cryptography protocols*, where the two parties are able to detect the presence of an eavesdropper. Prototypes for doing *quantum cryptography* have been already used in some real-world applications.

Among the most interesting applications of quantum information processing, a key role is played by distinctively quantum mechanical phenomena, such as *superposition* and *entanglement*. For example, entanglement is responsible for the security of quantum key distribution and for the speedup of a quantum computer compared to classical computers. Entangled states are also used as keywords in quantum secret-sharing protocols [HHHH07].

Graph states form an interesting class of multipartite entangled states, which play important roles in areas as diverse as quantum error correction, one-way computation model and cryptographic protocols. The one-way model is a fairly recent computational model in which computation is driven by measurements instead of unitary evolution as it is done in the standard set-up of quantum computation.

Graph states can be analyzed from different points of views, because they can be represented as different objects: unit vectors in Hilbert space, simple undirected graphs, quadratic Boolean forms, codewords of stabilizer codes. Graph states are important because they are relatively easy to implement in laboratory. Additionally, it appears that graph states allow the scaling of many qubit, therefore being particularly promising as tools in quantum information processing, the realization of various protocols and, in general, the construction of nanotechnology devices. The discovery of new quantum algorithms, that can speed-up tasks over all known classical algorithms, and the discovery of efficient quantum error-correcting codes, that can fight decoherence, are the most promising theoretical approaches that could lead quantum computers to become usable in everyday life. Many hopes are set on the power of graph states for the achievement of these objectives. The key feature of graph states is the presence of multipartite entanglement in them. So far, the typical approach of studying the entanglement in graph states is investigating their local equivalences, that is determining equivalence classes of graph states under *local operations*. Even if the main conjecture on the problem of classifying graph states under local operations (*LU-LC conjecture*) has been proved [JCWY07], many questions are still open.

In this thesis we seek new directions in classifying graph states. We think that the relation between homogeneous boolean polynomials and graph states has not been investigated

thoroughly. For example, the weight of boolean functions representing certain graphs could bound the geometric measure of entanglement of the associated graph states . We give an efficient algorithm to calculate this number for a generic graph. A closed formula is given only for important classes of connected graphs.

The relation with boolean functions allows us also to associate a Cayley graph to a graph state. We can then use tools from Cayley graph theory for investigations on graph states.

Contrary to the local complementation, the switching action on a graph has not been studied properly with regards to its effects in a graph state. From the observation that local complementation and switching can generate any graph, we construct a formalism to track how these transformations relate graph states.

In our research we have followed an experimental methodology: we have investigated properties and patterns of some mathematical objects using computation as a help to gain insight and intuition. Computer programs for generating graphs and on-line databases of mathematical structures have been used to test our conjectures before proving them formally.

1.1 Overview

- Chapter 2 provides a brief introduction to the theories that lie at the basis of this thesis. In the first section of the chapter we explain, starting from elementary notions, the principles of quantum computing. The *stabilizer formalism*, which will be useful in the following chapters to define graph states, is also presented. The rest of the chapter presents notation and terminology of all the concepts of *graph theory* that we will need in the thesis.

- In Chapter 3, we give three equivalent definitions of the concept of graph state and we explain their role in quantum information processing. Moreover, a section of this chapter deals with the equivalence classes of graph states under local operations. Chapter 2 and 3 contain only previously known results.

- Chapter 4 focuses on the investigation of an invariant of graphs, the number of induced subgraphs with a odd number of edges.

- In Chapter 5 we associate a *cubelike graph* to a graph state. In Section 5.2 we classify graph states according to isomorphism of the cubelike graphs associated with them, while in Section 5.3 and 5.4, we investigate respectively the property of connectedness and the eigensystem of such graphs.

- In Chapter 6, and in particular in Section 6.3, we formally define *Local-Complementation-and-Switching graphs*, a formalism that allows us to track transformations between graph states as edge in a graph. Section 6.2 describes a graph transformation called *switching*. Section 6.4 shows two examples of LCS graphs associated to graphs of order 3 and 4.

Results contained in Chapter 4 constitute the main body of the following research article: A. Bernasconi, A. Cosentino, S. Severini, *On the number of induced subgraphs with odd number of edges*, 2009. (*In preparation.*)

1.2 Notation

This section collects all the conventions we adopted in the thesis about nomenclature and notation.

Given a set X we denote with $|X|$ its cardinality.
$\mathbb{F}_2$ is the finite field of two elements (0 and 1), where arithmetic is performed modulo 2.
We will use the symbol $\oplus$ to denote modulo two addition.
A^T denotes the transpose of the matrix A.
$U^\dagger$ and $\mathbf{x}^\dagger$ denote the Hermitian conjugate of the matrix U and of the vector $\mathbf{x}$.
$\biguplus$ denotes the disjoint union of sets.

2 Preliminaries

In this chapter we introduce the reader to the background necessary to understand the work in this thesis. First we give some background in quantum information science, an emergent field that extends information theory to the quantum mechanics world.

This thesis deals with the concept of graph state, a special case of quantum state. A *quantum state* is a mathematical description of a quantum system. In the mathematical formulation of quantum mechanics, quantum states are vectors in a *Hilbert space*. Since we are interested only in finite dimensions, we will use indifferently the terms Hilbert space and *inner product space*. In the next section we will define the notions of inner product space and *qubit*, the simplest quantum system. Then we describe the *entanglement* of quantum states, a phenomenon without counterpart in classical computation. For a more comprehensive treatment on quantum information processing, see [EHI07, NC00].

The quantum states subject of this thesis can be represented by *simple undirected graphs*. Graphs are objects studied by a branch of mathematics called graph theory. At the end of this chapter we give some basic notation and terminology used in this branch and we present a description of the most familiar classes of graphs. For a survey of graph theory, see [Die05, GR01]. Many definitions and examples can be found in [Wei].

2.1 Quantum information processing

2.1.1 Basic concepts of quantum computing

Inner product space

Definition 1. *An* inner product *over a vector space V is a function $(\cdot,\cdot)$ which takes two vectors $x, y \in V$ as input, produces a complex number as output and satisfies:*

- $(x, x) \geq 0$;
- $(x, x) = 0 \iff x = 0$;
- $(x, y) = (y, x)^\dagger$;
- $(x, y + z) = (x, y) + (x, z)$;
- $(x, \alpha y) = \alpha(x, y)$.

Definition 2. *A vector space equipped with an inner product is called* inner product space.

In quantum mechanics, Dirac notation (invented by Paul Dirac) is the standard notation to describe quantum states. We write $|x\rangle$ and we read "ket x" to emphasize that x is an element of a Hilbert space. An element of the dual Hilbert space is denoted by $\langle x|$ ("bra x"). This is why Dirac notation is also called "bra-ket" notation. In this notation, the inner product of x and y is written as $\langle x|y\rangle$.

Quantum bits

The simplest of all quantum physical system is the *qubit*, short form for *quantum bit*. Quantum bits are the bricks which the quantum computation is built upon. Exactly as a *classical bit*, a qubit has a *state*. The difference is that a qubit can be in a state other than $|0\rangle$ or $|1\rangle$. Indeed, it is possible to form linear combinations of states, called *superpositions*:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle\,,$$

where α and β are complex numbers.

A qubit's state is not observable: we cannot determine the values of α and β. When we *measure* a qubit we get either 0, with probability $|\alpha|^2$, or 1 with probability $|\beta|^2$. Evidently, $|\alpha|^2 + |\beta|^2 = 1$. The quantities α and β are called *amplitudes*.

In general a qubit's state is a unit vector in a Hilbert space.

Let us now define the term *phase* that we will use through the thesis. With the term phase we will always mean the *relative phase*: two amplitudes differ by a relative phase in some basis if each of the amplitudes in the basis is related by such a phase factor. For example, the states

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad \text{and} \qquad \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

differ only by a relative phase shift.

A qubit can be physically implemented with any two-level quantum system, for example the polarisation of a photon or the spin of an electron.

We can have multiple qubits. Suppose, for instance, we have a system of two qubits. Then its state is

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle\,,$$

where α_{00} is the probability of measuring both qubits as zero, and so on. We can also measure just a subset of the qubits. In our two-qubits example, if we measure only the first qubit, the probability of getting $|0\rangle$ is $|\alpha_{00}|^2 + |\alpha_{01}|^2$ and the state remaining after the measurement is

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}\,.$$

This is a simple example of two-particle system.

In general, the notion of *tensor product* is fundamental to describe the state of multiparticle systems. The tensor product, denoted by $\otimes$, is a multiplication between vectors from two spaces, that satisfies the following properties:

- $a(|x\rangle \otimes |y\rangle) = (a|x\rangle) \otimes |y\rangle = |x\rangle \otimes (a|y\rangle)$;
- $(|x\rangle + |y\rangle) \otimes |z\rangle = |x\rangle \otimes |z\rangle + |y\rangle \otimes |z\rangle$;

We will often use the abbreviated notations $|x\rangle|y\rangle$ and $|xy\rangle$ for the tensor product $|x\rangle \otimes |y\rangle$. The joint state of a quantum system with n components, each of them prepared in the state $|\psi_i\rangle$, is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$. A collection of n qubits is called a *quantum register* of size n.

Quantum entanglement

Quantum entanglement is defined by what it is not: a quantum state $|\psi\rangle$ is called entangled if it cannot be described by the product of its component systems. Let us consider the state

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}\,.$$

This is an entangled state since there are no single qubit states $|a\rangle$ and $|b\rangle$ such that $|\psi\rangle = |a\rangle|b\rangle$. In this example, a measurement of the first qubit affects the second qubit. This does not mean that quantum entanglement provides communication faster than light-speed, because the result would be random (*no-communication theorem*). If we want to do teleportation of qubits, we need classical communication, whose speed is limited by the speed of the light.

Given a n-qubit state, if we divide the qubits into two sets and we study the entanglement between them, we are studying *bipartite* entanglement. If the sets are more than two, we are

talking about *multipartite* entanglement.

Many interesting quantum algorithms and protocols are based on phenomenon of entanglement.

Quantum operations

Another important concept in the modelling of a computational framework is what kind of transformations we can operate on the states. The simplest operations act on a single qubit. Since quantum operations are linear, they can be described by matrices. Every operation on a qubit must preserve the normalization condition, so the matrix U that describes it has to be unitary, i.e. $U^\dagger U = I$.

Some of the most important one-qubit operations are the *Pauli* matrices $\sigma_x = X$, $\sigma_y = Y$, $\sigma_z = Z$, I (the identity matrix) and the *Hadamard* matrix H:

$$X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; \quad Y \equiv \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}; \quad Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}; \quad H \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Notice that $H = (X + Z)/\sqrt{2}$ and that $H^2 = I$. Applying the Hadamard matrix we can transform the state $|0\rangle$ into $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, a "halfway" position between $|0\rangle$ and $|1\rangle$.

Starting from elementary operations we can implement complex controlled operations, as the "if-statement", very important both in classical and quantum algorithms. Given a single qubit unitary operation U, *controlled-U* is an operation on two qubits (known as *control qubit* and *target qubit*), that applies U to the target qubit only if the control qubit is set, that is $|c, t\rangle \rightarrow |c\rangle U^c |t\rangle$. The notation U^c is a convention for $U^1 = U$ and $U^0 = I$. The most known two-qubit gate is the controlled-NOT, whose action can be summarized as $|A, B\rangle \rightarrow |A, A \oplus B\rangle$. In this case A is the control qubit and B the target one.

In the following chapters we will use the two-qubit controlled-Z gate (or C_Z), whose definition by a matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The effect of the unitary transformation C_Z is to flip the phase of the target qubit if and only if the control qubit is set to $|1\rangle$. Because of its effect, this transformation is also denoted as controlled-phase (CP). Since controlled-Z is symmetric with respect to exchanging qubits, we do not need to distinguish control from target.

A significant difference between quantum and classical information is that qubits cannot

be copied. It has indeed been proved that a general quantum cloning device cannot exist (see [NC00]).

Quantum Circuits

In many texts of quantum information theory, the most used model of computation is the *quantum circuit model*. It resembles the classical circuit model, with *wires* and *gates*. Wires carry information from one part of the circuit to another. In a quantum circuit, usually, wires are not physical components, but they represent passage of time. Gates operate on the qubits in the circuit and represent the quantum operators described in Section 2.1.1.

Generally all input states of a circuit are assumed to be $|0\rangle$ and, as claimed by the *principle of deferred measurement*, shown in [NC00], measurements can be always moved at the end of the circuit.

Figure 2.1 depicts a circuit with two Hadamard gates and a C_Z gate that transforms the standard basis to

$$
\begin{aligned}
|00\rangle &\to \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) \\
|01\rangle &\to \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle + |11\rangle) \\
|10\rangle &\to \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle + |11\rangle) \\
|11\rangle &\to \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle - |11\rangle)
\end{aligned}
$$

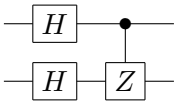


Figure 2.1: Example of quantum circuit

2.1.2 Stabilizer formalism

The stabilizer formalism was introduced in 1997 by Gottesman in [Got07]. We will use this formalism to give an alternative definition of graph states. The following example explains the stabilizer formalism. Consider the state of two qubits

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}\,.$$

It holds that $|\psi\rangle$ is the unique quantum state such that $|\psi\rangle = X_1X_2|\psi\rangle$ and $|\psi\rangle = Z_1Z_2|\psi\rangle$, that is $|\psi\rangle$ is the only quantum state *stabilized* by the operators X_1X_2 and Z_1Z_2. The surprising idea is that quantum states, but also operations and measurements on them, can be more easily described using the operators that stabilize them. A state that can be described by operators that stabilize it is called *stabilizer state*.

We are mostly interested in commutative subgroups of the Pauli group, where the *Pauli group* for a single qubit consists of all the Pauli matrices, together with multiplicative factors ± 1, $\pm i$. More formally,

Definition 3. *A* stabilizer state *is a state of an n-qubit system that is a simultaneous eigenvector of a commutative subgroup of the Pauli group.*

The advantage of using stabilizers to define states and operations on them comes from group theory: generators provide a compact means of describing a group (the saving is exponentially on the cardinality of the group). In the stabilizer formalism, it is possible to efficiently describe different operations, such as controlled-NOT, Pauli operators, Hadamard gate and also measurements. Unfortunately for a wide class of quantum circuits, which include, for example, $\pi/8$ and Toffoli gates, this is not possible.

Keeping track of the generators of the stabilizer corresponding to operations and measurements can be done using $\mathcal{O}(n^2)$ steps on a classical computer. This leads to the *Gottesman-Knill theorem*:

Theorem 4. *Every Clifford circuit (a circuit composed only of Hadamard, Phase and controlled-*NOT *gates), when applied to a state prepared in the computational basis and followed by measurements in the computational basis, can be efficiently simulated on a classical computer.*

2.1.3 Fast quantum algorithms

David Deutsch described the first algorithm that exploits properties of quantum mechanics. Deutsch showed that with a quantum computer it is possible to calculate $f(0) \oplus f(1)$ using only *one* evaluation of $f(x)$. For the same problem, any classical computer would need at least two evaluations of $f(x)$. An important extension of Deutsch's algorithm is *Deutsch-Josza's algorithm* [DJ92], which exponentially speeds up any classical algorithm for the problem of deciding if a function is constant (the same value of $f(x)$ for all values of x) or balanced ($f(x)$ is equal to 1 for exactly half of all the possible x and 0 for the other half).

The most important quantum algorithm known is the *factoring algorithm* [Sho97] introduced by mathematician Peter Shor in 1994. It belongs to the same class of Deutsch-Josza algorithm, since they both exploit interference between qubits using a quantum analog of the

Fourier transform. Shor's algorithm finds the prime factors of a integer N in polynomial time in $\mathcal{O}(\log N)$. If quantum computer were realized, this algorithm would break RSA, a widely-used public-key cryptographic scheme based on the difficulty of factoring large numbers using classical computers. This amazing result motivated researchers to work in the direction of realizing the quantum computer. An NMR (Nuclear Magnetic Resonance) implementation of a quantum computer, able to factorize the number 15 using 7 qubits, was actually realized by a group at IBM.

Another important step in the history of quantum computing was a *quantum search algorithm* [Gro97] developed by Grover in 1997. *Grover's algorithm* sped up substantially the problem of "searching for a needle in a haystack" requiring only $\mathcal{O}(\sqrt{N})$ operations for a space of N elements.

For a detailed survey of quantum computer algorithms, see [Mos08].

2.1.4 Quantum computational complexity

Improving the computational power of classical computers has always been the main purpose of research in quantum computing. The class of problems solvable on a quantum computer with unlimited time and space resources is no larger than the class of problems solvable on a classical computer, but quantum computer may be more efficient than their classical counterparts.

A problem is in the class Bounded-Error Quantum Polynomial Time (**BQP**) if it can be solved with bounded probability of error using a polynomial size quantum circuit. Another requirement is that there has to be a classical polynomial-time algorithm to produce the quantum circuit.

Relating classical and quantum computational complexity theories is of considerable interest. The following significant results have been achieved in this field:

- $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP}$
- $\mathbf{BQP} \subseteq \mathbf{PSPACE}$

where

PSPACE is the class of decision problems solvable by a classical Turing machine in polynomial space;

P is the class of decision problems solvable by a classical Turing machine in polynomial time;

BPP is the class of decision problem solvable by a probabilistic Turing machine such that:

- if the answer is *yes*, then the input is accepted with probability at least 2/3.
- if the answer is *no*, then the input is rejected with probability at most 1/3.

It has not been proved yet if **BQP** $\neq$ **BPP**, that is if quantum computers are more powerful than classical computers.

2.1.5 Quantum error-correction codes

One reason why quantum computers are difficult to build is *decoherence*. In the process of quantum decoherence, some qubits become entangled with the environment. It is impossible to exclude noise completely when we build quantum systems, so we need to find a way to protect information during the computation. The solution is a quantum analog of error-correcting codes in the transmission of information over a noisy channel between classical computers.

The developing of *quantum error-correcting codes* (*QECCs*) is not just a trivial translation of already existing classical codes. In the process of creating redundancy with quantum computers, we have to deal with the following difficulties:

- because of the no-cloning theorem, it is impossible to duplicate the quantum state of a qubit;
- errors are in a continuous space;
- observing the state of a qubit, we destroy information.

Shor [Sho95] introduced in 1995 a code that bypasses these difficulties and is able to correct arbitrary errors on a single qubit. One qubit is encoded as nine qubits and codewords are given by:

$$\begin{aligned} |0\rangle \rightarrow |0_L\rangle &\equiv \frac{(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)}{2\sqrt{2}} \\ |1\rangle \rightarrow |1_L\rangle &\equiv \frac{(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)}{2\sqrt{2}}. \end{aligned} \tag{2.1}$$

Shor code is based on the fact that an error on a qubit may be expanded as a linear combination of the following operators: identity, bit flip, phase flip and the product of bit flip and phase flip.

The study of QECCs led to the development of an important class of quantum codes described using the stabilizer formalism, known as *stabilizer codes* [Got07].

2.2 Graph Theory

2.2.1 Notation and terminology

A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$; thus, the elements of E are distinct pairs of elements from V. The elements of V are the *vertices* of the graph G. An element of E, a pair (v_i, v_j), is called an *edge* of the graph G. If E is a set of ordered pairs of vertices, then the graph G is a *directed* graph, otherwise it is called *undirected*. A *loop* is an edge which starts and ends on the same vertex. A *simple* graph is an undirected graph that has no loops and no more than one edge between any two different vertices. Otherwise it is called *multigraph*. Unless differently specified, in this thesis the term "graph" will denote a "simple graph".

The *order* and the *size* of a graph are respectively the number of its vertices, *i.e.* $|V(G)|$, and the number of its edges, *i.e.* $|E(G)|$.

A vertex v is *incident* with an edge e if $v \in e$; then e is an edge at v. The two vertices incident with an edge are its *endpoints* and an edge *joins* its ends. Two vertices x, y of G are *adjacent* if (x, y) is and edge of G and x is called *neighbour* of y. This is denoted by writing $x \sim y$. The *neighbourhood* of a vertex v, denoted N_v, is the set of vertices that are adjacent to v.

The *degree* of a vertex v, denoted $d(v)$, is the number of edges incident on v. The *maximum degree* $\Delta(G)$ is the largest degree over all vertices; the *minimum degree* $\delta(G)$, the smallest. A graph is *regular of degree* k or *k-regular* if all the vertices have the same degree k.

The *complement* $\overline{G}$ of G is the graph on V with edge set $[V]^2 \setminus E$. The *line graph* $L(G)$ of G is the graph on E in which $x, y \in E$ are adjacent as vertices if and only if they are adjacent as edges in G.

A *path* in a graph is a sequence of consecutive edges. The *length of a path* is the number of edges that the path has. A *cycle* is a path such that start vertex and end vertex are the same. A graph is *connected* if there is a path between all pairs of vertices.

We denote by $\mathcal{S}(V(G))$ the powerset of $V(G)$, that is, the set of all subsets of $V(G)$ For any subset S of $V(G)$, we can define the corresponding *induced subgraph* $G[S]$ as a graph with vertex set S and that contains all edges in G which join nodes in S.

An *edge-induced subgraph* is a subset of the edges of a graph G together with their endpoints. A maximal connected (induced) subgraph of a graph is a *connected component*, or simply a *component*.

Graphs in which labels are assigned to nodes are called *labeled*. Otherwise they are called *unlabeled*.

The *adjacency matrix* of a graph G is the matrix $A(G)$ such that

$$A(G)_{i,j} = \begin{cases} 1, & \text{if } \{i,j\} \in E(G); \\ 0, & \text{if } \{i,j\} \notin E(G). \end{cases}$$

2.2.2 Familiar classes of graphs

A *complete graph* is a graph in which each pair of graph vertices is connected by an edge. The complete graph with n vertices is denoted K_n. The complete graph on one node K_1, that is a single isolated node with no edges, is commonly called *singleton*. An induced subgraph that is a complete graph is called a clique. Any induced subgraph of a complete graph forms a clique. A *cycle graph* is a graph that consists of a single cycle through all nodes and it is denoted C_n. Notice that $K_3 = C_3$.

A *tree* is a connected graph with no cycles. A *binary tree* is a tree such that the degree of each vertex is at most 3. A *path graph* P_n on n vertices is a tree with two nodes of degree 1, and the other $n-2$ nodes of degree 2. A *star graph* S_n is a tree on n nodes with one node having degree $n-1$ and the other $n-1$ having degree 1.

2.2.3 Hypergraphs

A *hypergraph* is a generalization of a graph, where edges can connect any number of vertices. Formally, a hypergraph H is a pair $H = (X, E)$ where X is a set of elements, called vertices, and E is a set of non-empty subsets of X called hyperedges. If all edges have the same cardinality k, the hypergraph is said to be uniform or k-uniform, or is called a k-hypergraph.

For any subset X_i of $X(H)$, we can define a *subhypergraph* of H induced by X_i, as the hypergraph with vertex set X_i and that contains only the superedges of H which connect vertices in X_i.

3

Graph States

Graph states are capturing an important role in quantum information processing. First of all, graph states are fundamental resources for quantum models of computation based on measurements. These models, when it comes to questions about complexity and simulability, are easy to study because they resemble the gate-model in classical computation. In addition, in quantum computation gate-model, graph states are used as codewords in quantum error correction. More precisely, they are the single quantum states encoded by *graph codes*, a particular kind of *stabilizer codes*. A detailed review on graph states is [HDE+07].

3.1 Definitions

In this section we define a graph state. First we give an intuitive definition based on the procedure to construct a graph state. Then we use the stabilizer formalism, described in Section 2.1.2, to give a more compact definition. The first one could be more useful to understand properties of the graph states connected with the classical graph theory and also to understand how they can be prepared in laboratory. The second one is more suitable to reason about the use of graph states in quantum error correction or in quantum computation models.

A graph state is a quantum state made up of several constituents, with the underlying structure of a graph that describes the interactions between these constituents.

Definition 5 (Constructive definition)**.** *Given a graph G, the corresponding graph state $|G\rangle$ is defined associating a qubit in the state $|+\rangle$ with each vertex and applying, for each edge*

between two qubits a and b, the unitary transformation C_Z on the qubits a and b, i.e.,

$$|G\rangle = \prod_{\{a,b\}\in E} C_{Z_{ab}} |+\rangle^V .$$

The choice of the initial state $|+\rangle$ and of the interaction C_Z creates *maximal entanglement* between the qubits. Let us consider the *maximally entangled* state of only two qubits A and B:

$$|\phi\rangle_{AB} = \frac{1}{2}(|00\rangle_{AB} + |01\rangle_{AB} + |10\rangle_{AB} - |11\rangle_{AB}). \tag{3.1}$$

To understand what "maximally entangled" means, we have to introduce the *density operator*. The density operator provides an alternative approach for describing quantum systems whose state is not completely known. The physical system with qubits A and B and state $|\phi\rangle_{AB}$ can be described by the density operator

$$\rho_{AB} = |\phi\rangle_{AB}\langle\phi|.$$

The density operator description becomes very useful for the description of subsystem of a composite quantum system.

If we have a system composed by two qubits A and B, described by the operator ρ_{AB}, we can trace out the operator ρ_{AB} over qubit B to find the density operator for the qubit A. A state is "maximally entangled" if its reduced state at one qubit is a multiple of the identity operator (*maximally mixed*). Then the state in 3.1 is maximally entangled, since

$$\rho_A = tr_B(|\phi\rangle_{AB}\langle\phi|) = \frac{1}{2}\mathbf{1}_A = \frac{1}{2}|0\rangle_A\langle 0| + \frac{1}{2}|1\rangle_A\langle 1|$$

C_Z has also the property that $C_Z^2 = I$ and $C_Z = C_Z^\dagger$, that is, applying once the transformation on two qubits we "create" the edge and applying it again we "destroy" the edge.

Example 6. *For instance, the graph state of the complete graph K_3, sketched in Figure 3.1, is*

$$|K_3\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle - |110\rangle - |111\rangle).$$

Figure 3.2 shows the quantum circuit for the preparation of the graph state $|K_3\rangle$.

With the aim of giving a more formal definition of graph state, we define a Boolean function f_G associated with the graph. Let us denote by S_x the subset of $V(G)$ defined by the string x, and by $G[S_x]$ the subgraph induced by S_x. The function $f_G(x)$ is then defined

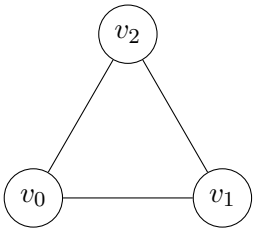


Figure 3.1: The complete graph K_3

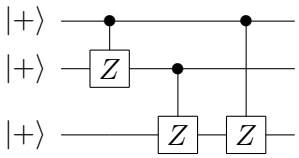


Figure 3.2: Quantum circuit for the preparation of $|K_3\rangle$

as

$$f_G(x) = \begin{cases} 0, & \text{if } |E(G[S_x])| \text{ is even;} \\ 1, & \text{otherwise.} \end{cases}$$

Definition 7 (Formal definition)**.** *Given a graph G, a graph state is a state that is a superposition over all basis states,*

$$|G\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f_G(x)} |x\rangle \,.$$

The two definitions are equivalent: first observe that

$$|+\rangle^{\otimes n} = 2^{-\frac{n}{2}} \sum_{x \in \mathbb{Z}_2^n} |x\rangle \,.$$

Recall that the transformation C_Z flips the sign of the state that is applied to if both the qubits are set to $|1\rangle$. Hence the sign of the coefficient of each computational basis state depends on how many times the transformation C_Z is applied. From the constructive definition, if we see each computational basis state as a subset of the set of vertices, we conclude that the number of applications of C_Z is equal to $|E(G[S])|$.

Alternatively, we can define a graph state using the stabilizer formalism, described in 2.1.2.

Definition 8. *Let $G = (V, E)$ be a graph. The associated graph state $|G\rangle$ is the unique state*

stabilized by the set

$$\{K_v | v \in V\},$$

where $K_v = \sigma_x^v \sigma_z^{N_v}$*, i.e., each generator of the stabilizer corresponds to a vertex* $v \in V$ *of the graph and represents the tensor product of the Pauli matrix* σ_x *on the vertex* v *with a Pauli matrix* σ_z *for each vertex in the neighborhood of* v*.*

Proposition 9. *The class of stabilizer states is strictly larger than the class of graph states.*

3.2 Local unitary and local Clifford equivalence

The study of the properties of the entanglement in stabilizers states leads naturally to an investigation of the action of local unitary operations on stabilizers states.

Definition 10. *An operator* U *that can be written as a tensor product of* 2×2 *unitary matrices is a* local unitary operator *(LU).*

An important subclass of LU operations is known as local Clifford operations, local unitary operations that map the Pauli group to itself under conjugation.

The Clifford group on one qubit is the group of all 2×2 unitary operators which map σ_u to $\alpha_u \sigma_{\pi(u)}$ under conjugation, where $u = x, y, z$, for some $\alpha_u = \pm 1$ and some permutation π of $\{z, y, z\}$. They play an important role in a classification of stabilizer states, since the stabilizer formalism itself is defined in terms of tensor of local Pauli matrices.

Definition 11. *A* local Clifford operator *(LC) on* n *qubits is a tensor product of* n *Clifford operators on one qubit.*

Up to a global phase factor any Clifford operation U can be decomposed into a sequence of $\mathcal{O}(N^2)$ one- and two-qubit gates in the set $\{H, S, \mathrm{CNOT}\}$, where H is defined as in 2.1.1,

$$S \equiv \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad \text{and} \quad \mathrm{CNOT} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

If there exists a local unitary operator U, such that $U|G\rangle = |G'\rangle$, the states $|G\rangle$ and $|G'\rangle$ are called *LU-equivalent* and they will have the same entanglement properties. In the same way, two states $|G\rangle$ and $|G'\rangle$ are called *LC-equivalent* if there exists a local Clifford operator U such that $U|G\rangle = |G'\rangle$.

A polynomial time algorithm has been found to recognize whether two given states are LC-equivalent. This algorithm is based on a graph transformation rule known in graph theory as *local complementation.*

Definition 12. *The* local complement G^i *of a graph* $G = (V, E)$ *at one of its vertices* $i \in V(G)$ *is the graph obtained by complementing the subgraph of* G *induced by the neighborhood* N_i *of* i *and leaving the rest of* G *unchanged.*

In algebraic terminology, the adjacency matrix A_{G^i} of G^i is defined as follows:

$$A_{G^i} = A_G \oplus (A_G)_i (A_G)_i^T \oplus D$$

where A_G is the adjacency matrix of G, $(A_G)_i$ is its i-th column and D is a diagonal matrix such as to yield zeros on the diagonal of A_{G^i}.

Example 13. *The graph in Figure 3.3(b) is obtained from the graph shown in Figure 3.3(a) by local complementation on the black vertex.*

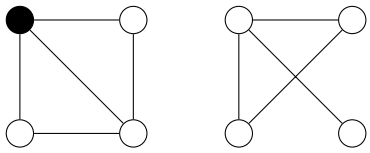


Figure 3.3: Example of local complementation

In Appendix B you can find an implementation written by Hyeyoun Chung (see [Chu08]) of the function that carries out local complementation on a graph, given its adjacency matrix and one of its vertices.

It was shown by Van den Nest et al. ([dNDM04]) that two graph states are equivalent under the local Clifford group if and only if there exists a sequence of local complementations which relates their associated graphs. Since a polynomial time algorithm is known which detects whether two given graphs are related by a sequence of local complementations, there exists a efficient algorithm which recognizes LC-equivalence of graph states.

Moreover, it was proved ([GKR07, Sch07]) that every stabilizer state is LC-equivalent to some graph state and if a particular stabilizer state is given then an LC-equivalent graph state can be found in polynomial time. Then the algorithm can recognize LC-equivalence of all stabilizer states (and not just the subclass of graph states).

It has been conjectured for several years that any two LU equivalent stabilizer states are also LC equivalent (*LU-LC conjecture*). Recently Ji et al. ([JCWY07]) found a counterexample (see Figure 3.4) and proved that the conjecture is false. However it is still not well understood when LU equivalence differs from LC equivalence and no efficient algorithms deciding LU equivalence for stabilizers are known. Another interesting challenge is finding a graph theoretical interpretation of LU equivalent graph states.

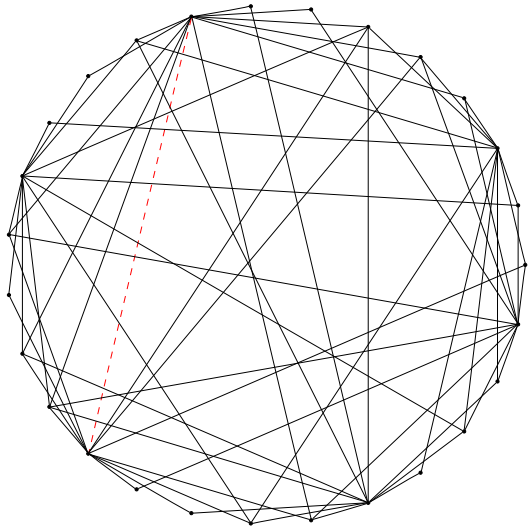


Figure 3.4: LU-LC counterexample.

3.3 One-way quantum computing

One-way quantum computing is a model of quantum computation alternative to the standard quantum circuit model. A one-way computation consists of the preparation of a graph state, followed by single-qubit measurements that destroy the entanglement of the state. Precisely, when the model was introduced by Raussendorf and Briegel [RB01] in 2001, *cluster states* were the class of entangled states considered to serve as a universal "substrate" for the computation. Cluster states are a special case of graph states where the graph G is a two-dimensional square lattice. This is why one-way computing is also known as cluster-state quantum computing (see [Nie05]). Lately, this term is disappearing since models using different graphs have been proposed. The name "one-way" comes from the fact that quantum measurement is fundamentally irreversible, so the entangled state can be used only once, because it is destroyed by the measurements. There are good indications that this model is particularly robust to errors and decoherence.

Let us enter into details on how a one-way quantum computation works. The first step consists in preparing the graph state. In Section 3.1 we explained the construction of a graph state in terms of applying quantum gates. Actually, this preparation process can be done using measurements alone. Then the one-way model may be considered a pure measurement-only model of quantum computation.

The second step is to perform a sequence of *processing single-qubit measurements*, where the choice of measurement basis may depend on the outcomes of earlier measurements. Because of this condition, the time order of the measurements is important. The output of the computation is the quantum state that remains after all the measurements.

4

The *MS-number*

In this chapter we investigate an invariant of graph states. More specifically, we study the number of negative amplitude associated with the basis vectors in the description of a graph state.

4.1 Definitions

From the definition of graph state, the number of "minus signs" in a graph state $|G\rangle$ corresponds to the number of induced subgraphs with odd number of edges in the graph G associated to $|G\rangle$. We call it *MS-number of* G and we denote it with $m(G)$. Another way to see this number is as the cardinality of the function f_G that describes the graph state. The function f_G in Definition 7 is, in other terms, the quadratic Boolean function

$$f_G(\mathbf{x}) = \bigoplus_{\substack{1 \leq i,j \leq n \\ i<j}} A_{ij} x_i x_j = \mathbf{x}^{\mathbf{T}} U_A \mathbf{x}, \tag{4.1}$$

where A is the adjacency matrix of the graph G and U_A is its upper triangular part. We use the notation $|f|$ to indicate the number of strings accepted by a boolean function f, i.e.,

$$|f| = |\{\mathbf{x} \in \{0,1\}^n \mid f(\mathbf{x}) = 1\}|. \tag{4.2}$$

Then $m(G) = |f_G|$.

Moreover we will use the denotation $\overline{m}(G)$ to indicate the number of subgraphs with even size of a graph G, i.e., $\overline{m}(G) = 2^{|G|} - m(G)$.

Example 14. *For the complete graph* K_3 *(see Figure 3.1),* $m(K_3) = \overline{m}(K_3) = 4$.

In the following sections we investigate some properties of $m(G)$ and how to calculate it from the graph in an efficient way. New sequences of MS-numbers for familiar classes of graphs have been found and added to *The On-Line Encyclopedia of Integer Sequences* [Slo09].

4.2 An explicit formula

The function in 4.1 is a *quadratic form* over $\mathbb{F}_2$, that is a homogeneous polynomial in $\mathbb{F}_2[x_1, \ldots, x_n]$ of degree 2, or the zero polynomial.

An explicit formula for the number of solutions of the equation $f(x_1, \ldots, x_n) = 0$ in $\mathbb{F}_2$ can be given. Many of the results in this section are from [LN97].

Definition 15. *Two quadratic forms f and g over $\mathbb{F}_2$ are called* equivalent *if f can be transformed into g by means of a nonsingular linear substitution of variables.*

If f and g are equivalent, then $|f| = |g|$.

Definition 16. *A quadratic form f in n variables is called* nondegenerate *if f is not equivalent to a quadratic form in fewer than n variables.*

Definition 17. *We will call a quadratic form $f \in \mathbb{F}_2[x_1, \ldots, x_n]$* read-once *if every variable x_i appears in f exactly once. If n is odd, $f = x_1 + x_2x_3 + \ldots + x_{n-1}x_n + z$, otherwise $f = x_1x_2 + x_3x_4 + \ldots + x_{n-1}x_n + z$, where $z \in \mathbb{F}_2$ is a constant.*

Lemma 18. *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a nondegenerate quadratic form. Then f is equivalent to a read-once quadratic form.*

Let us consider, as example, the function $f = x_1x_2 + x_1x_3 + x_2x_3$. It is easy to verify that f is equivalent to the read-once quadratic form $g = y_1 + y_2y_3$ and the substitution of the variables can be expressed by the identity $\mathbf{x} = T\mathbf{y}$, where

$$T \equiv \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Lemma 19. *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a read-once quadratic form. If n is odd, $|f| = 2^{n-1}$. If n is even, $|f|$ is equal to $2^{n-1} + (-1)^{z \oplus 1} 2^{\frac{n-2}{2}}$.*

Note also that:

Lemma 20. *We can easily calculate the cardinality of the function $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ from the cardinality of its complement, that is:*

$$|f| = 2^{n-1} - 2^{\frac{n-2}{2}} \iff |f \oplus 1| = 2^{n-1} + 2^{\frac{n-2}{2}}$$

Proof.

$$\begin{aligned} |f| &= |\{\mathbf{x} \in \{0,1\}^n \mid f(\mathbf{x}) = 1\}| = 2^n - |\{\mathbf{x} \in \{0,1\}^n \mid f(\mathbf{x}) = 0\}| \\ &= 2^n - |f \oplus 1| = 2^n - (2^{n-1} + 2^{\frac{n-2}{2}}) = 2^{n-1} - 2^{\frac{n-2}{2}}. \end{aligned}$$

It is also easy to check that $|f| = 2^{n-1}$ *iff* $|f \oplus 1| = 2^{n-1}$. □

Graph Union

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with disjoint vertex sets and edge sets, their union $G = (V, E) = G_1 \cup G_2$ is the graph with $V = V_1 \cup V_2$ and $E = E_1 \cup E_2$.

Isolated vertices can be seen as a particular case of graph union where G_1 or G_2 is an *empty graph*. If $G = \overline{K}_n \cup G_2$ is the union of an empty graph on n nodes and a graph G_2, then $m(G) = 2^n m(G_2)$.

Lemma 21. *The tensor product of two states associated to the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is equal to the graph state associated to their union $G = (V, E) = G_1 \cup G_2$, that is:*

$$|G_1\rangle \otimes |G_2\rangle = |G\rangle.$$

Proof. From the definition of graph state, we can write the above tensor product as:

$$\begin{aligned} &(\frac{1}{\sqrt{2^{|V_1|}}} \sum_x (-1)^{f_{G_1}(x)} |x\rangle) \otimes (\frac{1}{\sqrt{2^{|V_2|}}} \sum_y (-1)^{f_{G_1}(y)} |y\rangle) = \\ &\frac{1}{\sqrt{2^{(|V_1|+|V_2|)}}} \sum_{x,y} (-1)^{f_{G_1}(x) \oplus f_{G_2}(y)} |xy\rangle = \\ &\frac{1}{\sqrt{2^{(|V|)}}} \sum_{x.y} (-1)^{f_G(xy)} |xy\rangle \end{aligned}$$

In the last step of the proof we use the equation

$$f_G(xy) = f_{G_1}(x) \oplus f_{G_2}(y). \tag{4.3}$$

The vector xy is the characteristic representation of a subgraph of G that is union of the

subgraphs of G_1 and G_2 represented by x and y. It follows, considering also the definition of f_G, that $f_G(xy) = 1$ if and only if subgraphs x and y have sizes of different parity. □

Theorem 22. *For $G = G_1 \cup G_2$, it holds:*

$$m(G) = m(G_1)\overline{m}(G_2) + \overline{m}(G_1)m(G_2).$$

Proof. It is straightforward, from Eq. 4.3, that:

$$f_G(xy) = 1 \Leftrightarrow (f_{G_1}(x) = 1 \wedge f_{G_2}(y)) \vee (f_{G_1}(x) = 0 \wedge f_{G_2}(y) = 1)$$

□

This result can be recursively extended to the union of n graphs $G = \bigcup_{i=1}^{n} G_i$:

$$m(G) = m(\bigcup_{i=1}^{n} G_i) = m(G_1)\overline{m}(\bigcup_{i=2}^{n} G_i) + \overline{m}(G_1)m(\bigcup_{i=2}^{n} G_i). \tag{4.4}$$

4.3 Algorithm

As we have seen in 4.1, we can associate a quadratic form in $\mathbb{F}_2$ to a graph G. The problem of calculating the MS-number of G can be seen as the problem of counting the number of solution of the associated quadratic form.

Boolean polynomials $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ of degree at most k are also called k-XOR formulas. Ehrenfeucht and Karpinski described an efficient algorithm to count the number of solutions of an arbitrary 2-XOR-formula (Lemma 4 in [EK90]).

A quadratic form in $\mathbb{F}_2$ is a homogeneous 2-XOR-formula, i.e., there are no linear terms.

Theorem 23. *Given a graph G with n vertices, represented by an adjacency matrix $A_G \in (\mathbb{F}_2)^{n \times n}$, there exists an algorithm working in $\mathcal{O}(n^3)$ time for computing the MS-number of $|G\rangle$.*

We make some improvements to Ehrenfeucht-Karpinski algorithm, exploiting the peculiarities of the functions we deal with. Since there are no linear terms, we can cut the branch "Case 1" in the algorithm. Moreover, if the graph is not connected, we can run an instance of the algorithm for each connected component and calculate the MS-number for the entire graph using Equation 4.4.

An implementation of the algorithm in GNU Octave is shown in Appendix B.

4.4 Binary rank

Godsil and Royle in [GR01] define the binary rank of a graph. In this section we report some results and definitions from [GR01] and then we seek a relation between the binary rank of a graph and its MS-number.

Definition 24. *A* read-once matrix *of order n, n even, is a block diagonal matrix with $n/2$ blocks of the form*

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

We denote it with R_n. If $n \geq 3$ is odd, we will call read-once matrix *of order n a matrix of the form*

$$\left[\begin{array}{c|ccc} 1 & 0 & \dots & 0 \\ \hline 0 & & & \\ \vdots & & R_{n-1} & \\ 0 & & & \end{array}\right]$$

Definition 25. *Let G be a graph with adjacency matrix A_G. The binary rank (or 2-rank) of G is the rank of A_G calculated over $\mathbb{F}_2$ and it is denoted by $rk_2(G)$.*

Theorem 26. *Let A be a symmetric $n \times n$ matrix over $\mathbb{F}_2$ with zero diagonal and binary rank m. Then m is even and there is a $n \times m$ matrix C of rank m such that*

$$A = CR_mC^T$$

where R_m is a read-once matrix of order m.

Proposition 27. *For even n, the read-once matrix R_n is the adjacency matrix associated with a 1-regular graph of order n. For odd n, the read-once matrix R_n is the adjacency matrix associated with the union of a 1-regular graph of order $n-1$ and a self-looped singleton.*

The following theorem relates the operation of local complementation on a graph G, defined in Section 3.2, with the binary rank of its adjacency matrix. G^i denotes the local complement of the graph G at the vertex $i \in V(G)$.

Theorem 28. *Let G be a graph and suppose that u and v are adjacent vertices in G. Then $((G^u)^v)^u = ((G^v)^u)^v$. If G' is the graph obtained by deleting u and v from $((G^u)^v)^u$, then $rk_2(G) = rk_2(G') + 2$.*

4.5 Properties of the MS-number

Let us consider the *empty graph*, that consists of n isolated vertices with no edges. Every subgraph of an empty graph is in turn an empty graph, then all of them have an even number of edges, that is zero. It follows that:

$$m(\overline{K}_n) = 0 \qquad \text{for all } n \geq 1.$$

Proposition 29. *If we exclude the graph P_2, it holds that $m(G)$ is even for all G.*

Proof. This is a consequence of Warning's Theorem (see [LN97]) for polynomial equations over finite fields. □

Two lower bounds and an upper bound can be easily given.

Proposition 30. *It holds that $m(G) \geq |E|$.*

Proof. For each edge we can consider the subgraph induced only by the endpoints of the edge. This subgraph has obviously odd size. □

Proposition 31. *For each order n, the graph with maximum MS-number is $K_4 \cup \overline{K}_{n-4}$, that is:*

$$m(G) \leq m(K_4 \cup \overline{K}_{n-4}), \qquad \text{for } n \geq 4.$$

Proof. $m(K_4 \cup \overline{K}_{n-4}) = 2^{n-4} \cdot 10 = 2^{n-1} + 2^{n-3}$. and $m(G) \leq 2^{n-1} + 2^{(n-2)/2}$ and $2^{n-1} + 2^{(n-2)/2} < 2^{n-1} + 2^{n-3} \iff n \geq 4$ □

4.6 MS-number for familiar classes of graphs

We use the algorithm presented in 4.3 to calculate the MS-number for the classes of graphs defined in Section 2.2.2. From the numerical results we then extrapolate and prove a general closed-form for each class. Only new results are presented in this section.

Complete graphs

Proposition 32. *For a complete graph K_n of order $n \geq 1$ it holds that*

$$m(K_n) = \sum_{i=0}^{\lfloor (n-1)/4 \rfloor} \binom{n+1}{4i+3}.$$

Proof. We know from graph theory that any subgraph induced by a clique is a complete subgraph and that the number of edges in a complete graph K_n of order n is equal to the n-th triangular number $t_n = 1 + 2 + 3 + \ldots + n$. It is also easy to show that the sequence of triangular numbers goes on according to the pattern odd, odd, even, even, odd, odd, ... Then we can easily count the number of the subgraphs of K_n with odd order and claim that

$$m(K_n) = \sum_{i=0}^{\lfloor (n-2)/4 \rfloor} [\binom{n}{4i+2} + \binom{n}{4i+3}] = \\ = \sum_{i=0}^{\lfloor (n-1)/4 \rfloor} \binom{n+1}{4i+3}. \tag{4.5}$$

□

The sequence of numbers that comes out from this formula is: $0, 1, 4, 10, 20, 36, \ldots$, starting from $n = 1$ (in Sloane's *Encyclopedia* [Slo09] as A000749).

Path graphs

Proposition 33. *For $n \geq 1$,*

$$m(P_n) = \begin{cases} 2^{n-1} - 2^{\frac{n-1}{2}}, & \text{if } n \text{ is odd;} \\ 2^{n-1} - 2^{\frac{n-2}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

Proof. A path graph on n vertices corresponds to the function $f_{P_n} = x_1x_2 + x_3x_4 + \ldots + x_{n-1}x_n$. If n is odd, f_{P_n} is equivalent to the readonce function with $n-1$ variables and $z = 0$. Since $n-1$ is even, it holds: $m(P_n) = 2(2^{(n-1)-1} - 2^{\frac{(n-1)-2}{2}}) = 2^{n-1} - 2^{\frac{n-1}{2}}$. If n is even, the function f_{P_n} corresponds to the readonce function with n variables and $z = 0$. Then, $m(P_n) = 2^{n-1} - 2^{\frac{n-2}{2}}$. □

The resulting sequence of numbers is: $0, 1, 2, 6, 12, 28, 56, 120, 240, \ldots$, starting from $n = 1$ (Sloane's A141447).

Cycle graphs

Proposition 34. *For $n \geq 2$,*

$$m(C_n) = \begin{cases} 2^{n-1}, & \text{if } n \text{ is odd;} \\ 2^{n-1} - 2^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

Proof. A cycle graph on n vertices corresponds to the function $f_{C_n} = x_1x_2 + x_3x_4 + \ldots + x_{n-1}x_n + x_nx_1$. If n is odd, f_{P_n} is equivalent to the readonce function with n variables and $z = 0$. Then $m(C_n) = 2^{(n-1)}$. If n is even, the function f_{P_n} corresponds to the readonce function with $n-2$ variables and $z = 0$. Then, $m(C_n) = 2^2(2^{n-2-1} - 2^{\frac{n-2-2}{2}}) = 2^{n-1} - 2^{\frac{n}{2}}$. □

Sequence of numbers: $a(2) = 0, 4, 4, 16, 24, 64, 112, 256, 480, \ldots$ (Sloane's A156232).

Star graphs

Proposition 35. *If S_n is a star graph of order $n \geq 2$, than*

$$m(S_n) = \sum_{i=0}^{\lfloor n/2 \rfloor - 1} \binom{n-1}{2i+1} = 2^{n-2} \tag{4.6}$$

Proof. The thesis follows from the fact that subgraphs with odd size must include the root and an odd number of leaves. □

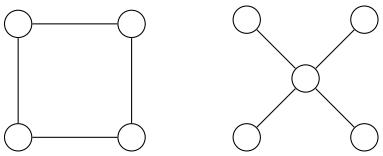


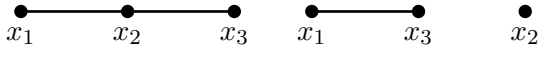
Figure 4.1: C_4 and S_5

Star graphs belong to the more general class of *complete bipartite graphs*. A complete bipartite graph is a special kind of bipartite graph such that every vertex of the first set is connected to every vertex of the second set. If the sets have cardinality p and q, than the graph is denoted with $K_{p,q}$. It is easy to check that:

$$m(K_{p,q}) = 2^{p+q-2}$$

4.7 MS-number and graph isomorphism

Two graphs G and G' are said to be isomorphic if there is a permutation p of $V(G)$ such that $(u, v) \in E(G)$ if and only if $(p(u), p(v)) \in E(G')$. It is obvious that for two isomorphic graphs G and G', it holds $m(G) = m(G')$. The opposite implication is not true, as proven by the following example.



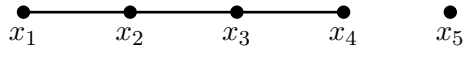
(a) Graph P_3 (b) Graph co-P_3

Figure 4.2: $m(P_3) = m(\text{co-}P_3) = 4$, but $P_3 \ncong \text{co-}P_3$

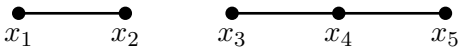
Example 36. *Graphs P_3 and co-P_3 in Figure 36 have the same MS-number, but they are not isomorphic.*

We could doubt that the graphs in the previous example have different MS-number because their sizes are different. This is not true, as you can see in the next example.

Example 37. *Graphs G and H in Figure 37 have the same size and the same MS-number, but they are not isomorphic.*



(a) Graph $G = P_4 \cup K_1$



(b) Graph $H = P_2 \cup P_3$

Figure 4.3: $m(G) = m(H) \wedge |E(G)| = |E(H)|$, but $G \ncong H$

We generated all the *unlabeled* graphs with small number of vertices ($|V| < 9$) using the utility `geng` included in `nauty` [McK03]. Then we used the algorithm to calculate the MS-number for these graphs. Besides that, we repeated this analysis only for *connected* and *bipartite* graphs. Tables 4.1, 4.2 and 4.3 summarize the results. An entry of Table 4.1, for example, tell us how many classes of graphs, with the same ms-number, there are.

Kind of graphs \ Order	1	2	3	4	5	6	7	8	9
all graphs	1	2	3	5	5	7	7	9	9
connected	1	1	2	4	4	6	6	8	8
bipartite	1	2	2	3	3	4	4	5	5

Table 4.1: Number of equivalence classes of graphs for $n = 1 \ldots 9$ on ms-number.

Kind of graphs \ Order	1	2	3	4	5	6	7	8	9
all graphs	1	2	4	11	34	156	1044	12346	274668
all graphs (ms,size)	1	2	4	11	21	51	79	151	204
connected	1	1	2	6	21	112	853	11117	261080
connected (ms,size)	1	1	2	6	12	34	52	111	151
bipartite	1	2	3	7	13	35	88	303	1119
bipartite (ms,size)	1	2	3	7	10	21	29	50	65

Table 4.2: Number of equivalence classes of graphs for $n = 1 \dots 9$ on (ms-number, size).

4.8 MS-number as a partition function

Goldberg et al. [GGJT08] studied the complexity of a family of graph invariants known as *partition functions*.

Let $A \in \mathbb{R}^{m \times m}$ be a symmetric matrix with entries $A_{i,j}$. We denote with $[m] = \{1, \dots, m\}$ the set of row and column indices of the matrix. In context of *partition functions*, elements of this set are called *spins*. A mapping $\xi : V \to [m]$ assigns a spin to each vertex of the graph. The *partition function* Z_A associates with every graph $G = (V, E)$ the real number

$$Z_A(G) = \sum_{\xi : V \to [m]} \prod_{\{u,v\} \in E} A_{\xi(u),\xi(v)}.$$

Now let us consider

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

that is the simplest nontrivial Hadamard matrix. Then, up to a simple transformation, Z_{H_2} counts induced subgraphs of G with an even number of edges. More precisely, our *MS-number* $m(G)$ is equals to $2^{N-1} - \frac{1}{2} Z_{H_2}(G)$.

order (number of classes)	n	size	$m(G)$
1 (1)	1	0	0
2	1	0	0
(2)	1	1	1
3	1	0	0
(4)	3	1	2
	3	2	2
	1	3	4
4	1	0	0
(11)	6	1	4
	12	2	4
	3	2	6
	4	3	4
	12	3	6
	4	3	8
	3	4	4
	12	4	6
	6	5	8
	1	6	10
5	1	0	0
(21)	10	1	8
	15	2	12
	30	2	8
	90	3	12
	10	3	16
	20	3	8
	180	4	12
	10	4	16
	20	4	8
	150	5	12
	102	5	16
	135	6	12
	60	6	16
	5	6	20
	10	6	8
	30	7	12
	90	7	16
	45	8	16
	10	9	20
	1	10	20

Table 4.3: Equivalence classes of graphs based on (order, size, MS-number).

5

Cubelike graphs

We associate a *cubelike graph* to a graph state. A cubelike graph is defined to be any Cayley graph over the Abelian group $(\mathbb{F}_2^n, \oplus)$. In literature, the two elements field, which cubelike graphs are defined over, is denoted with $\mathbb{Z}_2$. In this section we prefer the notation $\mathbb{F}_2$ for uniformity with the previous chapters.

5.1 Definition

Definition 38. *Let Γ be a group with identity element e. Suppose C is a Cayley subset of Γ, that is $e \notin C$ and whenever $g \in C$, then $g^{-1} \in C$. The Cayley graph $X(\Gamma, C)$ of Γ with respect to C is the graph whose vertex set is Γ, with two vertices g and h adjacent if $gh^{-1} \in C$.*

In Definition 7 we associated a Boolean function f_G to a graph state. Given a graph state $|G\rangle$, the set $\Omega_G := \{x \in \mathbb{F}_2^n : f_G(x) = 1\}$ defines a Cayley graph $X_G = X(\mathbb{F}_2^n, \Omega_G)$. In analogy with Ω_G, we define $\overline{\Omega}_G = \{x \in \mathbb{F}_2^n : x \neq \mathbf{0} \wedge \overline{f}_G(x) = 1\}$. We impose the condition that x is different from the zero vector to avoid the presence of self-loops in the graph. The edge set E_{X_G} of the Cayley graph X_G is defined as follows:

$$E_{X_G} = \{(u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \mid f_G(u \oplus v) = 1\}.$$

In Chapter 4 we defined the cardinality of the function f_G as the *MS-number* of G, that is $|f_G| = m(G)$. Cubelike graphs associated to a graph state are regular. In particular, it holds:

Proposition 39. *The graph X_G is regular of degree $m(G)$.*

Proof. Let us consider any vertex v of the graph X_G. The adjacency set of v is the set $A_v = \{u \mid f_G(u \oplus v) = 1\} = \{x + v \mid x \in \Omega_G\}$. The cardinality of the set A_v is then the

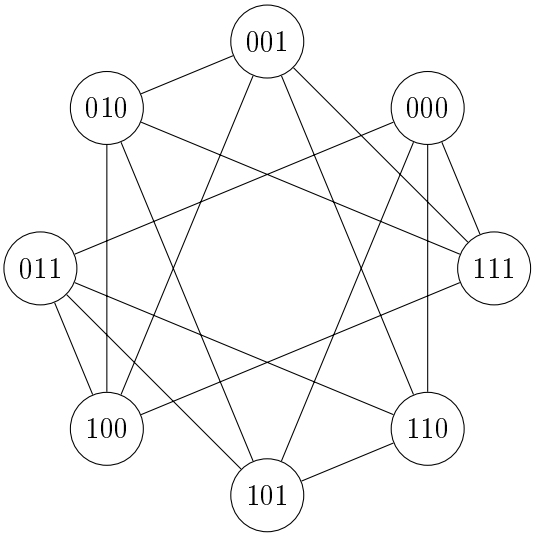


Figure 5.1: The Cayley graph $X(\mathbb{F}_2^n, \Omega_{K_3})$.

same of the set Ω_G. This implies that X_G is a regular graph of degree $|\Omega_G|$. The proposition follows from the fact that $|\Omega_G| = |f_G| = m(G)$. □

For instance, the Cayley graphs $X(\mathbb{F}_2^n, \Omega_{K_3})$ and $X(\mathbb{F}_2^n, \overline{\Omega}_{K_3})$ associated to $|K_3\rangle$ are, respectively, the quartic graph in Figure 5.1 and the well-known 3-cube.

5.2 Isomorphism

It is interesting to define the following relation on the set of graph states: $|G\rangle \sim_{Cayley} |H\rangle$ if and only if the Cayley graphs associated to them are isomorphic, that is $X(\mathbb{F}_2^n, \Omega_G) \cong X(\mathbb{F}_2^n, \Omega_H)$. The above Cayley graphs are isomorphic if there is a permutation p of $\mathbb{F}_2^n$ such that $(u+v) \in \Omega_G$ iff $(p(u)+p(v)) \in \Omega_H$.

First of all we notice that the isomorphism between two Cayley graphs can be determined even if we examine the graphs constructed from the complementary sets.

Lemma 40. *The graphs $X(\mathbb{F}_2^n, \Omega_G)$ and $X(\mathbb{F}_2^n, \Omega_H)$ are isomorphic iff the complementary graphs $X(\mathbb{F}_2^n, \overline{\Omega}_G)$ and $X(\mathbb{F}_2^n, \overline{\Omega}_H)$ are isomorphic.*

Proof. Let h be the isomorphism between $X(\mathbb{F}_2^n, \Omega_G)$ and $X(\mathbb{F}_2^n, \Omega_H)$. Then

$$\begin{aligned}
&(u,v) \in E(X(\mathbb{F}_2^n, \overline{\Omega}_G)) \Leftrightarrow \\
&(u,v) \notin E(X(\mathbb{F}_2^n, \Omega_G)) \Leftrightarrow \\
&(h(u),h(v)) \notin E(X(\mathbb{F}_2^n, \Omega_H)) \Leftrightarrow \\
&(h(u),h(v)) \in E(X(\mathbb{F}_2^n, \overline{\Omega}_H)).
\end{aligned}$$

□

It is obvious that Cayley graphs associated to states contructed from isomorphic graphs are isomorphic.

Lemma 41. $G \cong H \Rightarrow |G\rangle \sim_{Cayley} |H\rangle$.

Proof. Let h be the isomorphism between G and H. We can define another bijection (more exactly a permutation) $\pi : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ such that

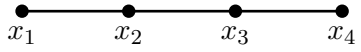
$$\pi(x_1 x_2 \cdots x_n) = x_{h(1)} x_{h(2)} \cdots x_{h(n)}.$$

Let $(u_1 u_2 \cdots u_n, v_1 v_2 \cdots v_n)$ be a generic edge of the Cayley graph associated to G and $z = z_1 z_2 \cdots z_n$ the bitwise sum modulo 2 of the two vertices. This means that $f_G(z_1 z_2 \cdots z_n) = 1$. Besides, if we consider the corresponding vertices $\pi(u_1 u_2 \cdots u_n), \pi(v_1 v_2 \cdots v_n) \in V(X(\mathbb{F}_2^n, \Omega_H))$, their bitwise sum modulo 2 is $\pi(z) = z_{h(1)} z_{h(2)} \cdots z_{h(n)}$. Since $\forall (i,j) \in V(G).(i,j) \in E(G) \Rightarrow (h(i),h(j)) \in E(H)$, we can easily see that the subgraph of H defined by $\pi(z)$ has the same number of edges of the subgraph of G defined by z, then $f_H(\pi(z)) = 1$ or equivalently $(\pi(u), \pi(v)) \in E(X(\mathbb{F}_2^n, \Omega_H))$. □

For the isomorphism of the Cayley graphs associated, the condition of isomorphism of the original graphs is sufficient, but it is not necessary. As counterexamples, let us consider the graphs in Figure 5.2. It is evident that G and H are not isomorphic, while their Cayley graphs are isomorphic.

Let us analyze the polynomials corresponding to these graphs: $f_G = x_1x_2 + x_2x_3 + x_3x_4$ and $f_H = x_1x_2 + x_3x_4$. We can rewrite f_G as $f_G = (x_1 + x_3)x_2 + x_3x_4$, that is $f_G(x) = f_H(h(x))$, where $h : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ is such that $h(x_1x_2x_3x_4) = (x_1 + x_3)x_2x_3x_4$. Then we can weaken the condition of Lemma 41:

Lemma 42. $\exists h$ *linear such that* $\forall x, f_G(x) = f_H(h(x)) \Rightarrow |G\rangle \sim_{Cayley} |H\rangle$



(a) Graph G

(b) Graph H

Figure 5.2: $G \ncong H$, but $|G\rangle \sim_{Cayley} |H\rangle$

Proof. $\exists h$ such that $\forall x, f_G(x) = f_H(h(x)) \Rightarrow \exists h$ such that $\forall x, y, f_G(x+y) = f_H(h(x+y))$. We assumed that h is linear, so $f_G(x+y) = f_H(h(x+y)) = f_H(h(x)+h(y))$, that is the condition of isomorphism for the Cayley graphs associated to the function. □

Lemma 43. $|G\rangle \sim_{Cayley} |H\rangle \Rightarrow \exists p$ *such that* $\forall x, f_G(x) = f_H(p(x))$

Proof. If $X(\mathbb{F}_2^n, \Omega_G)$ and $X(\mathbb{F}_2^n, \Omega_H)$ are isomorphic, from the definition of isomorphism we know that exists a bijection h such that $\forall x, y. f_G(x+y) = f_H(h(x)+h(y))$. From here (if we impose $y = 00\ldots0$), we can also state that h is such that $\forall x. f_G(x) = f_H(h(x)+h(\mathbf{0}))$. Then p can be easily defined as $p(x) = h(x) + h(\mathbf{0})$. □

5.3 Connectedness

A set of elements $g_1, \ldots, g_l$ in a group G is said to *generate* the group G if every element of G can be written as a product of elements from the list $g_1, \ldots, g_l$ and we write $G = \langle g_1, \ldots, g_l \rangle$. A Cayley graph $X(\mathbb{F}_2^n, T)$ is connected if and only if $\mathbb{F}_2^n = \langle T \rangle$.

Lemma 44. *The graph* $X(\mathbb{F}_2^n, \overline{\Omega}_G)$ *is connected.*

Proof. Let $h_n(x) := |\{x_i \in x : x_i = 1\}|$ be the Hamming weight of $x \in \{0,1\}^n$. Let $H_k(n) := \{x : h_n(x) = k\}$. By the definition of $\overline{\Omega}_G$, we have $H_1(n) \subseteq \overline{\Omega}_G$. The statement of the lemma is true, since $H_1(n)$ is the standard generating set of $\mathbb{F}_2^n$. □

For Ω_G, we have an analogue of Lemma 44, but less straightforward. Defining $\overline{\Omega}_{G,i} := \{x : \overline{f}_G(x) = 1 \wedge h(x) = i\}$, we have $\overline{\Omega}_G = \biguplus_{i=2}^n \overline{\Omega}_{G,i}$. For the set Ω_G, we have $\Omega_{G,i} := \{x : f_G(x) = 1 \wedge h(x) = i\}$ and $\Omega_G = \biguplus_{i=1}^n \Omega_{G,i}$.

Lemma 45. *If* G *is a connected graph then* $X(\mathbb{F}_2^n, \Omega_{G,2}) = \biguplus_{i=1}^2 X_i(\mathbb{F}_2^{n-1}, \Omega_{G,2})$.

Proof. Let $(\Omega_{G,2}, \Delta)$ be the closure of $\Omega_{G,2}$ with respect to the symmetric difference of sets Δ. This is defined as $A \Delta B = (A - B) \cup (B - A)$, for sets A and B. Let $\delta_G : \mathcal{S}(V(G)) \longrightarrow \{0,1\}^n$

be the characteristic function of each $S \in \mathcal{S}(V(G))$. Explicitely, $\delta_G(S) = x_1 x_2 \cdots x_n$, with $x_i = 1$ if and only if $i \in S$. Because of δ_G, the symmetric difference corresponds to bitwise addition modulo 2 of elements in $\mathbb{F}_2^n$. Now, given that G connected, $|E(G)| = |\Omega_{G,2}| \geq n-1$. By the pigeonhole principle, since $|V(G)| = n$, it follows that there are three vertices i, j, k such that $\{i,j\}, \{j,k\} \in E(G)$. Therefore $\{i,k\} \in (\Omega_{G,2}, \Delta)$, because $\{i,k\} = \{i,j\}\Delta\{j,k\}$. By repeated applications of Δ, it results that $(\Omega_{G,2}, \Delta) = \biguplus_{p \text{ even}} H_p(n)$. Thus the connected components of $X(\mathbb{F}_2^n, \Omega_{G,2})$ are two isomorphic copies of $X(\mathbb{F}_2^{n-1}, \Omega_{G,2})$. The vertices of these two graphs are labeled by the elements of $\mathbb{F}_2^{n-1} \lhd \mathbb{F}_2^n$ and of its coset, respectively. □

Lemma 46. *Let G be a connected graph on n vertices. Then $X(\mathbb{F}_2^n, \Omega_G)$ is connected, if there is a set $S \subseteq V(G)$ such that $|S|$ and $|E(G[S])|$ are odd.*

Proof. By Lemma 45, $X(\mathbb{F}_2^n, \Omega_G)$ has two isomorphic subgraphs with 2^{n-1} vertices each, labeled by the elements of $\biguplus_{p \text{ even}} H_p(n)$ and $\biguplus_{q \text{ odd}} H_q(n)$, respectively. Suppose that there is a set $S \subseteq V(G)$ such that $|S|$ and $|E(G[S])|$ are odd. Then $\delta_G(S) \in \Omega_{G,q}$, where q is odd. The lemma follows, given that $\mathbb{F}_2^n = \langle H_2(n) \cup \{\delta_G(S)\}\rangle$. In fact, $\delta_G(S)$ gives a perfect matching between the two copies of $X(\mathbb{F}_2^{n-1}, \Omega_{G,2})$. □

The star graph on n vertices is denoted by $K_{1,n-1}$. The following observation is a consequence of Lemma 46.

Lemma 47. *Let $G \cong K_{1,n-1}$. Then $X(\mathbb{F}_2^n, \Omega_G) = \biguplus_{i=1}^2 X_i(\mathbb{F}_2^{n-1}, \Omega_{G,2})$.*

5.4 Eingesystem

The adjacency matrix of $X = X(\mathbb{F}_2^n, \Omega_G)$ is the $2^n \times 2^n$ matrix $A(X) = \sum_{x \in \Omega_G} \rho_{reg}(x)$, where $\rho_{reg}(x)$ is the regular (permutation) representation of $x \in \Omega_G$. Specifically

$$\rho_{reg}(x_1 x_2 \cdots x_n) = \bigotimes_{i=1}^{n} \sigma_x^{x_i}$$

,

It is clear that $A(X)$ commutes with any other adjacency matrix of a Cayley graph of $\mathbb{F}_2^n$, given that this group is abelian. By this fact, the eigensystem of $A(X)$ is readily available, being this matrix diagonalizable by an Hadamard matrix (of Sylvester type) $H^{\otimes n}$:

$$H^{\otimes n} A(X) H^{\otimes n} = \bigoplus_{i=1}^{n} (\lambda_i) = \bigoplus_{i=1}^{n} \left(\sum_{x \in \Omega_G} \chi_i(x) \right),$$

where χ_i is the i-th irreducible character of $x \in \Omega_G$.

6

Local-Complementation-and-Switching Graph

6.1 Introduction

The operation of local complementation, described in 3.2, is deeply studied in the context of graph states. It is a very simple operation and it is efficient to check whether two graphs are related by a sequence of local complementations. Local complementation is also guaranteed to preserve the amount of entanglement into the associated graph states. In this section we introduce another graph transformation called *switching*. Deciding switching equivalence of graphs is polynomial time reducible to graph isomorphism [CC80]. Graph states associated to graphs that belong to different switching classes may have different amount of entanglement. Since every graph state on n qubits can be constructed from the empty graph by a composition of switching and local complementation operations on the empty graph state [Sev06], studying the properties of the switching operator could be useful for the classification of graph states.

6.2 Switching

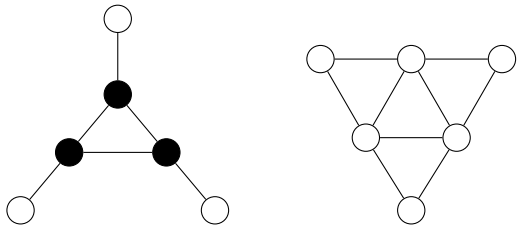
The graph operation of *switching* was introduced by van Lint and Seidel (1966). For a survey, see [Hag01].

Definition 48. *The graph* $G_S^i = (V, E')$ *is the* switching *of* $G = (V, E)$ *at* i *if* $(k, l) \in E'$ *if and only if one of the following two conditions is satisfied:*

1. $(k, l) \in E$ *and* $(k \neq i$ *and* $l \neq i)$*;*
2. $(k, l) \notin E$ *and* $(k = i$ *or* $l = i)$*.*

Instead of a single vertex we can apply the switching operation to a subset σ of V, which is then called *selector*. In other words, given a graph $G = (V, E)$ and a selector $\sigma \subseteq V$,

the *switching* of G by σ is defined as the graph $G_S^\sigma = (V, E')$, which is obtained from G by removing all edges between σ and its complement $V - \sigma$ and adding as edges all nonedges between σ and $V - \sigma$. Figure 6.1(b) shows an example of switching for the graph G in 6.1(a), where σ is the set of black vertices.



(a) G (b) G^σ

Figure 6.1: G^σ is a switch of G

In Appendix B you can find an implementation in GNU Octave of the function that carries out switching on a graph, given its adjacency matrix and a selector σ.

6.3 LCS graphs

A group $\mathcal{G}$ acting on a set Ω is called *transitive* if for every $\alpha, \beta \in \Omega$ there is $g \in \mathcal{G}$ such that $g\alpha = \beta$. Let Ω_n be the set of labeled graphs on n vertices. Ehrenfeucht et al. [AER04] proved that the composition of local complementation and switching forms a transitive group acting on the set Ω_n. In other words, every labeled graph in Ω_n can be constructed from the empty graph by using switchings and local complementation.

Definition 49. *The* Local-Complementation-and-Switching Graph *(for short,* LCS graph*)* G_n *is the graph defined as follows:*

- *the set of vertices of* G_n *is* Ω_n*;*
- *two vertices* X *and* Y *are adjacent in* G_n *if* X *can be obtained from* Y *by a single application of local complementation or switching.*

Paths on such graphs tell us which unitary transformations we should apply to a graph state $|X\rangle$ if we want obtain another graph state $|Y\rangle$ with the same dimension. Once that a physical meaning will be associated to the switching operator, they will also tell us the difference of entanglement between two graph states.

Basic properties

The order of G_n is $|\Omega_n| = 2^{2n(n-1)/2}$, that is the number of labeled graphs on n vertices. The set

$$[X] = \{X^\sigma \mid \sigma \in V(X)\}$$

is called the *switching class* of X. The graph X is called a *generator* of the switching class. It is well known that a switching class of a graph $X = (E, V)$ has $2^{|V|-1}$ graphs [Hag01]. There are $2^{\frac{1}{2}n^2-\frac{3}{2}n+1}$ switching classes that completely partition Ω_n.

6.4 Graph G3 and G4

We show two examples of LCS graphs: G_3 and G_4.

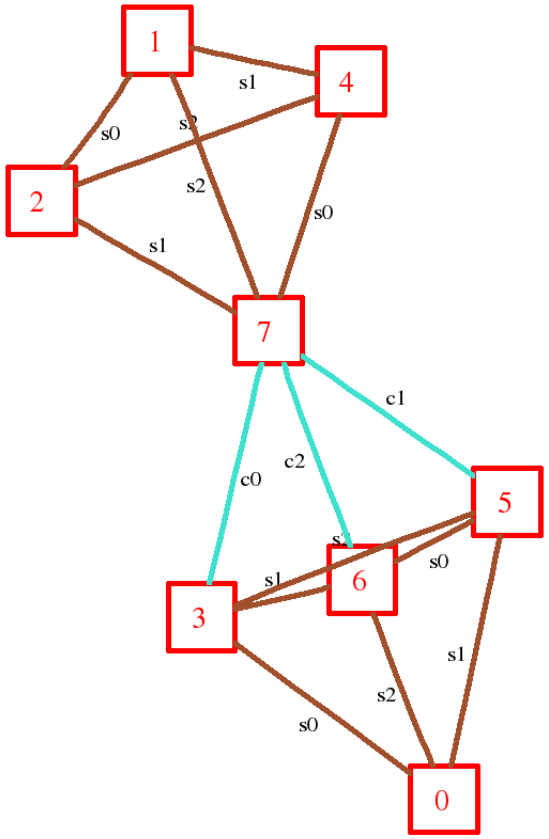


Figure 6.2: The LCS-graph G3

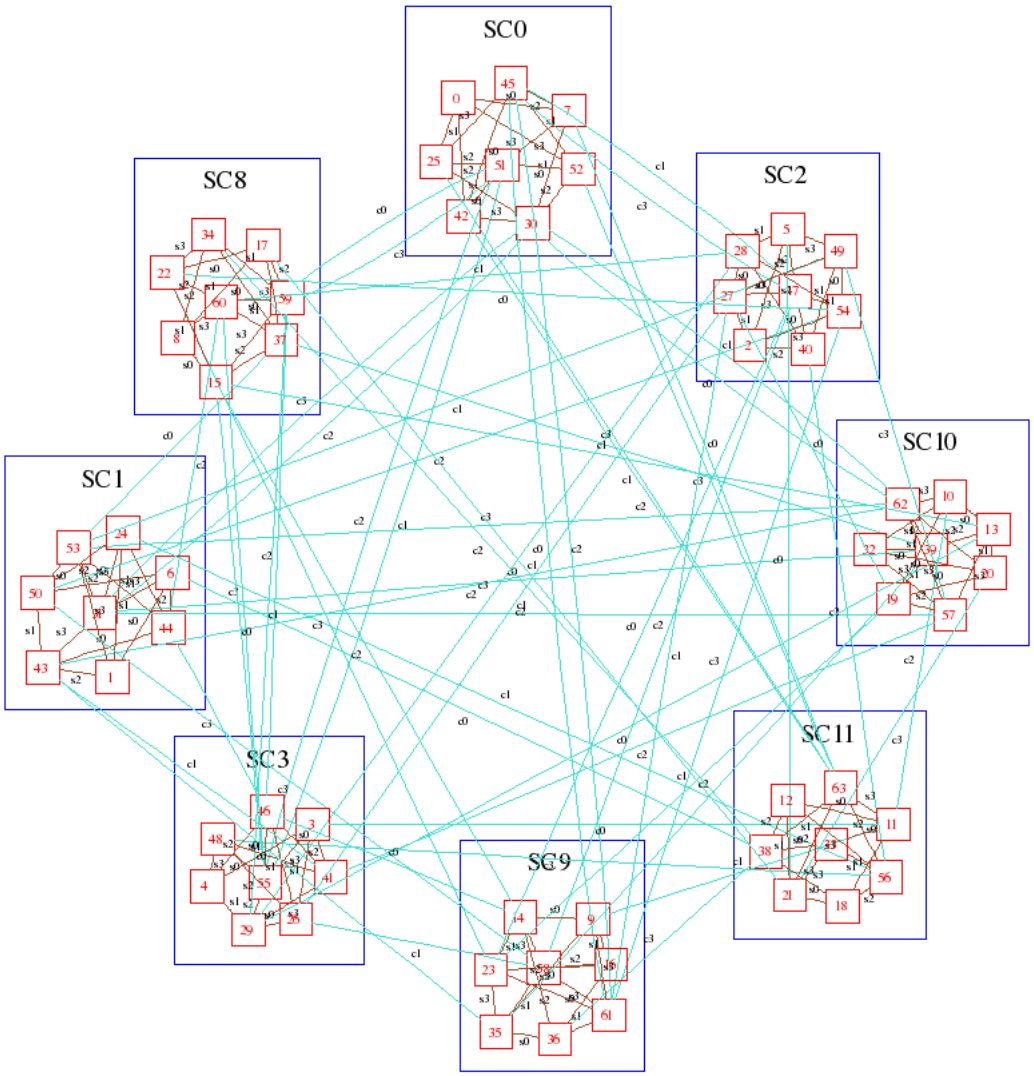


Figure 6.3: The LCS-graph G4

6.5 The conjecture

We observe that, for $n = 3, 4, 5$, the subgraphs induced by the switching classes are, respectively, K_4, $K_{4,4}$ and the *Clebsch graph* (see [Wei] for a description of the Clebsch graph).

Let us define a particular cubelike graph associated with a connected graph. Let X be a connected graph and consider a cubelike graph whose connection set consists of the weight-2 characteristic vectors of the edges of X. This graph has two isomorphic connected components; let $\mathbb{Z}_2(X)$ denote one of these two components [Roy05].

We conjecture the following:

Conjecture 50. *Each of the graphs induced by the switching classes is $\mathbb{Z}_2(C_n)$, where C_n is the cycle graph with n vertices.*

7 Open Problems

There are many interesting open problems connected with the results of this thesis:

1. Characterizing the MS-number with regards to graph transformations (complementation, local complementation, switching).
2. Find the relation between the minimum/maximum MS-number of the graph over its LC orbit and the geometric measure of entanglement of the graph state.
3. How is the structure of the entire LCS graphs G_n?
4. Prove Conjecture 6.5 on LCS graphs.
5. Find the physical operation corresponding to the switching transformation on a graph state.
6. Study the properties of codes corresponding to a 3-hypergraph state (see Appendix A).

A

Extensions

In this chapter we study what happens if we modify the idea of graph state as defined in Section 3.1.

A.1 Edge graph states

Let us try to consider, as inputs of the boolean function f_G, the subsets of $E(G)$, instead of the subsets of $V(G)$. We are then interested no longer to the vertex-induced, but the edge-induced subgraphs. An *edge-induced subgraph* is a subset of the edges of the graph G together with their endpoints. Given a subset S_E of $E(G)$ we can denote the corresponding edge-induced subgraph by $G[S_E]$.

Let $\mathcal{S}(E(G))$ be the powerset of $E(G)$ and $\delta_G : \mathcal{S}(E(G)) \longrightarrow \{0,1\}^n$ the characteristic function of each $S_E \in \mathcal{S}(E(G))$. Then we can define

Definition 51. *The* edge graph state *of G as the vector*

$$|G\rangle := \frac{1}{\sqrt{2^n}} \sum_{S_E \subseteq E(G)} (-1)^{|V(G[S_E])|} |\delta_G(S_E)\rangle.$$

In this section we will name *vertex graph states* the graphs states defined as in Chapter 3. For instance, the edge graph state of the graph G in Figure 5.2(a) is $|G\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle + |111\rangle)$.

Before stating some results about edge graph states we recall further terminology about graph theory.

A *line graph* $L(G)$ of a graph G is obtained by associating a vertex with each edge of the graph and connecting two vertices with an edge if and only if the corresponding edges of G meet at one or both endpoints.

A graph is *claw-free* if and only if it does not contain the complete bipartite graph $K_{1,3}$ (known as *claw*) as a vertex induced subgraph.

The following theorem holds:

Theorem 52. *Given a graph G with maximum degree 2, let $|\phi\rangle$ be its edge-graph state and let $|\psi\rangle$ be the vertex-graph state of its line graph $L(G)$. Then $|\phi\rangle = |\psi\rangle$.*

Proof. If $\Delta(G) \leq 2$ each connected component of the graph G can be either an isolated vertex, a path graph or a cycle graph. Let us notice that the isolated vertices do not affect the structure of the graph states and that subgraphs of a cycle graph are again path graphs. The line graph of a path graph P_n is the path graph P_{n-1}. Since $|V(P_n)| = n$ and $|E(P_{n-1})| = n-2$, the parity is the same, so the corresponding vectors in the graph states will have the same coefficient. Moreover the line graph of the cycle graph C_n is the graph C_n itself, and also in this case the sign will be the same, since $|V(C_n)| = |E(C_n)| = n$. We just proved that $|\phi\rangle = |\psi\rangle$, since they are sums of vectors with the same coefficients.

If $\Delta(G) \geq 3$ than the graph *claw* is an edge-induced subgraph of the graph G. It is easy to see that the line graph of $K_{1,3}$ is the complete graph K_3. Now, since $|V(K_{1,3})| = 4$ and $|E(K_3)| = 3$, the vector corresponding to this subgraph has a positive coefficient in $|\phi\rangle$ and a negative one in $|\psi\rangle$, so the two graph states are different. □

A.2 3-hypergraph states

We seek for mathematical objects "similar to graph states" that correspond to homogeneous polynomials in $\mathbb{F}_2[x_1, \ldots, x_n]$ of degree n higher than 2. This generalization may be represented in graph theory by n-uniform hypergraphs. In this section we analyze the instance $n = 3$. The graph H in Figure A.1 is a 3-hypergraph with four vertices and two edges. Formally, $H = (\{v_1, v_2, v_3, v_4\}, \{e_1 = \{v_1, v_2, v_3\}, e_2 = \{v_1, v_3, v_4\}\})$.

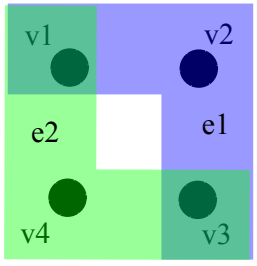


Figure A.1: An example of 3-hypergraph.

The construction of the quantum state associated to the hypergraph is very similar to the case $n = 2$. We associate a qubit in the state $|+\rangle$ with each vertex and we apply, for each edge $e_1 = \{a, b, c\}$, the unitary tansformation controlled-controlled-Z (or controlled-controlled-phase, $\mathsf{C^2P}$) on the qubits a, b, c. The effect of this transformation on three qubits, each of them in the state $|+\rangle$, is:

$$|\phi\rangle = \mathsf{C^2P}(|+\rangle^{\otimes 3}) = |\phi\rangle_{ABC} = \frac{1}{2\sqrt{2}} \sum_{x_1 x_2 x_3} (-1)^{x_1 x_2 x_3} |x_1 x_2 x_3\rangle_{ABC}. \tag{A.1}$$

We call *3-hypergraph state* a quantum state constructed as above.

Under this construction, the quantum state corresponding to the hypergraph H of Figure A.1 is the following:

$$\begin{aligned} |H\rangle = \frac{1}{4}(&|0000\rangle + |0001\rangle + |0010\rangle + |0011\rangle + |0100\rangle + |0101\rangle + |0110\rangle + |0111\rangle + \\ &+ |1000\rangle + |1001\rangle + |1010\rangle - |1011\rangle + |1100\rangle + |1101\rangle - |1110\rangle + |1111\rangle). \end{aligned} \tag{A.2}$$

The circuit in Figure A.2 prepare the system in the state $|H\rangle$.

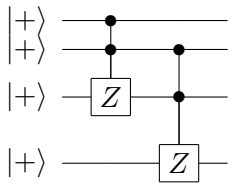


Figure A.2: Quantum circuit for the preparation of $|H\rangle$

It is important to observe that 3-hypergraph states form a class of multipartite entangled states, but the transformation $\mathsf{C^2P}$ does not create "maximal entanglement" between qubits. Indeed, the reduced state at one qubit of the state $|\phi\rangle$ in Equation A.1 is not maximally mixed:

$$\rho_A = tr_{BC}(|\phi\rangle_{ABC}\langle\phi|) = \frac{3}{4}|+\rangle\langle+| + \frac{1}{4}|-\rangle\langle-|. \tag{A.3}$$

Hypergraph states, in particular 3-hypergraph state, may find application in a new model of one-way quantum computation based on natural three-qubit interactions (see [TPKV06]).

The definition of *MS-number* can be easily extended for 3-hypergraph states. The MS-number of a 3-hypergraph state $|H\rangle$ is the number of subhypergraphs of H with odd number of superedges. The complexity class $\#\mathbf{P}$ contains function problems of the form "compute $f(x)$", where f is the number of accepting paths of an $\mathbf{NP}$ machine. It is interesting to

notice that the problem of calculating the MS-number of an arbitrary 3-hypergraph state is #**P**-complete (the proof of this statement comes straightforward from Theorem 1 in [EK90]).

B

Code

GNU Octave

In the implementation of the following programs, we used the extra package `Communications` to deal with vectors and matrices from the Galois field $GF(2)$.

Calculating the MS-number of a graph state

```
% Copyright Alessandro Cosentino (6th February 2009)
% Given the adjacency matrix A of a graph G
% the function MS calculate in polynomial time
% the number of induced subgraphs of G with odd number of edges

function ms = MS(A)

  [T, C, z] = karpinski(A);
  n = size(A, 1);
  m = size(T, 1);

  if (mod(m, 2) == 1)
    ms = 2^(n-1);
  else
    ms = 2^(n-1) + (-1)^(~z)*2^(n-(m+2)/2);
  end

endfunction


% The function KARPINSKI takes as input the adjacency matrix of a graph,
% A, and returns the matrix T that transform A to a readonce matrix
```

```
% using the algorithm Ehrenfeucht–Karpinski

function[T, C, z] = karpinski(A)

  [nr, nc] = size(A);

  if (nr == nc) % check if A is a square matrix
    n = nr;
  else
    error("not a square matrix");
  end
  if ¬(any(any(A))) % check if G(A) is an empty graph
    error("empty graph");
  end
  if ¬(connected(A)) % check if the graph G(A) is connected
    error("not connected graph");
  end

  [T, C, z] = karpinskiRec(A, n, 1);

endfunction


function [T, C, z] = karpinskiRec(A, n, i)

  if (i == n)
    T = [];
    C = [];
    z = 0;
    return;
  end

  alpha = A(i, i+1:n);
  [T_beta, C_beta, z_old] = karpinskiRec(A, n, i+1);

  k = size(T_beta, 1);

  if (k == 0 && any(alpha))
    y_k = [1 zeros(1, n–i)];
    T = [y_k; 0 alpha];
    C = [0 0];
    z = z_old;
    return;
  elseif (k == 0 && ¬any(alpha))
```

```
  T = T_beta;
  C = C_beta;
  z = z_old;
  return;
elseif ¬any(alpha)
  T = [zeros(k, 1) T_beta];
  C = [0 C_beta];
  z = z_old;
  return;
end

%% check if alpha can be expressed as a linear combination
%% of the rows of matrix T_beta
M = [alpha; T_beta];

if (brank(M) ≠ k) % Case 2
  y_k = [1 zeros(1,n-i)];
  T_beta_large = [zeros(k,1) T_beta];
  T = [T_beta_large; y_k; 0 alpha];
  C = [C_beta 0 0];
  z = z_old;
  return;
else % Case 3
  y = gf(T_beta', 1) \ gf(alpha', 1);

  freex = 0;
  betaType = mod(k, 2); % betaType=0 if Type II, =1 if Type I
  if (betaType)
    j = 2;
  else
    j = 1;
  end

  islarge = 0;
  while (j < length(y))
    s = y(j);
    t = y(j+1);
    if (s == 1 && t == 1)
  if ¬islarge
    yp = zeros(k,1);
    yp(j) = 1;
    yp(j+1) = 1;
    T_beta = [yp T_beta];
    islarge = 1;
```

```
else
  T_beta(j,1) = 1;
  T_beta(j+1,1) = 1;
end
freex = freex + 1;
  elseif xor(s,t)
if ¬islarge
  yp = zeros(k,1);
  yp(j) = ¬s;
  yp(j+1) = ¬t;
  T_beta = [yp T_beta];
  islarge = 1;
else
  T_beta(j,1) = ¬s;
  T_beta(j+1,1) = ¬t;
end
  end
  j = j + 2;
end

%% Case 3.c: beta is of type I
if betaType
  dep = y(1); %% dep = alpha is dipendent of y_0 ? (1 yes, 0 no)

  if (¬dep && mod(freex,2) == 1)
if islarge
  T_beta(1,1) = 1;
  T = T_beta;
end
  elseif (dep && mod(freex,2) == 1)
z = xor(z_old, 1);
T(1:k-1,:) = T_beta(2:k,:);
T(k,:) = T_beta(1,:);
T(k+1,:) = [1 zeros(1,n-i)];
C(1:k-1) = C_beta(2:k);
C(k) = xor(C_beta(1), 1);
C(k+1) = 1;
  elseif (dep && mod(freex,2) == 0)
z = z_old;
T(1:k-1,:) = T_beta(2:k,:);
T(k,:) = T_beta(1,:);
T(k+1,:) = [1 zeros(1,n-i)];
C(1:k-1) = C_beta(2:k);
C(k) = C_beta(1);
```

```
    C(k+1) = 1;
      else
    T = T_beta;
    C = C_beta;
    z = z_old;
      end
    else
      %% Case 3.d: beta is of type II and the number of 'free' x is odd
      if (¬betaType)
    z = z_old;
    if (mod(freex,2))
      T = [1 zeros(1, n-i); T_beta];
      C = [0 C_beta];
    else
      T = T_beta;
      C = C_beta;
    end
      end
    end
  end
endfunction
```

Switching

```
%Copyright Alessandro Cosentino (6th February 2009)
%The function SWITCHING takes as input the adjacency matrix of a graph,
%G, and a set of vertices s, and carries out switching on s.

function G_S = Switching(G, s)

[rows, cols] = size(G);

%First check to make sure that the inputs are valid:
%i.e. that each element in s is not pit of range,
%and that G is a square matrix.

if(rows != cols)
  G_S = ['Invalid adjacency matrix.'];
  return
end

if((min(s) < 1) || (max(s) > rows))
  G_S = ['Invalid vertex index.'];
```

```
  return
end

%Calculate the new adjacency matrix.
G_S = G;
G_S(s, :) = not(G_S(s, :));
G_S(:, s) = not(G_S(:, s));
```

Finding switching orbit

With the aim of constructing switching orbits of graphs, we generalize the program `findLCOrbit` from [Chu08] in order to accept any graph transformation. The signature of the function is now:

`function L = findTrOrbit(G, Transformation, disp)`,

where `Transformation` is any function handle.

QCL

QCL is a programming language developed by Bernhard Ömer ([Ö03]) for simulation of quantum algorithms on classical computers. The syntax is derived from procedural languages, such as C or Pascal.

Generating a graph state in QCL

```
int n;
input "order",n;
qureg G[n];
Mix(G);

int i;
int j;
int q;
for i = 0 to n-1 {
    for j = i+1 to n-1 {
        print "edge (",i,",",j,")";
        input "1(yes)/0(no)", q;
        if q==1 {
           CPhase(pi, G[i]&G[j]);
        }
        else {
             if q!=0 {
                exit "input error";
             }
        }
    }
}

dump;
```

Bibliography

[AER04] Tero Harju Andrzej Ehrenfeucht and Grzegorz Rozenberg. Transitivity of local complementation and switching on graphs. *Discrete Mathematics*, 278 Issues 1-3:45–60, 2004.

[CC80] Charles J. Colbourn and Derek G. Corneil. On deciding switching equivalence of graphs. *Discrete Applied Mathematics*, 2:181–184, September 1980.

[Chu08] Hyeyoun Chung. The study of entangled states in quantum computation and quantum information science. Master's thesis, MIT, August 2008. `arXiv:0808.1546`.

[Dan05] Lars Eirik Danielsen. On self-dual quantum codes, graphs, and boolean functions. Master's thesis, Dept. Informat., Univ. Bergen, Norway, Mar. 2005. `arXiv:quant-ph/0503236v1`.

[Dan08] Lars Eirik Danielsen. *On Connections Between Graphs, Codes, Quantum States, and Boolean Functions*. PhD thesis, Dept. Informat., Univ. Bergen, Norway, May 2008.

[Die05] Reinhard Diestel. *Graph Theory*. Springer-Verlag, third edition, 2005. Available from: `http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.counted.pdf`.

[DJ92] D Deutsch and R Jozsa. Rapid solution of problems by quantum computation. *Proc Roy Soc Lond A*, 439:553–558, October 1992.

[dNDM04] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. An efficient algorithm to recognize local clifford equivalence of graph states. *Phys. Rev. A*, 70,:034302, 2004. arXiv:quant-ph/0405023.

[EHI07] Artur Ekert, Patrick Hayden, and Hitoshi Inamori. Basic concepts in quantum computation. 2007. arXiv:quant-ph/0011013.

[EK90] Andrzej Ehrenfeucht and Marek Karpinski. The computational complexity of (xor, and)-counting problems. Technical Report 8543-CS, ICSI - Berkeley, 1990. Available from: citeseer.ist.psu.edu/ehrenfeucht90computational.html.

[Fey81] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 1981.

[GGJT08] Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs. April 2008. arXiv:0804.1932.

[GKR07] Markus Grassl, Andreas Klappenecker, and Martin Roetteler. Graphs, quadratic forms, and quantum codes. 2007. arXiv:quant-ph/0703112.

[Got07] Daniel Gottesman. Stabilizer codes and quantum error correction. 2007. arXiv:quant-ph/9705052.

[GR01] Chris Godsil and Gordon Royle. *Algebraic Graph Theory*. Graduate Texts in Mathematics. Springer, 2001.

[Gro97] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79, 1997.

[Hag01] Jurriaan Hage. *Structural Aspects of Switching Classes*. PhD thesis, Leiden University, May 2001.

[HDE+07] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. Van den Nest, and H. J. Briegel. Entanglement in graph states and its applications. 2007. arXiv:quant-ph/0602096.

[HHHH07] Ryszard Horodecki, Pawel Horodecki, Michal Horodecki, and Karol Horodecki. Quantum entanglement. 2007. arXiv:quant-ph/0702225.

[IR01] IBM-Research. Ibm's test-tube quantum computer makes history, December 2001. Available from: `http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20011219_quantum.html`.

[JCWY07] Zhengfeng Ji, Jianxin Chen, Zhaohui Wei, and Mingsheng Ying. The lu-lc conjecture is false. September 2007. `arXiv:0709.1266`.

[LN97] Rudolf Lidl and Harald Niederreiet. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1997.

[McK03] Brendan D. McKay. nauty user's guide., 2003. Available from: `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`.

[Mos08] Michele Mosca. Quantum algorithms. August 2008. `arXiv:0808.0369`.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Nie05] Michael A. Nielsen. Cluster-state quantum computation. *Rev. Math. Phys*, XX, 2005.

[Ö03] Bernhard Ömer. *Structured Quantum Programming*. PhD thesis, Institute for Theoretical Physics - Vienna University of Technology, 2003. Available from: `http://tph.tuwien.ac.at/~oemer/qcl.html`.

[RB01] R. Raussendorf and H. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86(22):5188–5191, 2001.

[Roy05] Gordon Royle. Colouring cubelike graphs, September 2005. Available from: `http://people.csse.uwa.edu.au/gordon/talks/cubelikevac.pdf`.

[Sch07] D. Schlingemann. Stabilizer codes can be realized as graph codes. 2007. `arXiv:quant-ph/0111080`.

[Sev06] Simone Severini. Two-colorable graph states with maximal schmidt measure. *Physics Letters A*, 356:99–103, July 2006. `arXiv:quant-ph/0511147`.

[Sho95] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52(4):R2493–R2496, Oct 1995.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J.Sci.Statist.Comput.*, 26:1484, 1997. `arXiv:quant-ph/9508027`.

[Slo09] N. J. A. Sloane. *The On-Line Encyclopedia of Integer Sequences*, 2009. published electronically at `http://www.research.att.com/~njas/sequences/`.

[TPKV06] M. S. Tame, M. Paternostro, M. S. Kim, and V. Vedral. Natural three-qubit interactions in one-way quantum computing. *Phys. Rev. A*, 73,:022309, 2006. `arXiv:quant-ph/0507173`.

[Wei] Eric Weisstein. *Wolfram MathWorld*. Available from: `http://mathworld.wolfram.com/`.