uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Laila Rissafi

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Symmetries, Asymmetries and Sense of Direction

TITRE DE LA THÈSE / TITLE OF THESIS

Paola Flocchini

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Amiya Nayak

Nicola Santoro

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Symmetries, Asymmetries and Sense of Direction

A Thesis
Submitted to
The Faculty of Graduate and Postdoctoral Studies

by

Laila Rissafi

For the Degree of a Master in Computer Science
Ottawa - Carleton Institute for Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5

March 2005

Your file Votre référence
ISBN: 0-494-11393-6
Our file Notre référence
ISBN: 0-494-11393-6

To my mother,

To my son Mohammad ,

To my husband.

ACKNOWLEDGEMENTS

I would like to thank Dr. Paola Flocchini for her support during the preparation of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis deals with problems related to the notion of sense of direction in graphs. We define this notion and provide some examples of labelling that realize it. We then survey the problem of minimal sense of direction in regular graphs using symmetric labelling, and its connection to various notions such as cycle symmetry, vertex symmetry, Cayley graph, view, and surrounding. We also present new types of symmetries that are equivalent to having minimal sense of direction. Afterward, we cover the problem of minimal sense of direction in regular graphs using an asymmetric labelling and establish several new results regarding this problem. We show that many classical topologies do not have asymmetric minimal sense of direction using the asymmetric labelling. We conclude this thesis with a program that finds the minimum chordal labelling in an arbitrary graph.

CHAPTER I

INTRODUCTION

A distributed system is a collection of computational entities communicating by exchanging finite amounts of information called messages. Each entity has a local non-shared memory and can communicate by sending to and receiving messages from its neighbors. Every entity has a distinct label (e.g., port number) associated with each of its incident links. Thus, the entire system can be viewed as a graph where each node corresponds to a system entity, and each edge corresponds to a direct communication link between two entities. Furthermore, every edge has two labels, one for each of its incident nodes. Informally, a sense of direction is a particular labelling of the graphs allowing to understand if different paths starting from the same node end in the same nodes or in different nodes. This ability of distinguishing among paths can sometimes prevent the nodes from sending the same message to the same node many times. An example of sense of direction is the mesh where the edge labels are from the set $\{north, south, east, west\}$. In general, any graph could be endowed with a sense of direction. The formal definition of sense of direction is given in Chapter 2. It has been shown in the literature that sense of direction has a definite impact on the computability and complexity in systems of communicating entities, and whose applicability ranges from the analysis of graph classes to distributed object systems.

1.1 Impact of Sense of Direction on Complexity

The evidence of the impact that a specific labelling with sense of direction have on the communication complexity of several problems has been accumulating in recent years (e.g [4, 15, 22, 24, 26, 28, 31, 35, 37]). In general, the investigations have focused on a small set of typical problems that recur in many applications, for example: broadcast, spanning tree construction, depth first traversal, topology recognition, minimum finding, leader election,

edge election, wake-up, etc.

The *leader election* is the process of transforming an initial system configuration, where one or more of the entities are in state *initiator* and the others in a different state, into a final system configuration where one entity is in a *leader* state and all the other entities are in state *lost*. Notice that there are no a priori restrictions on which entity can become *leader*. The *wake-up* problem consists of arriving from an initial configuration where some entities are in the initiator state while the others are in a different state, to a configuration where all the entities are *awake*. The *broadcast* problem consists of arriving from an initial configuration where exactly one entity (the unique initiator) has an information, to a configuration where all the entities have the information. The *minimum finding* problem consists of moving to a final configuration where every entity knows the minimum of the input values. The *depth first traversal* problem consists in moving to a final configuration in which a depth first spanning tree of the graph rooted at the unique initiator has been identified (i.e., every entity knows which of its neighbors belong to the tree). The *spanning tree construction* problem consists of moving to a final configuration in which a spanning tree of the graph rooted at the unique initiator has been identified (i.e., every entity knows which of its neighbors belong to the tree). The *topology recognition* problem consists in moving from an initial configuration where entities do not know the graph and the labelling used, to a final configuration where they know the labelled adjacency matrix of the graph. The *complete topology recognition* problem, in the final configuration of which, besides knowing the labelled adjacency matrix of the labelled graph, each entity must also know its position in the matrix [19].

In an arbitrary topology, sense of direction has an impact on the complexity of many of the previous problems. It has been shown that broadcast and depth first traversal problems can be performed with $\Theta(n)$ messages with any sense of direction even if the system is anonymous. This represents a dramatic improvement from the $\Theta(e)$ bound for these problems for arbitrary labelling. Moreover, leader election, spanning tree construction, and minimum finding problem can be solved with $O(n \log n)$ messages with any sense of

direction; this represents a significant improvement on the $\Theta((e + n \log n)$ bound for these problems in the case of arbitrary labelling.

Sense of direction has an impact also on specific topologies. For example, it was shown in [32] that the *chordal* labelling of a complete graph allows the message complexity of the election process to be reduced from $\Theta(n \log n)$ to $\Theta(n)$ messages, where n is the number of entities in the system. Similarly, it was shown that in *chordal ring*, the chord structure has a critical impact on the complexity of some algorithms. The first result was the observation that the chord structure $S = \{1, 2, ..., k\}$ with $k = O(\log n)$ suffices to obtain a $\Theta(n)$ election algorithms [2]; other chord structures achieving the same bound with the same number of chords were also identified. Subsequent investigations have reduced the size of the structure necessary to achieve linear election algorithms from $O(\log n)$ to $O(\log \log n)$ [27], to $O(\log \log \log n)$ [42], to $O(1)$ [33, 39].

Similarly, it was shown that for the case of the hypercube topology, sense of direction has an impact on the election problem. In absence of sense of direction, the best known solution that has been developed uses $O(n \log \log n)$ messages [16]. In presence of sense of direction, several $\Theta(n)$ election algorithms have been presented [23, 36, 38]. Most of these solutions exploit the implicit region partitioning of the topology and an efficient and implicit scheme to compute and represent shortest paths.

1.2 Impact of Sense of Direction on Computability

A large amount of research has been devoted to the study of computability in anonymous systems; i.e., the study of what problems can be solved when there are no distinct identifiers associated to the nodes (e.g., [3, 12, 13, 29, 30, 34, 41]). Which problem can be solved depends on many factors including the structural properties of the system as well as the amount and type of structural knowledge available to the system entities.

There are two notions in sense of direction. *Weak Sense of Direction* is informally defined by the ability of distinguishing among paths starting from the same nodes, the ones that terminate in the same nodes. The notion of *Sense of Direction* is stronger and involves

an additional property. A formal definition of both notions is given in Chapter 2.

Informally, the surrounding of a node x at a distance d is a subgraph induced by the nodes at distance less that or equal to d from x. The computational power of sense of direction in anonymous system has been studied in [18] and shown to be linked to the notion of *surrounding*, introduced in [17], and that will be covered in more details in Chapter 3.

The general result according to [19] on the impact of sense of direction on the computability is that with *weak sense of direction* no other knowledge is needed. This result is based on the fact that in a labelled graph with *weak sense of direction*, every node can construct its *surrounding* even if the system is anonymous. Note that without sense of direction, with an arbitrary labelling, the problem of constructing a *surrounding* in an anonymous network is unsolvable. A powerful implication of this result is that, with *weak sense of direction*, it is possible to do shortest path routing in anonymous networks, even if there are no global identifiers for neither the source nor the destination. Another interesting consequence of this result is that from a computational point of view, weak sense of direction and sense of direction are equivalent.

The computational power of anonymous systems with *weak sense of direction* has been studied in [18], considering some specific problems: *leader election*, *edge election*, *spanning tree construction*, *topology recognition*, and *complete topology recognition*. The solvability of these problems with sense of direction has been shown to depend on the level of symmetry of the surrounding. In particular, it was shown that *leader election* is solvable in a system with *weak sense of direction* if and only if all surroundings are different. Moreover, *topology recognition* and *complete topology recognition* are solvable for every labelled graph with weak sense of direction.

1.3 Impact of Sense of Direction on Systems: Object Naming

Sense of direction can be used to construct an efficient naming scheme in systems of distributed objects [5].

A distributed object system consists of a collection of objects and relations between objects; each object has a state and a behavior and the global behavior of a system is described in terms of interactions between its objects.

A naming scheme is a mechanism used for naming objects and manipulating them through their names. In object systems, some of the main objectives of a naming scheme are: to designate an object, to distinguish between objects independently of their state; to communicate to other objects the knowledge about some objects. Furthermore, a naming scheme should be stable and reliable, that is, names must remain valid, and keep denoting the same objects when the system changes its state [19].

To achieve these goals, the existing naming schemes either use an approach based on global names, where each object uses as its name an intrinsic property which distinguishes it from the other objects, or they use hierarchical names based on the location of the objects. On the other hand, locality of names and independence on the location are two very desirable characteristics of a naming scheme.

In [5], a naming scheme in which a name of an object depends neither on its state nor on its characteristics, not even on its location, has been proposed based on sense of direction. This approach constitutes the first naming scheme that uses local names, provides mechanisms for correct communication of names and is location independent [19]. Sense of direction has also been shown to be an interesting approach to the problem of naming in the context of object composition [6].

1.4 Graph Theory Results

There have been many studies on sense of direction. Some of these studies are related to testing and constructing sense of direction [e.g., [8, 9, 14]], and others are related to the problem of minimal sense of direction in regular graphs (the number of labels used is equal to the degree of the graph) [7, 11, 17, 21].

It was shown in [9] that there exist polynomial algorithms for testing both weak sense of direction and sense of direction. Moreover, it was shown that deciding weak sense of

direction can be done efficiently in parallel. The algorithm for deciding sense of direction is more complicated that the one for deciding weak sense of direction and its time complexity is $O(n^{14.256} \log n)$.

Since the algorithm for testing sense of direction on a given labelled graph G has a high polynomial complexity, an interesting research direction is the study of how to exploit the topological property of G to find simpler testing algorithms. Many questions were raised and have been answered. Among those questions: in which graphs does every labelling guarantees the existence of sense of direction? for which graphs, the fact that the labelling has a given property is sufficient for the existence of sense of direction?

Several properties of labelling have been considered, and in each case a complete characterization of the corresponding graph class has been given. With arbitrary labelling, just few trivial graphs have sense of direction. The presence of both local orientation and edge symmetry guarantees sense of direction in a larger class of graphs which includes trees and rings. Local orientation is defined by the ability of each node to distinguish among the incident links, and edge symmetry is defined by the ability to understand from the local label what is the label at the other end of the edge. A labelling suffices for having sense of direction in a larger class of graphs which includes particular types of *spiked rings*. As a consequence, testing for sense of direction becomes very easy for graphs in those classes; for example, if G is a tree or a ring, the test consists of checking whether the labelling used is symmetric and has local orientation, which can be done in $O(n)$.

The problem of constructing sense of direction in an unlabelled network has been studied in [39, 40]. It has been shown that any algorithm constructing the traditional sense of direction for cliques, hypercubes, and tori, exchanges at least $\Omega(e - \frac{1}{2}n)$ messages in a network with n nodes and e edges.

The properties of sense of direction with the minimum number of labels has been extensively studied for regular graphs [11, 17, 21]. It has been shown that the existence of minimal sense of direction is related to the notion of *cycle symmetry*. A graph is cycle symmetric if each node belongs to the same number of cycles of the same length. It was

shown that cycle symmetry is a necessary but not sufficient condition for the existence of a minimal sense of direction in a regular graph. Cycle symmetry has been studied in relations to other notions of symmetries in graphs; in particular it has been compared to the notion of *vertex transitivity*, and it was shown that cycle symmetry is sufficient but not a necessary condition for vertex transitivity [21, 25].

The problem of minimal sense of direction for the case of *symmetric labelling* has been studied in [11, 17], and extended to include also the case of non symmetric labelling [7]. For symmetric labelling, minimal sense of direction has been shown to be equivalent to the notion of *Cayley graph with Cayley labelling*, and the notion of *surrounding symmetry* (where all the nodes have the same surrounding). For the case of non symmetric labelling, this necessary and sufficient condition does not hold. A general characterization of minimal sense of direction in directed graphs was given in [7].

This thesis presents other results for the problem of minimal sense of direction. For the case of symmetric labelling, we show that the notion of sense of direction is equivalent to the notion of *labelled cycle symmetric*, and the notion of *labelled vertex transitive.* For the case of asymmetric labelling, we provide several necessary conditions about the labelling and show that the following topologies: the ring, the torus, the hypercube, and the complete graph, cannot be labelled with an asymmetric sense of direction labelling. On the other hand, this thesis shows how to label the complete bipartite graphs using an asymmetric sense of direction labelling.

1.5 The Organization of the Thesis

In this thesis, we introduce the definition of sense of direction and give some examples of sense of direction that were already defined in the literature. The examples are: Cartographic, Chordal, Sum, Contracted, and Neighboring Sense of direction. Then we define two new labelling and show that they are also sense of direction. One of them is a generalization of a class that was already defined (chordal sense of direction), and the other one is a mixture between two classes that were also defined before (cartographic and chordal sense

of direction). In the rest of the thesis, we focus on minimal sense of direction in regular graphs. A sense of direction is said to be minimal if the number of labels used to label the graph are equal to the degree of the graph.

We cover the relationship between minimal sense of direction in regular graphs and the notion of the symmetry. When we talk about symmetry we refer to some regularity that is verified by the nodes or the edges of the graph. We review some results that were already shown in the literature. For example, it was shown that the notion of *minimal symmetric sense of direction* coincides with the notion of *surrounding symmetry* and the notion of *Cayley graph with Cayley labelling* [11, 17]. Then, we define two new notions: *Labelled cycle symmetry* and *Labelled vertex transitivity*, and we show that they are equivalent to the notion of the minimal symmetric sense of direction. After that, we present the relationship between the different notions.

We also study *non symmetric minimal sense of direction* in regular graphs. We give several properties of this type of labelling. In particular, we identify some necessary conditions on the structure of a graph for minimal sense of direction to be verified. We also establish a relationship among the nodes of a graph labelled with minimal sense of direction, and show that it is an equivalence relation. We show that all the equivalence classes contain the same number of nodes. Moreover, we show that many known topologies cannot be labelled in this way. In doing so, we move some steps toward a simpler characterization of graphs supporting minimal asymmetric sense of direction in the case of undirected graphs.

In the last chapter, we solve the problem of minimum chordal sense of direction. It is the problem of finding the minimum number of labels needed in a graph such that it is chordal sense of direction. We write two different programs in Java language, the first one is a backtrack algorithm and it finds the exact solution to the problem, and the other one is a genetic algorithm and it gives a near optimal solution. It is known that the problem of minimum sense of direction is NP hard in general. That is why we limit ourself to writing a program for just the class of chordal labels.

CHAPTER II

BASIC DEFINITIONS

Sense of direction is a property of edge labelled graphs that has been extensively studied in graph theory and distributed computing. Informally, in an edge labelled graph, each node x associates a unique label to each of its incident edges. There is a sense of direction when it is possible to understand from the labels associated to the edges whether different paths from any given node x end in the same node or in different nodes.

In this chapter, we define the notions of edge labelling, path labelling, local orientation, symmetric labelling, coding and decoding function, and then give the definition of sense of direction. After that, we define local naming and redefine the coding and the decoding function in terms of translation capability of local names. We also present some examples of labelling that are sense of direction [20]: Cartographic, Polar, chordal , Sum, Contracted, and Neighboring sense of direction. We provide new classes of labelling: General Group labelling and Chordal-Polar labelling, and show that these labelling are sense of direction.

2.1 Definitions and Terminology

Let $G(V, E)$ be an undirected graph, and $\mathcal{L}$ be a set of labels. Let $E(x)$ denote the set of edges incident to a node $x \in V$. A local edge labelling is any function

$$\lambda_x : E(x) \to \mathcal{L}$$

which associates a label to each edge.

Let $\lambda = \{\lambda_x : x \in V\}$ be defined as the set of labelling functions. Given a labelling λ and a node $x \in V$, let $\Lambda_x : P[x] \to \mathcal{L}^*$ be the path labelling function defined as follows:

for every path $\pi \in P[x_1]$ starting from x_1,

$$\Lambda_{x_1}(\pi) = [\lambda_{x_1}(x_1, x_2), ..., \lambda_{x_m}(x_m, x_{m+1})]$$

where $\pi = [(x_1, x_2), ..., (x_m, x_{m+1})]$.

Let $\Lambda_{(G,\lambda)}[x] = \{\Lambda_x(\pi) : \pi \in P[x]\}$ and $\Lambda_{(G,\lambda)}[x, y] = \{\Lambda_x(\pi) : \pi \in P[x, y]\}$.

Definition 2.1.1. *(Local orientation)*

A labelling λ is a local orientation if $\forall e_1, e_2 \in E(x)$,

$$\lambda_x(e_1) = \lambda_x(e_2) \Longleftrightarrow e_1 = e_2,$$

where $\lambda_x(e_1)$ is the label that x associates to the edge e_1.

In other words, if we have local orientation, any node x can distinguish between its incident edges.

Definition 2.1.2. *(Symmetric labelling)*

A labelling λ is called symmetric when :

$\forall x, y, x_1, y_1 \in V, \quad (x, y) \in E, \quad (x_1, y_1) \in E,$

$$\lambda_x(x, y) = \lambda_{x_1}(x_1, y_1) \Longleftrightarrow \lambda_y(y, x) = \lambda_{y_1}(y_1, x_1).$$

We also say that a pair of labels l, l' are symmetric when:

$$\forall (x, y) \in E \quad \lambda_x(x, y) = l \Longleftrightarrow \lambda_y(y, x) = l'.$$

When a labelling is symmetric, we also say that λ has edge symmetry.

Definition 2.1.3. *(Coding function)*

A coding function c for a labelling λ is any function with domain $\mathcal{L}^*$ such that :

$\forall x, y, z \in V, \quad \forall \pi_1 \in P[x, y], \quad \pi_2 \in P[x, z],$

$$c(\Lambda_x(\pi_1)) = c(\Lambda_x(\pi_2)) \Longleftrightarrow y = z$$

We shall denote by $\mathcal{N}$ the codomain of c.

In other words, a coding function maps paths originating from the same node to the same value if and only if they end up in the same node. In all what follows, when we will be talking about the path labelling which labels are $[a_1, a_2, .., a_n]$, we will denote it by $[a_1, a_2, .., a_n]$ and its coding by $c([a_1, a_2, .., a_n])$.

Definition 2.1.4. *(Weak sense of direction)*

A system (G, λ) has weak sense of direction if and only if it has a coding function.

Definition 2.1.5. *(Decoding function)*

Given a coding function c, a decoding function d for c is any function $d : \mathcal{L} \times \mathcal{N} \to \mathcal{N}$ such that:

$\forall x, y, z \in V$, *with* $(x, y) \in E(x)$ *and* $\pi \in P[y, z]$,

$$d(\lambda_x(x, y), c(\Lambda_y(\pi))) = c(\lambda_x(x, y) \circ \Lambda_y(\pi))$$

where $\circ$ is the concatenation operator.

Definition 2.1.6. *(Sense of direction)*

A system (G, λ) has a sense of direction if and only if the following conditions hold:

1. *There exists a coding function c.*
2. *There exists a decoding function d for c.*

We say also that (c, d) is a sense of direction for (G, λ). If no ambiguity arises, we say that λ is a sense of direction.

The definition of both coding and decoding function can be restated in terms of "translation" capability of local names.

Each node x refers to the other nodes using local names from a finite set $\mathcal{N}$ called name space. Let $\beta_x(y)$ be the name associated by x to y. These local names are not necessarily identities, as the system could be anonymous. The family of injective functions $\beta = \{\beta_x : V \to \mathcal{N} : x \in V\}$ will be called a local naming of G, then the definition of coding and decoding function can also be defined as follows:

Definition 2.1.7. *(Coding function)*

A coding function c of a graph (G, λ) endowed with a local naming β is any function such that:

$\forall x, y \in V, \forall \pi \in P[x, y]$,

$$c(\Lambda_x(\pi)) = \beta_x(y).$$

In other words, a coding function translates the sequence of labels of a path from x to y into the local name that x gives to y.

Definition 2.1.8. *(Decoding function)*

Given a coding function c, a decoding function d for c is any map such that:

$\forall x, y, z \in V, (x, y) \in E(x)$,

$$d(\lambda_x(x, y), \beta_y(z)) = \beta_x(z).$$

In other words, if a node x knows the name that y is giving to z, $\beta_y(z)$, and the label $\lambda_x(x, y)$, then x can translate that into its own local name $\beta_x(z)$.

Definition 2.1.9. *(Sense of direction)*

A sense of direction (c, d) in (G, λ) is symmetric if λ is symmetric.

Definition 2.1.10. *(Homonymous sense of direction)*

A sense of direction (c, d) in (G, λ) is homonymous if

$$\forall x, y, \quad \beta_x(x) = \beta_y(y).$$

In other words, in a homonymous sense of direction, the coding function applied to any two cycles give the same value.

Property 2.1.1. *In undirected graphs, a sense of direction is homonymous if and only if it is symmetric.*

After we have presented all the above definition, we show that local orientation and edge symmetry are not sufficient for a sense of direction.

From Figure (1), we see that local orientation and symmetric labelling does not imply a sense of direction in the cube. For example, from node A we have $c([1]) = c([3, 2, 3])$ but from node B we have $c([1]) \neq c([3, 2, 3])$.

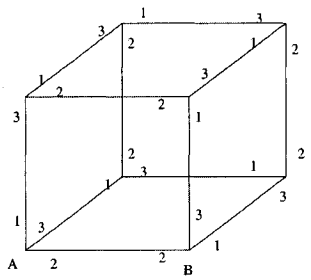

Figure 1: $\mathcal{LO}$ and edge symmetry are not sufficient for SD.

2.2 Examples of Sense of Direction

In the following, five classes of sense of direction are discussed: Cartographic, Chordal, Sum, Contracted, and Neighboring sense of direction.

2.2.1 Cartographic Sense of Direction

A cartographic sense of direction is any sense of direction SD which uses properties of an embedding of $G = (V, E)$ in the plane. Instances of Cartographic SD are: coordinate SD, and Polar SD [20].

Definition 2.2.1. *(Coordinate sense of direction)*

Given an embedding of G in the plane , λ is a coordinate labelling if and only if:

$\forall (u, v) \in E(u)$,

$$\lambda_u(u, v) = (x_1 - x_0, y_1 - y_0),$$

where (x_0, y_0) and (x_1, y_1) are the coordinates of u and v, respectively in the embedding.

Note that the labels are elements of $\mathbb{R}^2$.

We define the coding of the coordinate labelling as follows:

$\forall \pi = [(u_0, u_1), (u_1, u_2), ..., (u_{m-1}, u_m)] \in P[u_0, u_m]$, where $u_i = (x_i, y_i)$,

$$c(\Lambda_{u_0}(\pi)) = c([(x_1 - x_0, y_1 - y_0), ..., (x_m, x_{m-1}, y_m - y_{m-1})]) = (\sum_{i=1}^{m} x_i - x_{i-1}, \sum_{i=1}^{m} y_i - y_{i-1}).$$

We define the decoding function of the coordinate labelling as follows:

$\forall (u_0, u_1) \in E(u_0), \forall \pi = [(u_1, u_2), ..., (u_{m-1}, u_m)] \in P[u_1]$, where $u_i = (x_i, y_i)$,

$d(\lambda_{u_0}(u_0,u_1), c(\Lambda_{u_1}(\pi))) = (x_m - x_0, y_m - y_0)$.

Example of a Coordinate Sense of Direction

Consider the graph of Figure 2 with four nodes u_1, u_2, u_3, u_4 that have as coordinates: $u_1 = (1,0)$, $u_2 = (3,2)$, $u_3 = (-1,1)$, $u_4 = (-2,-2)$. If we take the paths $\pi_1 = [u_1, u_2, u_3]$ and $\pi_2 = [u_1, u_4, u_3]$ we have that: $c(\Lambda(\pi_1)) = c(\Lambda(\pi_2)) = (-2,1)$.

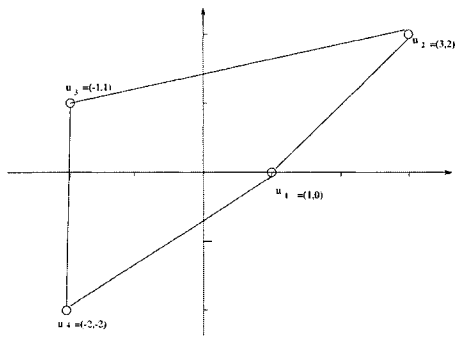

Figure 2: Example of a Coordinate Sense of Direction.

Definition 2.2.2. *(Polar sense of direction)*

Given a graph (G,λ) in polar representation, λ is a polar labelling if and only if:

$\forall (x,y) \in E(x)$,

$$\lambda_x(x,y) = \alpha_{xy},$$

where α_{xy} is the angle under the arc (x,y).

We define the coding function for the polar labelling as follows:

$\forall \pi \in P[x_0]$, $\pi = [(x_0,x_1),(x_1,x_2),...,(x_{m-1},x_m)]$,

$$c(\Lambda_{x_0}(\pi)) = c([\alpha_{x_0x_1}, \alpha_{x_1x_2}, ..., \alpha_{x_{m-1}x_m}]) = \sum_{i=0}^{m-1} \alpha_{x_ix_{i+1}} \mod (2\pi).$$

We define the decoding function for the polar labelling as follows:

$\forall (x_0,x_1) \in E(x_0)$, $\forall \pi = [(x_1,x_2),...,(x_{m-1},x_m)] \in P[x_1]$,

$$d(\Lambda_{x_0}(x_0,x_1), c(\Lambda_{x_1}(\pi))) = (\alpha_{x_0x_1} + c(\Lambda_{x_1}(\pi))) \mod (2\pi).$$

Example of a Polar Sense of Direction

Consider the graph of Figure 3 with four nodes u_1, u_2, u_3, u_4, where $\alpha_{u_1u_2} = \pi/2$, $\alpha_{u_1u_3} = 7\pi/6$, and $\alpha_{u_1u_4} = 11\pi/6$. If we take the paths $\pi_1 = [u_1, u_2, u_3]$ and $\pi_2 = [u_1, u_4, u_3]$, we have that: $c(\Lambda(\pi_1)) = c(\Lambda(\pi_2)) = 7\pi/6$.

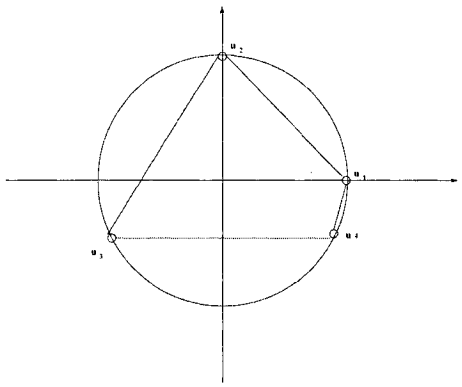

Figure 3: Example of a Polar Sense of Direction.

2.2.2 Chordal and Sum Sense of Direction

In the following, we define the Chordal and the Sum labelling, then we present the coding and the decoding functions for those labelling.

A Chordal labelling of a graph $G = (V, E)$ with $|V| = n$ is defined by fixing a cyclic ordering of the nodes and labelling each incident link by the distance according to this ordering [20].

Definition 2.2.3. *(Chordal sense of direction)*

Let $\gamma : V \to V$ be a successor function defining a cyclic ordering of the nodes of (G, λ), and let $\delta : V \times V \to \{0, ..., n-1\}$ be the corresponding distance function; ie., $\delta(x, y)$ is the smallest k such that $\gamma^k(x) = y$. The labelling λ is a chordal labelling if and only if $\forall (x, y) \in E(x)$,

$$\lambda_x(x, y) = \delta(x, y).$$

In other words, γ is the function defining the cyclic ordering of the nodes, which means that different chordal labelling arise from different functions γ. Note that if x_i is labelling the edges (x_i, x_j) by a label d, then x_j is labelling the edge (x_i, x_j) by the label $n - d$, where

n is the number of vertices of the graph.

We define the coding function for the chordal labelling as follows:

$$\forall \pi \in P[x_0], \quad \pi = [(x_0, x_1), (x_1, x_2), ..., (x_{m-1}, x_m)],$$

$$c(\Lambda_{x_0}(\pi)) = \sum_{i=0}^{m-1} \lambda_{x_i}(x_i, x_{i+1}) \mod (n).$$

We define the decoding function for the chordal labelling as follows:

$$\forall (x_0, x_1) \in E(x_0), \quad \forall \pi = [(x_1, x_2), ..., (x_{m-1}, x_m)] \in P[x_1],$$

$$d(\Lambda_{x_0}(x_0, x_1), c(\Lambda_{x_1}(\pi))) = (\lambda_{x_0}(x_0, x_1) + c(\Lambda_{x_1}(\pi))) \mod (n).$$

Example 2.2.1. Consider the example of Figure 4. Let us denote the node i by x_i. The paths $[2, 1, 1], [3, 1], [1, 2, 1]$ from x_0 to x_4 have the same coding function. In fact:

c([2,1,1])=$(2 + 1 + 1) \mod (6) = 4.$

c([3,1])=$(3 + 1) \mod (6) = 4.$

c([1,2,1])=$(1 + 2 + 1) \mod (6) = 4.$

The nodes have also a consistent decoding function. For example, the coding function from x_0 to x_4 is 4. If x_2 knows the coding that x_0 gives to x_4 and since it knows the label that it gives to x_0, which is 4 , x_2 concludes its coding to node x_4 by applying :

d([4,4])=$(4 + 4) \mod (6) = 2.$

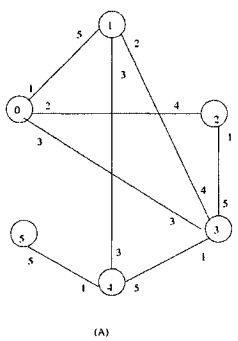

Figure 4: Example of Chordal Sense of Direction.

Informally, the sum sense of direction is defined as follows: in a graph with sum sense of direction, fixing a source node x and a destination node y, all the paths from x to y are

such that the sum of their labels is identical. On the other hand, taking any path from x to a node other than y, the sum of the edge-labels is different. More formally,

Definition 2.2.4. *(Sum sense of direction)*

Let S be a set of rational numbers, and α a string in S^* where $\alpha = \alpha_0, \alpha_1, ..., \alpha_k$. Let $\sum(\alpha)$ be the sum of all the α_i.

The system (G, λ) has a sum sense of direction when there exists a coding and decoding functions such that:

1. $\forall x, y \in V, \forall \pi \in P[x, y]$,
$$c(\Lambda_x(\pi)) = \sum(\Lambda_x(\pi)).$$

2. $\forall x, y, z \in V$, *with* $(x, y) \in E(x)$ *and* $\pi \in P[y, z]$,
$$d(\lambda_x(x, y), c(\Lambda_y(\pi))) = \lambda_x(x, y) + c(\Lambda_y(\pi)).$$

Example 2.2.2. Consider the graph of Figure 5. If we take the two different paths [5,10] and [30,-15] between the nodes x and y and calculate the coding function using the sum sense of direction, we get:

c([3,8])=5+10=15.

c([28,-13])=30-15=15.

For the decoding function, if we take the path labelling [5,10] between x and y, and the edge labelling $\lambda_z(z, x) = -30$, we have:

$d(-30, c([5, 10])) = -30 + 15 = -15.$

Note that the sum labelling can be easily constructed in the same way as a chordal labelling. To label a graph using the sum labelling, we assign different values to the nodes of the graph and label each incident link by applying the subtraction on the values of the incident nodes.

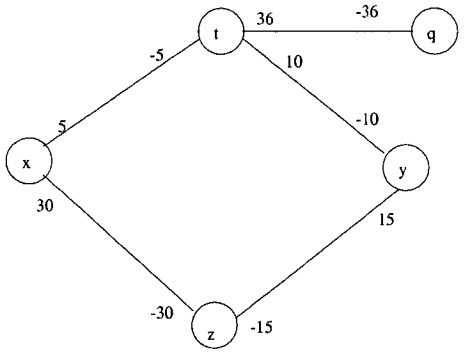

Figure 5: Example of a Sum Sense of Direction.

2.2.3 Contracted Sense of Direction

Given a sequence $\alpha \in \mathcal{L}^*$, the contraction of α is the sequence $\overline{\alpha}$ of labels obtained from α by deleting every pair of symmetric labels l and l', and lexicographically sorting the resulting sequence.

Definition 2.2.5. *(Contracted sense of direction)*

A labelling λ with edge symmetry is contracted if and only if

$\forall x, y \in V$, $\forall \pi_1, \pi_2 \in P[x, y]$,

$$\overline{\Lambda_x(\pi_1)} = \overline{\Lambda_x(\pi_2)}.$$

In other words, if λ is contracted, then all the sequences of all the paths from x to y have the same contraction.

We define the coding function for the contracted labelling as follows:

$\forall \pi \in P[x_0]$, $\pi = [(x_0, x_1), (x_1, x_2), ..., (x_{m-1}, x_m)]$,

$$c(\Lambda_{x_0}(\pi)) = \overline{\Lambda_{x_0, x_m}}.$$

We define the decoding function for the contracted labelling as follows:

$\forall (x_0, x_1) \in E(x_0)$, $\forall \pi = [(x_1, x_2), ..., (x_{m-1}, x_m)] \in P[x_1]$,

$$d(\lambda_{x_0}(x_0, x_1), c(\Lambda_{x_1}(\pi))) = \overline{\lambda_{x_0}(x_0, x_1) \circ c(\Lambda_{x_1}(\pi))}.$$

where $\circ$ is the concatenation operator.

Example of a Contracted Sense of Direction

Consider the mesh of Figure 6 with the traditional compass assignment of labels $\mathcal{L} = \{North, South, East, West\}$. Consider the two paths π_1 and π_2 from x to y with $\Lambda_x(\pi_1) = [North,\ East,\ South,\ South]$ and $\Lambda_x(\pi_2) = [West,\ South,\ East,\ East]$. The contraction of both $\Lambda_x(\pi_1)$ and $\Lambda_x(\pi_2)$ is $[East, South]$.

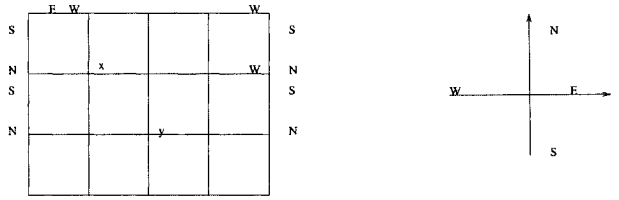

Figure 6: Example of a Contracted Sense of Direction.

2.2.4 Neighboring Sense of Direction

Definition 2.2.6. *(Neighboring sense of direction)*

Given a graph (G, λ), λ is a neighboring labelling if and only if:
$\forall (x,y) \in E(x)$, $(z,w) \in E(z)$,

$$\lambda_x(x,y) = \lambda_z(z,w) \text{ if and only if } y = w.$$

In other words, in a neighboring labelling all the links ending in the same node x are labelled with the same label which we denote by $l_{(x)}$. It was shown in [20] that (G, λ) has a sense of direction.

We define the coding function for the neighboring labelling as follows:

$\forall \pi \in P[x_0]$, $\pi = [(x_0, x_1), (x_1, x_2), ..., (x_{m-1}, x_m)]$,

$$c(\Lambda_{x_0}(\pi)) = \lambda_{x_{m-1}}(x_{m-1}, x_m).$$

We define the decoding function for the neighboring labelling as follows:

$\forall (x_0, x_1) \in E(x_0)$, $\forall \pi = [(x_1, x_2), ..., (x_{m-1}, x_m)] \in P[x_1]$,

$$d(\lambda_{x_0}(x_0, x_1), c(\Lambda_{x_1}(\pi))) = c(\lambda_{x_1}(\pi)).$$

Example of a Neighboring Sense of Direction

Consider the graph of Figure 7 with four nodes u_1, u_2, u_3, u_4. If we take the paths $\pi_1 = [u_2, u_3, u_4]$ and $\pi_2 = [u_2, u_4]$ we have that: $c(\Lambda(\pi_1)) = c(\Lambda(\pi_2)) = 3$.

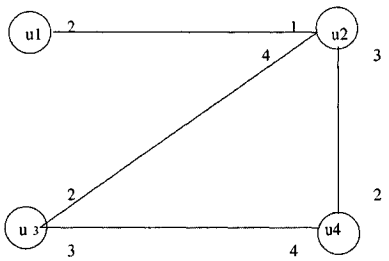

Figure 7: Example of a Neighboring Sense of Direction.

2.3 New Classes of Sense of Direction

In this section, we describe other types of labelling that create sense of direction.

2.3.1 General Group Labelling

We describe a generalization of Chordal labelling. The idea behind this, is that instead of labelling the edges using elements of **Z**, we will choose elements of any group. The idea of using the elements of a group to have sense of direction was done before in [39] but in another way. The author of the paper has used the addition operation in a group and the result works only for abelian groups. The sense of direction that we propose in this thesis uses the multiplication operation in a group and works for any group, abelian or not. In fact, we give an example of how to label the permutation group which is not an abelian group using our generalization of chordal sense of direction.

Definition 2.3.1. *(General group labelling)*

Let $G = (V, E)$ be a graph with $|V| = n$ and $(\mathcal{G}, \cdot)$ be a group with $|\mathcal{G}| = n$. Let $\varphi : V \to \mathcal{G}$ be a bijection that associates to each vertex of V an element of $\mathcal{G}$. Let $\delta : V \times V \longrightarrow \mathcal{G}$ be the function defined as follows:

$\forall (x, y) \in E, \quad \delta(x, y) = \varphi(x).\varphi(y)^{-1}$.

The labelling λ is a general group labelling if and only if, $\forall (x,y) \in E(x)$,

$$\lambda_x(x,y) = \delta(x,y).$$

Theorem 2.3.1. *Let (G,λ) be a graph with a general group labelling, then (G,λ) has a sense of direction.*

Proof. Consider the coding function c defined as follows:

$$\forall \pi \in P[x_0], \quad \pi = ((x_0,x_1),(x_1,x_2),...,(x_{m-1},x_m)),$$

$$c(\Lambda_{x_0}(\pi)) = c([\lambda_{x_0}(x_0,x_1),\lambda_{x_1}(x_1,x_2),...,\lambda_{x_{m-1}}(x_{m-1},x_m)])$$

$$= \prod \lambda_{x_i}(x_i,x_{i+1}),$$

the product being in $\mathcal{G}$. It follows that

$$c(\Lambda_{x_0}(\pi)) = \varphi(x_0)\cdot\varphi(x_m)^{-1} = \delta(x_0,x_m) = \beta_{x_0}(x_m).$$

We now consider the following decoding function d:

$\forall \ (x_0,y_0) \in E(x_0), \quad \forall \pi \in P[y_0]$,

$$d(\lambda_{x_0}(x_0,y_0), c(\Lambda_{y_0}(\pi))) = \lambda_{x_0}(x_0,y_0)\cdot c(\Lambda_{y_0}(\pi)),$$

where the multiplication is in the group $\mathcal{G}$. Then:

$$d(\lambda_{x_0}(x_0,y_0), c(\Lambda_{y_0}(\pi))) = \delta(x_0,x_m) = \beta_{x_0}(x_m).$$

Thus, (G,λ) has a sense of direction. □

Example of a General Group Sense of Direction

As an example of a general group sense of direction we use the permutation group and we show how to label a graph with the elements of this group. We take

$$S_3 = \{(1)(2)(3),(12)(3),(13)(2),(1)(23),(123),(132)\}.$$

Consider the graph of Figure 9. By calculating the labels of the edges based on the multiplication table, we obtain the labelled graph of Figure 10. If we take the paths $\pi_1 = [x_1,x_2,x_3]$ and $\pi_2 = [x_1,x_5,x_3]$ we have that:

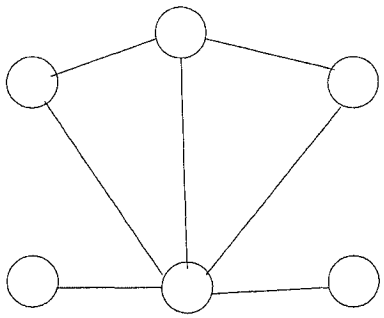

Figure 8: Labelling the nodes of the graph using the permutation group S_3.

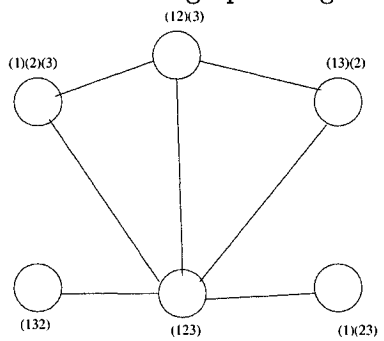

Figure 9: A graph that we want to label using a General Group Sense of Direction.

	(1)(2)(3)	(12)(3)	(13)(2)	(1)(23)	(123)	(132)
(1)(2)(3)	(1)(2)(3)	(12)(3)	(13)(2)	(1)(23)	(123)	(132)
(12)(3)	(12)(3)	(1)(2)(3)	(123)	(132)	(13)(2)	(1)(23)
(13)(2)	(13)(2)	(132)	(1)(2)(3)	(123)	(1)(23)	(12)(3)
(1)(23)	(1)(23)	(123)	(132)	(1)(2)(3)	(12)(3)	(13)(2)
(123)	(123)	(1)(23)	(12)(3)	(13)(2)	(132)	(1)(2)(3)
(132)	(132)	(13)(2)	(1)(23)	(12)(3)	(1)(2)(3)	(123)

Table 1: The multiplication table of S_3.

$c(\Lambda(\pi_1)) = (12)(3) \cdot (123) = (13)(2)$.

$c(\Lambda(\pi_2)) = (132) \cdot (12)(3) = (13)(2)$.

Then, $c(\Lambda(\pi_1)) = c(\Lambda(\pi_2))$.

For the decoding function, if we take the edge (x_5, x_1) and the path $\pi_1 = [x_1, x_2, x_3]$, we have that:

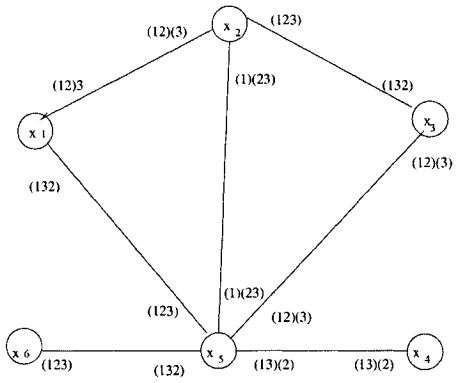

Figure 10: Labelling the edges of the graph using a General Group Sense of Direction.

$d(\lambda_{x_5}(x_5, x_1), c(\Lambda(\pi_1)) = (123) \cdot (13)(2) = (12)(3)$.

2.3.2 Chordal-Polar Sense of Direction

Definition 2.3.2. *(Chordal-Polar sense of direction)*

Let $G = (V, E)$ *be a graph and* $\varphi : V \to \mathbb{R}$ *be an embedding of* V *in* $\mathbb{R}$ *such that:* $\forall x, y \in V, \quad x \neq y \Longrightarrow (\varphi(x) \mod n) \neq (\varphi(y) \mod n)$ *where* $n = |V|$.

For every $x \in V$, *let* $\delta(x) = e^{2\pi i \varphi(x)/n}$,

the Chordal-Polar labelling is defined as follows: $\lambda_x(x, y) = \delta(x)/\delta(y)$.

Note that δ acts as if the vertices are on the unit circle. This labelling is useful if want to work with nodes that are on the unit circle but the nodes that we have are not originally on a unit of a circle. We put the condition:

$\forall x, y \in V, \quad x \neq y \Longrightarrow (\varphi(x) \mod n) \neq (\varphi(y) \mod n)$ *where* $n = |V|$,

so that each two nodes that have different real values assigned to them, will have different locations on the unit circle.

Theorem 2.3.2. *Let* λ *be a Chordal-Polar labelling and* $\forall x, y \in V$, *let* $\beta_x(y) = \delta(x)/\delta(y)$, *then* λ *is a sense of direction.*

Proof. To verify that it is sense of direction, consider the coding function c defined as

follows:

$$\forall \pi \in P[x_0], \quad \pi = ((x_0, x_1), (x_1, x_2), ..., (x_{m-1}, x_m)),$$

$$c(\Lambda_{x_0}(\pi)) = c([\lambda_{x_0}(x_0, x_1), \lambda_{x_1}(x_1, x_2), ..., \lambda_{x_{m-1}}(x_{m-1}, x_m)])$$

$$= \prod \lambda_{x_i}(x_i, x_{i+1}).$$

It follows that

$$c(\Lambda_{x_0}(\pi)) = e^{2\pi i(\varphi(x_0)-\varphi(x_m))/n} = \delta(x_0)/\delta(x_m) = \beta_{x_0}(x_m).$$

Consider the following decoding function d:

$$\forall (x_0, y_0) \in E(x_0), \forall \pi \in P[y_0],$$

$$d(\lambda_{x_0}(x_0, y_0), c(\Lambda_{y_0}(\pi))) = \lambda_{x_0}(x_0, y_0) \times c(\Lambda_{y_0}(\pi)).$$

It follows that

$$d(\lambda_{x_0}(x_0, y_0), c(\Lambda_{y_0}(\pi))) = \delta(x_0)/\delta(x_m) = \beta_{x_0}(x_m).$$

Then, (G, λ) has a sense of direction.

Example 2.3.1. Consider the graph of Figure 11 with real numbers associated to each node such that those values verify the condition:

$\forall x, y \in V, \quad x \neq y \Longrightarrow (\varphi(x) \mod n) \neq (\varphi(y) \mod n)$ *where* $n = |V|$.

We calculate $\delta(x)$ for each $x \in V$ and put those values on the unit circle. We have that:

$a1 = \delta(a) = e^{4\pi i/5}$.

$b1 = \delta(b) = e^{2\pi i/5}$.

$c1 = \delta(c) = e^{3\pi i/5}$.

$d1 = \delta(d) = e^{6\pi i/5}$.

$e1 = \delta(e) = e^{\pi i}$.

Figure 12 shows the result of this operation.

We give some examples of the coding and the decoding function for the Chordal-Polar sense of direction.

Let $\pi_1 = [a1, b1, e1]$, $\pi_2 = [a1, b1, c1, d1, e1]$, $\pi_3 = [a1, e1]$, $\pi_4 = [a1, b1]$, $\pi_5 = [a1, c1]$.

The coding of the path labelling $\Lambda_{a1}(\pi_1)$, $\Lambda_{a1}(\pi_2)$, $\Lambda_{a1}(\pi_3)$, $\Lambda_{a1}(\pi_4)$, $\Lambda_{a1}(\pi_5)$ are calculated as follows:

$c(\Lambda_{a1}(\pi_1)) = c(\Lambda_{a1}(\pi_2)) = c(\Lambda_{a1}(\pi_3)) = e^{9\pi i/5}$.

$c(\Lambda_{a1}(\pi_4)) = e^{2\pi i/5}$.

$c(\Lambda_{a1}(\pi_5)) = e^{\pi i/5}$.

And the decoding of the path which starts in $b1$ and ends in $e1$ is calculated as follows: $d(\lambda_{b_1}(b_1, a_1), c(\Lambda_{a1}(\pi_1))) = \frac{e^{2\pi i/5}}{e^{4\pi i/5}} \times e^{9\pi i/5} = e^{7\pi i/5}$.

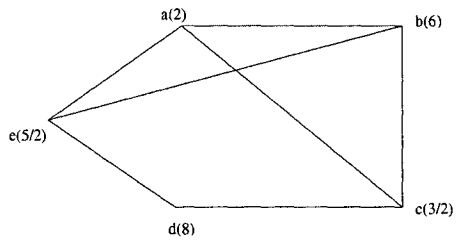

Figure 11: Placing the new values of the nodes on the unit circle.

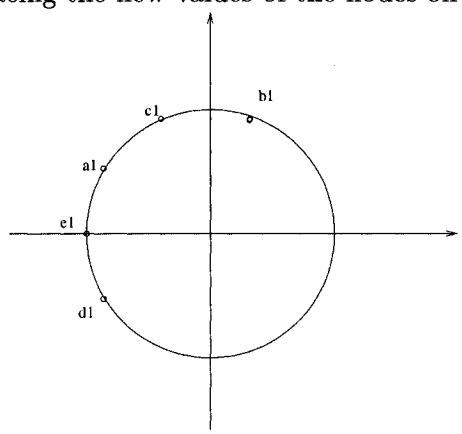

Figure 12: A graph with real values associated to each node.

CHAPTER III

SYMMETRIES AND MINIMUM SENSE OF DIRECTION

A **minimum** sense of direction is a labelling that is sense of direction and uses the smallest number of labels. The number of vertices is a trivial upper bound for the number of labels and it can always be achieved by using the chordal labelling. The maximum degree Δ is a trivial lower bound because any sense of direction is a local orientation, and to have local orientation at least Δ different labels are needed. A graph has **minimal** sense of direction whenever it has sense of direction and the number of labels is equal to its maximum degree. Minimum and minimal sense of direction have been extensively studied and a complete characterization of graphs having minimum sense of direction has been given in [11, 17] in the case of sense of direction with edge symmetry. In particular, it has been shown that there is a strong relationship between some common notions of symmetries in graphs and minimal sense of direction.

In this chapter, we first introduce some notions of symmetries in unlabelled and labelled graphs. Then, we review the results on minimal sense of direction in graphs with edge symmetry. When describing the relationship between different symmetries, we also close an open problem by establishing two new relationships. The first one is between "labelled cycle symmetry" and minimal sense of direction, and the second relationship is between "labelled vertex transitivity" and "surrounding symmetry". In what follows, we define the notion of vertex transitivity, cycle symmetry, Cayley graph, view, surrounding, labelled vertex symmetry, labelled cycle symmetry, and then present the relationship among those notions.

3.1 Unlabelled Symmetries

In this section, we define some common symmetries in unlabelled graphs. In all what follows, when we will be talking about labels a, b such that $\lambda_x(x,y) = a$ and $\lambda_y(y,x) = b$ for some nodes x and y, we will denote by $(a,b)_x$ the fact that the pair of labels (a,b) is incident on node x. We also consider only regular graphs. When no ambiguity arises, we simply denote $G = (V,E)$ with degree d by G.

First, we recall the notion of a vertex automorphism of a graph.

Definition 3.1.1. *(Vertex automorphism)*

A vertex automorphism of $G = (V,E)$ is a bijection $\mathcal{V} : V \longrightarrow V$ such that:

$(\mathcal{V}(u), \mathcal{V}(v)) \in E$ *if and only if* $(u,v) \in E$

The automorphisms of G together with the composition operation form a group $Aut(G)$ that we shall call the automorphism group of G.

Definition 3.1.2. *(Vertex transitive)*

$G = (V,E)$ is vertex transitive if the automorphism group of G, $Aut(G)$, is transitive over V, in other words, if $\forall u,v \in V$, $\exists \alpha \in Aut(G)$ such that $\alpha(u) = v$.

For any node $x \in V$, we denote with $C_x[i]$ the number of simple cycles of length i to which x belongs.

Example 3.1.1. The graph of Figure 13 is an example of vertex transitive graph.

We can see that by considering the two subsets $S = \{a,b,c,d,e\}$ and $S' = \{f,g,h,i,j\}$. If we are looking for an automorphism α such that $\alpha(x) = y$ where $x,y \in S$ or $x,y \in S'$, a simple rotation of the graph shows that such an automorphism exists. On the other hand, if we are looking for an automorphism α such that $\alpha(x) = y$ and where $x \in S'$ and $y \in S$, we can always find it by applying simple rotations and using the automorphism α' defined as follows: $\alpha'(a) = f$, $\alpha'(b) = i$, $\alpha'(c) = g$, $\alpha'(d) = j$, $\alpha'(e) = h$, $\alpha'(f) = a$, $\alpha'(g) = d$, $\alpha'(h) = b$, $\alpha'(i) = e$, $\alpha'(j) = c$, see Figure 14. Note that with α', the set S will be inside the graph and S' will be outside it. We can apply the same process if we are looking for an automorphism α such that $\alpha(x) = y$ where $x \in S$ and $y \in S'$.

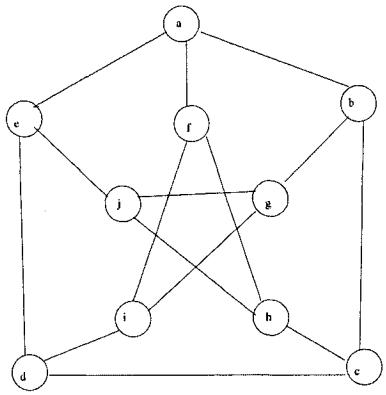

Figure 13: Peterson graph is an example of vertex transitive graph.

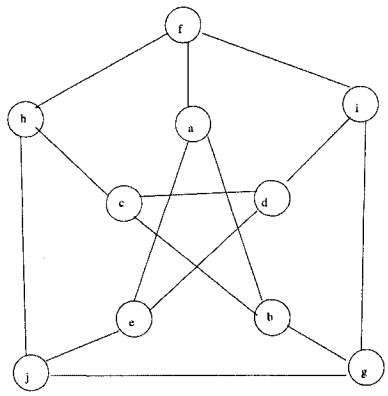

Figure 14: A permutation of the nodes in Peterson graph.

Definition 3.1.3. *(Cycle symmetric)*

$G = (V, E)$ *is cycle symmetric if and only if:*

$$\forall i \in \mathbb{N}, \quad \forall x, y \in V, \quad C_x[i] = C_y[i].$$

In other words, a graph is cycle symmetric when all the nodes belong to the same number of simple cycles of the same length. The cycles C_n, the complete graphs K_n, and the hypercubes H_n are all examples of cycle symmetric graphs.

It was shown in [21, 25] that vertex transitivity is necessary but not sufficient condition for cycle symmetry.

3.2 Labelled Symmetries

In this section, we define some symmetries in labelled graphs. First we remind the notion of Cayley graphs. Given a set of generators Ω for a finite group $(\Gamma, \cdot)$, a Cayley graph is a graph $G_\Gamma = (V, E)$, where the vertices correspond to the elements of the group ($V = \Gamma$) and the edges correspond to the action of the generators; that is $(x, y) \in E$ if and only if $\exists g \in \Omega \quad x \cdot g = y$, where $\cdot$ is the group operation. The set of generators is closed under inverses, so we can consider the graph undirected. The natural labelling λ for a Cayley graph G_Γ is the following: $\forall (x, y) \in E(x)$, $\lambda_x(x, y) = g$, where g is the generator such that $y = x \cdot g$. In the following, we shall call this labelling Cayley labelling.

We now define the notion of view and view symmetry. Informally, the view $T_{(G,\lambda)}(v)$ of a node v in a labelled graph (G, λ) is an infinite, labelled, rooted tree "downward locally isomorphic" to G, such that there exists a mapping from the vertices of the tree to the vertices of G that maps the root of the tree to v, the children of the root to the neighbors of v and recursively, the children of a node to the neighbors of that node. When no ambiguity arises, we shall denote a view $T_{(G,\lambda)}(v)$ simply by $T(v)$. For any integer $i \geqslant 0$, let $T^i(v)$ denote the i-view of node v, i.e., $T(v)$ truncated to distance i, where distance is defined in terms of edges, see Figure 15.

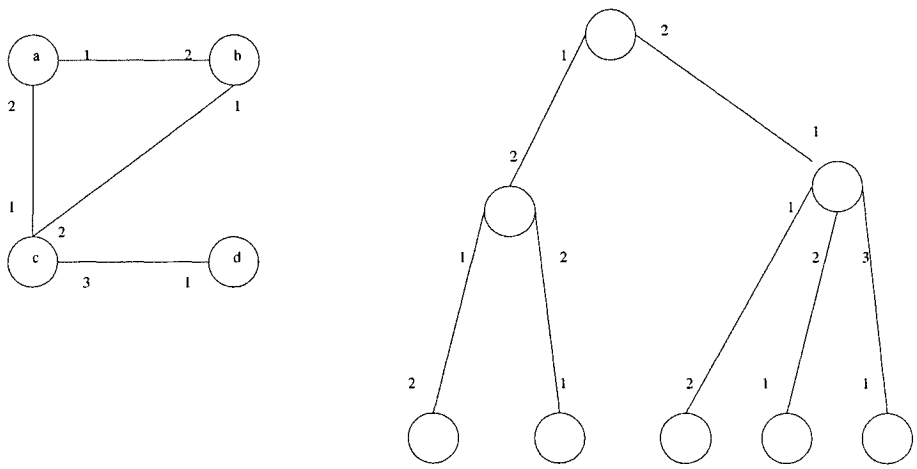

Figure 15: Example of a graph and its 2-view from node a.

More formally, the definition of the view is the following:

Definition 3.2.1. *(View)*

Given a labelled graph $(G = (V, E), \lambda)$, the i-th view of $v \in V$ is a rooted tree, $(T_i(v), \lambda')$ defined recursively as follows:

$$T_0(v) = (\{(v, \varepsilon, 0)\}, \emptyset),$$

$$T_i(v) = (V(T_{i-1}(v)) \cup X_i, E(T_{i-1}(v)) \cup Y_i),$$

where

$$X_i = \{(x, y, i) | (y, x) \in E(G) \quad and \quad \exists y' \quad such \quad that \quad (y, y', i-1) \in V(T_{i-1}(v))\},$$

$$Y_i = \{((x, y, i-1), (x', x, i)) | (x, y, i-1) \in V(T_{i-1}(v)) \quad and \quad (x', x, i) \in X_i\},$$

where the edges are labelled as follows:

$$\lambda'_{(x,y,i-1)}((x, y, i-1), (x', x, i)) = \lambda_x(x, x'),$$

and

$$\lambda'_{(x',x,i)}((x', x, i), (x, y, i-1)) = \lambda_{x'}(x', x).$$

Every vertex in $T_i(v)$ is of the form (x, y, i), where x corresponds to the vertex x in G. y corresponds to a neighbor of x in G and i is the distance of the node (x, y, i). Note that y is the father of x in the rooted tree. We include the distance to distinguish among the vertices of the tree.

We recall the notion of labelled graph isomorphism.

Definition 3.2.2. *(Labelled graph isomorphism)*

Let $(G = (V, E), \lambda)$ and $(G' = (V', E'), \lambda')$ be two labelled graphs. A labelled graph isomorphism (lg-isomorphism) of (G, λ) in (G', λ') is a bijection $\alpha : V \longrightarrow V'$ such that:

1. *$(\alpha(u), \alpha(v)) \in E'$ if and only if $(u, v) \in E$.*

2. *$\lambda_u(u, v) = \lambda'_{\alpha(u)}(\alpha(u), \alpha(v))$.*

An isomorphism from (G, λ) to (G, λ) is called an automorphism in (G, λ).

Definition 3.2.3. *(Same view)*

Two vertices u and v have the same view if and only if $\forall i \in \mathbb{N}$,

1. *there is a bijection $\phi : V(T_i(u), \lambda) \longrightarrow V(T_i(v), \lambda)$ such that: $\phi(u, \varepsilon, 0) = (v, \varepsilon, 0)$.*
2. *$(T_i(u), \lambda)$ and $(T_i(v), \lambda)$ are isomorphic.*

Let $\mathcal{L}$ be a finite set, and $\mathcal{L}^*$ be the set of strings of element of $\mathcal{L}$ including the empty string ε. In a system (G, λ) with local orientation, let $\overrightarrow{}_{(G,\lambda)} : V \times \mathcal{L}^* \to V$ be the partial function defined as follows: $\overrightarrow{}_{(G,\lambda)}(v, \alpha) = w \Leftrightarrow \exists \pi \in P[v, w] \wedge \Lambda_v(\pi) = \alpha$. The function $\overrightarrow{}_{(G,\lambda)}$ is well defined because, with local orientation, two paths starting from the same node are the same if and only if their labels are the same. When (G, λ) is clear from the context, we denote $\overrightarrow{}_{(G,\lambda)}$ with $\to$. Let $\mathcal{L}^d \subseteq \mathcal{L}^*$ be the set of strings of length at most d, and $\Lambda^d_{(G,\lambda)}[x, y] = \Lambda_{(G,\lambda)}[x, y] \cap \mathcal{L}^d$, and $\Lambda^d_{(G,\lambda)}[x] = \cup_{y \in V} \Lambda^d[x, y]$. For each $\alpha \in \Lambda^d_{(G,\lambda)}[u]$, let $[[\alpha]]^d_u = \Lambda^d_{(G,\lambda)}[u, u\overrightarrow{}_{(G,\lambda)}\alpha]$ (or $[[\alpha]]$ when u and d are given).

We now define the notion of surrounding and surrounding symmetric. Informally, the surrounding $N^d_{(G,\lambda)}(u)$ of a node u in (G, λ) is the labelled graph lg-isomorphic to G, where the lg-isomorphism $\mathcal{X}_u$ maps each node $v \in V$ to the set of strings $\Lambda[u, v]$ and u to the set of strings $\Lambda[u, u] \cup \{\varepsilon\}$, where ε is the empty string.

More formally, we define the notion of surrounding as follows,

Definition 3.2.4. *(Surrounding)*

Given a labelled graph $(G = (V, E), \lambda)$, an integer $d \geqslant 0$, and a node $u \in V$, the d-surrounding of u is the labelled graph $N^d_{(G,\lambda)}(u)$, where $\mathcal{V}(N^d(u))$, $\mathcal{E}((N^d(u))$, and $\mathcal{L}((N^d(u))$ denote the vertices, the edges and the labelling of $N^d(u)$, respectively, which are defined as follows:

1. *$\mathcal{V}(N^d(u)) = \{[[\alpha]]^d_u : \alpha \in \Lambda^d{}_{(G,\lambda)}[u]\}$.*
2. *Given α and β, $([[\alpha]]^d_u, [[\beta]]^d_u) \in \mathcal{E}(N(u))$ if and only if $e = (u \to \alpha, u \to \beta) \in E$ and $d_G(u, e) \leqslant d$. The edge e will be called the corresponding edge of $([[\alpha]]^d_u, [[\beta]]^d_u)$.*

3. $\mathcal{L}(N(u))_{[[\alpha]]}([[\alpha]],[[\beta]]) = \lambda_{u\to\alpha}(u \to \alpha, u \to \beta)$.

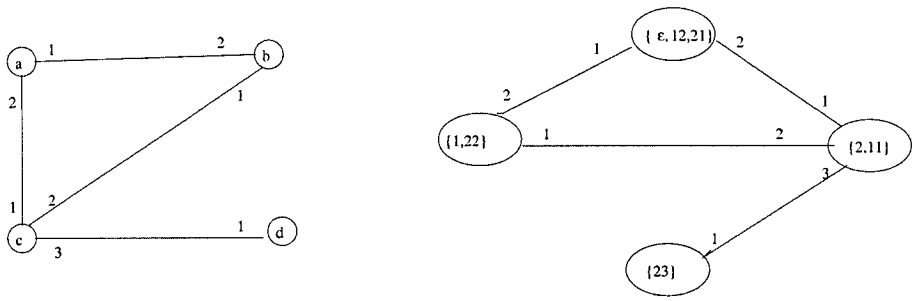

Figure 16: A labelled graph and its 2-surrounding from node a.

Definition 3.2.5. *(S_k-symmetric)*

A labelled graph (G,λ) is S_k-symmetric when there are k classes of nodes such that two nodes have the same surrounding if and only if they are in the same class. In particular, a graph is said to be surrounding symmetric if all the nodes have the same surrounding.

Definition 3.2.6. *(Labelled vertex transitive)*

A system $(G=(V,E),\lambda)$ is called labelled vertex transitive (L-vertex transitive) if and only if for every two nodes x,y in V there exists a labelled graph automorphism α such that $y=\alpha(x)$.

Definition 3.2.7. *(Labelled cycle symmetric)*

A system (G,λ) is called L-cycle symmetric if and only if all nodes of G belong to the same number of simple cycles with the same labelling.

For example in Figure 17, from nodes A, B, C, D, the simple cycles of length 3 are $[1,3,1]$, $[2,3,2]$, $[3,1,1]$,$[2,2,3]$,$[1,1,3]$, $[3,2,2]$. Then considering cycles of length 3, nodes A, B, C, D belong to the same number of simple cycles with the same labelling.

Proposition 3.2.1. *Given a system (G,λ), where λ is a local orientation, assume that we can label G using the minimal number of labels. If (G,λ) is L-cycle symmetric, then λ is symmetric.*

Proof. Assume (G,λ) is a L-cycle symmetric . For all the nodes of G, we have the same

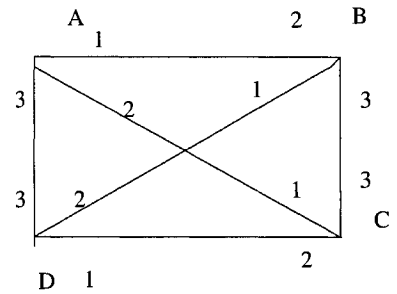

Figure 17: Example of L-Cycle Symmetric.

cycle labelling of length 2. In other words, for all $x, x_1 \in V$, $(a,b)_x$ if and only if $(a,b)_{x_1}$ where $a, b \in \mathcal{L}$. Since λ is a local orientation, we have:

$$\forall x, y, x_1, y_1 \in V, \quad \text{with} \quad (x,y) \in E, \quad (x_1,y_1) \in E$$

$$\lambda_x(x,y) = \lambda_{x_1}(x_1,y_1) \Longleftrightarrow \lambda_y(y,x) = \lambda_{y_1}(y_1,x_1).$$

Then, λ is symmetric. □

3.3 Relationship Between The Different Notions

In this section, we state the relationships that were already proven among some notions of unlabelled-unlabelled symmetries, unlabelled-labelled symmetries, and labelled-labelled symmetries. We then provide other relationships introduced in this thesis with their proofs.

It has been shown that:

- Vertex transitivity is necessary but not sufficient to have cycle symmetry [21, 25].
- (G, λ) is view symmetric if and only if G is regular and λ is both minimal and symmetric [17].
- View symmetry is a necessary but not a sufficient condition to have S_1-symmetry [18].
- Cayley graphs are vertex transitive, but not all vertex transitive graphs are Cayley graphs [1].

- If (G, λ) is a regular undirected graph with minimal symmetric labelling, then the following statements are equivalent [17]:

  1. (G, λ) has a weak sense of direction.
  2. (G, λ) has a sense of direction.
  3. (G, λ) is a Cayley graph with Cayley labelling.
  4. (G, λ) is S_1-symmetric.

- If (G, λ) is a regular directed graph with minimal labelling then the following two conditions are equivalent [11]:

  1. (G, λ) is Homonymous.
  2. (G, λ) is a Cayley graph with Cayley labelling.

The results in [11] are a bit more general than the ones in [17] because they apply to directed graphs. In [7], a full characterization of directed graphs with minimal sense of direction has been given. It was shown that a regular graph (G, λ) has a minimal sense of direction if and only if G is the graph of a semi group S that is a direct product of a group and a "Left-Zero" semigroup, and λ corresponds to the generators of S. The graph of a semi group is the graph where the nodes correspond to the elements of the semigroup and the edges correspond to the action of the generators. This characterization extends the one of [11] for homonymous sense of direction and the one of [17] for symmetric sense of direction in undirected graphs. In other words, it includes also directed graphs that are not homonymous extending [11], and in the case of undirected graphs, it includes the ones with no edge symmetry extending [17].

In the following, we establish new relationships among different notions mentioned before. Notice that, with a non symmetric labelling, L-cycle symmetric is not necessary for minimal sense of direction. In Figure 18, we have a system which is a sense of direction but not L-cycle symmetric. For example, the cycles labelling of length 2 from node A are: $[a, a]$,$[b, a]$, but from node c the cycles labelling of length 2 are: $[a, b]$,$[b, b]$.

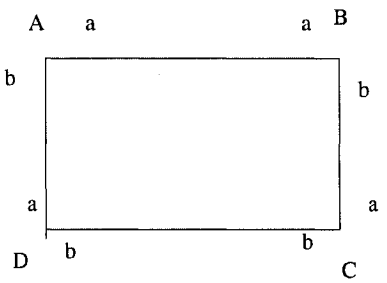

Figure 18: Example of non symmetric minimal sense of direction.

Theorem 3.3.1. *Given a system (G, λ) where λ is a symmetric minimal labelling, and local orientation. (G, λ) has a sense of direction if and only if (G, λ) is L-cycle symmetric.*

Proof.

$\Longrightarrow$ By contradiction: assume that (G, λ) is not L-cycle symmetric, and suppose that (G, λ) has a minimal sense of direction.

Since (G, λ) is not L-cycle symmetric, there are two nodes x and y in G such that x has a cycle labelling $[\lambda_x(x, x_1), \lambda_{x_1}(x_1, x_2), ..., \lambda_{x_n}(x_n, x)]$ but y does not have this cycle labelling.

Let $\lambda_x(x, x_1) = a$ and $\Lambda_x((x, x_n), (x_n, x_{n-1}), ..., (x_2, x_1)) = \alpha$. Since (G, λ) has a minimal sense of direction and the labelling is symmetric, we have that :

$$c(a) = c(\alpha), \tag{1}$$

where c is the coding function.

Since the graph is d-regular and λ is minimal, there exists an edge (y, y_1) such that $\lambda_y(y, y_1) = a$, and a path π starting from y, such that $\Lambda_y(\pi) = \alpha$. But since y does not have the cycle labelling $[\lambda_x(x, x_1), \lambda_{x_1}(x_1, x_2), ..., \lambda_{x_n}(x_n, x)]$, it must be : $c(a) \neq c(\alpha)$, contradicting (1).

$\Longleftarrow$ Proving that (G, λ) has a sense of direction is the same as proving that (G, λ) is S_1-symmetric since the two notions: sense of direction and S_1-symmetric are equivalent. We need to show that:

$$[[\alpha]]_u = [[\alpha]]_v \text{ for all } u, v \in V, \alpha \in \mathcal{L}^*.$$

Let $\beta \in [[\alpha]]_u$, that is, β is a path labelling from u to $u \longrightarrow \alpha$. Then, $\beta.\alpha^{-1}$ is a cycle at the node u. Since (G, λ) is L-cycle symmetric, $\beta.\alpha^{-1}$ is a cycle also at the node v. Then, $(\beta.\alpha^{-1}).\alpha$ is a path from v to $v \longrightarrow \alpha$, which means that $\beta \in [[\alpha]]_v$. Hence, $[[\alpha]]_u \subseteq [[\alpha]]_v$. Similarly, $[[\alpha]]_v \subseteq [[\alpha]]_u$. It follows that: $[[\alpha]]_u = [[\alpha]]_v$ for all $u, v \in V, \alpha \in \mathcal{L}^*$. Therefore, λ is a sense of direction. □

Notice that the property of L-cycle symmetric can be used to test if a regular graph with a minimal number of labels is a sense of direction. For example, the labelling of Peterson graph of Figure 19 is not a sense of direction. In fact, if we start from node A and follow the path labelling $[2, 3, 1, 3, 2]$, we get a cycle (see Figure 19). But if we follow the same path from node G, we do not get a cycle. Then, the property of L-cycle symmetric is violated.

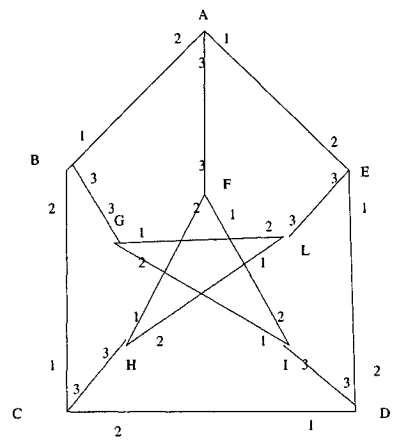

Figure 19: Verifying sense of direction using L-Cycle Symmetric properties.

In the following theorem, we present the relationship between L-vertex transitive and S_1-symmetric. First we present some lemmas and then proceed to the theorem.

Lemma 3.3.2. *If (G, λ) is L-vertex transitive then $\forall u, v \in V$,*

$$\mathcal{V}(N(u)) = \mathcal{V}(N(v)).$$

Proof. Since $(G = (V, E), \lambda)$ is L-vertex transitive, there exists an lg-automorphism φ of G such that $\forall u, v \in V, \quad \varphi(u) = v$. Let $\Lambda[u, u'] \in \mathcal{V}(N(u))$. Since φ is a lg-automorphism,

$\Lambda[u,u'] = \Lambda[\varphi(u),\varphi(u')]$, and since $\varphi(u) = v$, it follows that $\Lambda[\varphi(u),\varphi(u')] = \Lambda[v,\varphi(u')] \in \mathcal{V}(N(v))$. Hence, $\Lambda[u,u'] \in \mathcal{V}(N(v))$.

On the other hand, $\Lambda[u,u] \cup \{\varepsilon\} = \Lambda[\varphi(u),\varphi(u)] \cup \{\varepsilon\} = \Lambda[v,v] \cup \{\varepsilon\} \in \mathcal{V}(N(v))$. We conclude that $\mathcal{V}(N(u)) \subseteq \mathcal{V}(N(v))$. Similarly it can be shown that: $\mathcal{V}(N(v)) \subseteq \mathcal{V}(N(u))$. Thus $\mathcal{V}(N(u)) = \mathcal{V}(N(v))$ □

Lemma 3.3.3. *If (G,λ) is L-vertex transitive, then $\forall u, v \in V$,*

$$\mathcal{E}(N(u)) = \mathcal{E}(N(v)).$$

Proof. Since $(G = (V,E),\lambda)$ is L-vertex transitive, there exists a lg-automorphism φ of G such that $\forall u, v \in V,\ \ \varphi(u) = v$. Let $(\Lambda[u,u'],\Lambda[u,u''])$ in $\mathcal{E}(N(u))$. From the definition of surrounding we have: $(u',u'') \in E$. Since φ is a lg-isomorphism, we have $(\varphi(u'),\varphi(u'')) \in E$. From the definition of surrounding, we conclude that $(\Lambda[v,\varphi(u')],\Lambda[v,\varphi(u'')]) \in \mathcal{E}(N(v))$. Since $\varphi(u) = v$, $(\Lambda[\varphi(u),\varphi(u')],\Lambda[\varphi(u),\varphi(u'')]) \in \mathcal{E}(N(v))$. Hence, $(\Lambda[u,u'],\Lambda[u,u'']) \in \mathcal{E}(N(v))$ because φ is lg-isomorphism. Finally, $\mathcal{E}(N(u)) \subseteq \mathcal{E}(N(v))$.

Similarly, it can be shown that: $\mathcal{E}(N(v)) \subseteq \mathcal{E}(N(u))$, and thus, $\mathcal{E}(N(u)) = \mathcal{E}(N(v))$. □

Lemma 3.3.4. *If (G,λ) is L-vertex transitive, then:*

$\forall u, v, u', u'' \in V$ such that $\lambda, \lambda', \lambda''$ are respectively the labelling of G, $N(u)$, $N(v)$,

$$\lambda'_{\Lambda[u,u']}(\Lambda[u,u'],\Lambda[u,u'']) = \lambda''_{\Lambda[u,u']}(\Lambda[u,u'],\Lambda[u,u'']).$$

Proof. Since $(G = (V,E),\lambda)$ is L-vertex transitive, there exists a lg-automorphism φ of G such that $\forall u, v \in V\ \ \varphi(u) = v$.

Let $\lambda'_{\Lambda[u,u']}(\Lambda[u,u'],\Lambda[u,u'']) \in \mathcal{L}(N(u))$. From the definition of surrounding, we have $\lambda'_{\Lambda[u,u']}(\Lambda[u,u'],\Lambda[u,u'']) = \lambda_{u'}(u',u'')$. Moreover, because φ is a lg-automorphism, we have $\lambda_{u'}(u',u'') = \lambda_{\varphi(u')}(\varphi(u'),\varphi(u''))$

$= \lambda''_{\Lambda[v,\varphi(u')]}(\Lambda[v,\varphi(u')],\Lambda[v,\varphi(u'')]) \in \mathcal{L}(N(v))$.

$= \lambda''_{\Lambda[\varphi(u),\varphi(u')]}(\Lambda[\varphi(u),\varphi(u')],\Lambda[\varphi(u),\varphi(u'')])$

$= \lambda''_{\Lambda[u,u']}(\Lambda[u,u'], \Lambda[u,u''])$

Thus: $\lambda'_{\Lambda[u,u']}(\Lambda[u,u'], \Lambda[u,u'']) = \lambda''_{\Lambda[u,u']}(\Lambda[u,u'], \Lambda[u,u''])$ □

Theorem 3.3.5. *(G,λ) is L-vertex transitive if and only if (G,λ) is S_1-symmetric.*

Proof.

$\Longleftarrow$ Assume that (G,λ) is S_1-symmetric. Let u, v be two arbitrary vertices in G. We construct a labelled graph automorphism φ of (G,λ) such that $\varphi(u) = v$.

Since (G,λ) is S_1-symmetric, we have $N(u)^d = N(v)^d$ for any distance d. Let $\mathcal{X}_u$ and $\mathcal{X}_v$ be two labelled graph isomorphisms such that $\mathcal{X}_u$ maps u' to the set of strings $\Lambda[u,u']$ and maps u to the set of strings $\Lambda[u,u] \cup \varepsilon$. Similarly, $\mathcal{X}_v$ maps v' to $\Lambda[v,v']$ and v to $\Lambda[v,v] \cup \varepsilon$. We have:

$\mathcal{X}_u(u) = \{\Lambda[u,u] \cup \{\varepsilon\}\}$.

$\mathcal{X}_u(u') = \{\Lambda[u,u']\}$ for $u' \neq u$.

$\mathcal{X}_v(v) = \{\Lambda[v,v] \cup \{\varepsilon\}\}$.

$\mathcal{X}_v(v') = \{\Lambda[v,v']\}$ for $v' \neq v$.

Let $\varphi = \mathcal{X}_v^{-1}\mathcal{X}_u$. Since $\mathcal{X}_u$ and $\mathcal{X}_v$ are lg-isomorphisms of G, φ is a lg-automorphism of G. Since $\mathcal{X}_u(u)$ and $\mathcal{X}_v(v)$ are the only sets that contain ε, and from the assumption, $N(u)^d = N(v)^d$, we conclude that $\mathcal{X}_u(u) = \mathcal{X}_v(v)$. Hence, $\varphi(u) = v$.

$\Longrightarrow$ Assume that (G,λ) is L-vertex transitive. Let $u, v \in V$ be two arbitrary vertices. We need to show that $N(u)^d = N(v)^d$ for any distance d. To this end, we have to show the three following conditions:

1. $\mathcal{V}(N(u)) = \mathcal{V}(N(v))$.
2. $\mathcal{E}(N(u)) = \mathcal{E}(N(v))$.
3. $\forall u', u'' \in V$ such that $\lambda, \lambda', \lambda''$ are respectively the labelling of G, $N(u)$, $N(v)$, we have:

$$\lambda'_{\Lambda[u,u']}(\Lambda[u,u'], \Lambda[u,u'']) = \lambda''_{\Lambda[u,u']}(\Lambda[u,u'], \Lambda[u,u'']).$$

However, these are direct consequences of Lemma 3.3.2, Lemma 3.3.3, and Lemma 3.3.4, it follows that: $N(u) = N(v)$. □

We summarize the relationship among different notions seen before by the diagram of Figure 20. The bolded relations represent our contribution to the symmetric sense of direction.

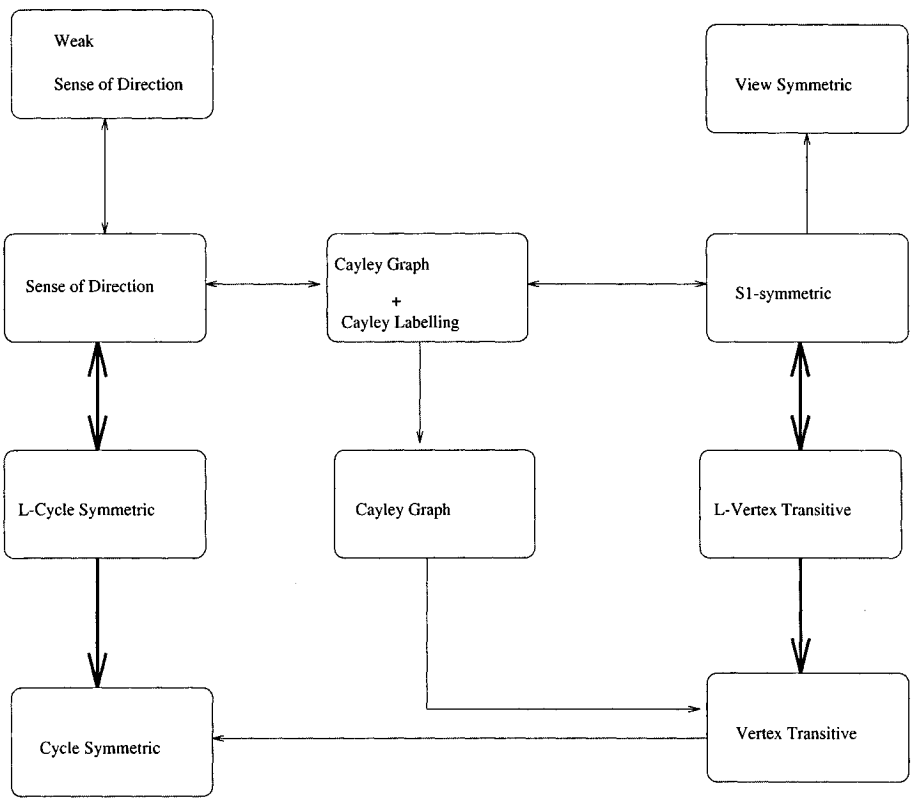

Figure 20: The relationship among different notions.

CHAPTER IV

ASYMMETRIC MINIMAL SENSE OF DIRECTION IN REGULAR GRAPHS

In this chapter, we investigate the properties required to have a sense of direction in regular graphs using the minimal number of labels, when the labelling is not symmetric. First we introduce some definitions so that we can use them to establish some propositions and theorems. We define symmetric, asymmetric, and fully asymmetric labelling. After that, we show that if we have a minimal sense of direction, then the labelling should be either symmetric or fully asymmetric. The fact that λ is either symmetric or fully asymmetric allows us to establish more properties for the case of asymmetric labelling. We show that the nodes of the graph can be divided into equivalent classes of the same size where the nodes belonging to the same equivalent class are incident on the same pairs of edge labelling, and nodes belonging to different equivalent classes do not share any edge labelling.

We apply the properties found on this chapter to prove that in the class of the ring, the ring of size four is the only graph who has asymmetric minimal sense of direction. We also show that the hypercube, the torus, and the complete graphs do not have an asymmetric minimal sense of direction. For the class of bipartite graphs, we show that complete bipartite graphs can have asymmetric minimal sense of direction. We also show how to label them to obtain an asymmetric minimal sense of direction.

4.1 Definitions and Propositions

As noted in the previous chapter, the result of [7] provides a full characterization of graphs with minimal sense of direction (including the case of undirected graphs without edge symmetry). The aim of this chapter is to study asymmetric minimal sense of direction in undirected graphs trying to provide a simpler characterization.

Definition 4.1.1. *(Symmetric label)*

A single label $a \in \mathcal{L}$ is symmetric when there exists $b \in \mathcal{L}$ such that:

$\forall x, y \in V$, *with* $(x, y) \in E$

$$\lambda_x(x, y) = a \Longleftrightarrow \lambda_y(y, x) = b.$$

Definition 4.1.2. *(Asymmetric label)*

Given a system (G, λ), a single label a is called asymmetric when :

$\exists\ x, y, x_1, y_1 \in V$, *with* $(x, y) \in E$, $(x_1, y_1) \in E$ *where:*

$$\lambda_x(x, y) = \lambda_{x_1}(x_1, y_1) = a \text{ but } \lambda_y(y, x) \neq \lambda_{y_1}(y_1, x_1).$$

Definition 4.1.3. *(Asymmetric labelling)*

Given a system (G, λ), the labelling λ is called asymmetric when:

$\exists\ x, y, x_1, y_1 \in V$, *with* $(x, y) \in E$, $(x_1, y_1) \in E$ *where:*

$$\lambda_x(x, y) = \lambda_{x_1}(x_1, y_1) \text{ but } \lambda_y(y, x) \neq \lambda_{y_1}(y_1, x_1).$$

In other words, a labelling λ is asymmetric if it uses at least one asymmetric label.

Definition 4.1.4. *(Fully asymmetric labelling)*

Given a system (G, λ), the labelling λ is called fully asymmetric when:

$\forall x, y \in V$ *such that* $(x, y) \in E$, *if* $\lambda_x(x, y) = a$, $\lambda_y(y, x) = b$, *then:*

$$\exists (x_1, y_1) \in E \text{ such that } \lambda_{x_1}(x_1, y_1) = a \text{ and } \lambda_{y_1}(y_1, x_1) \neq b.$$

In other words, a labelling is fully asymmetric if all labels are asymmetric

The following theorem states that symmetric or fully asymmetric labelling account for all sense of direction with the minimal number of labels in a regular graph.

Theorem 4.1.1. *Given a system (G, λ), if λ has a sense of direction, then λ is either symmetric or fully asymmetric.*

Proof. By contradiction, assume that there is a minimal sense of direction λ where some labels are symmetric and others are not symmetric.

Let a be a symmetric label and let b be its symmetry. Let e be a non symmetric label. Then, there are nodes x, y, z, t, such that $\lambda_x(x,y) = e$, $\lambda_y(y,x) = f$, $\lambda_z(z,t) = e$, $\lambda_t(t,z) = g$, and $f \neq g$.

From node x, we have $c([a,b]) = c([e,f])$ because G is regular, and λ is a minimal sense of direction. From node z, we have $c([a,b]) = c([e,g])$. Then, $c([a,b]) = c([e,f]) = c([e,g])$. But since $f \neq g$, we have a contradiction with the fact that λ is a sense of direction. □

As an example, consider Figure 21. The labelling uses both symmetric and not symmetric labels, thus the graph does not have a sense of direction.

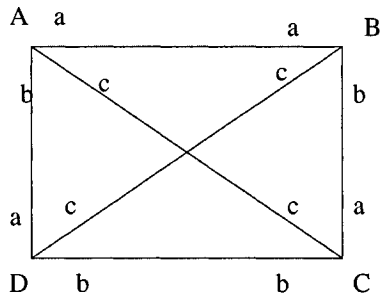

Figure 21: Example of a mixture of symmetric and asymmetric labels.

The following proposition says that if we take two different nodes x, y we have two cases: either the labels incident on x and the ones incident on y are identical, or they are all distinct.

Proposition 4.1.2. *Given a system (G, λ), where λ is an asymmetric minimal sense of direction, let $\{(a_1,b_1)_x, ..., (a_d,b_d)_x\}$, $\{(e_1,f_1)_y, ..., (e_d,f_d)_y\}$ be the set of pairs of labels incident on x and y, where d is the degree of the graph and $x, y \in V$, then we have one of the following:*

1. $\{(a_1,b_1)_x, ..., (a_d,b_d)_x\} = \{(e_1,f_1)_y, ..., (e_d,f_d)_y\}$
2. $\forall (a_i,b_i)_x \in \{(a_1,b_1)_x, ..., (a_d,b_d)_x\}$, $\forall (e_j,f_j)_y \in \{(e_1,f_1)_y, ..., (e_d,f_d)_y\}$, $i, j \in [1,d]$

$$(a_i, b_i)_x \neq (e_j, f_j)_y$$

Proof. We show that if there are $i, j \in [1, d]$ such that: $(a_i, b_i)_x = (e_j, f_j)_y$, then

$\{(a_1, b_1)_x, ..., (a_d, b_d)_x\} = \{(e_1, f_1)_y, ..., (e_d, f_d)_y\}$.

Assume that there are $i, j \in [1, d]$ such that: $(a_i, b_i)_x = (e_j, f_j)_y$.

Since we have a sense of direction, the following equalities hold:

$c([a_1, b_1]) = c([a_2, b_2]) = ... = c([a_d, b_d])$.

$c([e_1, f_1]) = c([e_2, f_2]) = ... = c([e_d, f_d])$.

$c([a_i, b_i]) = c([e_j, f_j])$ (because $(a_i, b_i) = (e_j, f_j)$).

Hence

$c([a_1, b_1]) = c([a_2, b_2]) = ... = c([a_d, b_d]) = c([e_1, f_1]) = c([e_2, f_2]) = ... = c([e_d, f_d])$.

And since the two sets $\{a_1, ..., a_d\}, \{e_1, ..., e_d\}$ are equal (because we have a regular graph with the minimal labelling), and there is a sense of direction, we conclude that:

$$\{(a_1, b_1)_x, ..., (a_d, b_d)_x\} = \{(e_1, f_1)_y, ..., (e_d, f_d)_y\}$$

. □

We now define a relation $\mathcal{R}$ between the nodes of (G, λ):

For all $x, y \in V$, define $x\mathcal{R}y$ if and only if x and y are incident on the same set of pairs of edges:

$$\{(a_1, b_1)_x, ..., (a_d, b_d)_x\} = \{(e_1, f_1)_y, ..., (e_d, f_d)_y\}.$$

The relation $\mathcal{R}$ is an equivalence relation because it is clearly reflexive, symmetric, and transitive.

We can then divide the nodes of G into equivalence classes where each equivalent class contains the nodes with the same set of incident pairs of labels.

Theorem 4.1.3. *Given a system (G, λ), if λ is an asymmetric minimal sense of direction, then G is cycle symmetric*

Proof. It comes from the proof that any minimal sense of direction is cycle symmetric[21]. □

In the following, we give a necessary condition for λ to be an asymmetric sense of direction.

Lemma 4.1.4. *If a system (G, λ) is an asymmetric sense of direction then:*

$\forall x \in V$, $\exists y \in V$ *such that there exist d different paths of length 2 between x and y.*

Proof. Let $x \in V$. In order to get an asymmetric minimal sense of direction, we should have at least two equivalence classes because otherwise, with one class, we obtain a symmetric labelling. Moreover, Proposition 4.1.2 states that all the pairs of labels incident on nodes belonging to different classes should be distinct. Without loss of generality, let C_1 and C_2 be two different equivalence classes in G such that x_1 is an element of C_1, and x is an element of C_2. Let $\{(a_1, b_1)_{x_1}, (a_2, b_2)_{x_1}, ..., (a_d, b_d)_{x_1}\}$ be the set of labels incident on x_1, where d is the degree of G. If we have a sense of direction, the condition $c([a_1, b_1]) = c([a_2, b_2]) = ... = c([a_d, b_d])$ should hold if we start from any node of the graph. In particular, it should be verified for x. If we look at the path labelling $[a_1, b_1], [a_2, b_2], ..., [a_d, b_d]$ starting from x, we see that they do not end-up in x otherwise x will belong to C_1. Moreover, $\{(a_1, b_1)_x, (a_2, b_2)_x, ..., (a_d, b_d)_x\}$ are d different paths. Hence, there exists a node $y \neq x$ where all those paths end. □

Lemma 4.1.5. *A necessary condition to have an asymmetric minimal sense of direction in a labelled graph G that has k equivalence classes is the following:*

$\forall x \in V$, $\exists (k-1)$ *nodes* $y \in V$ *such that there exist d different paths of length 2 between x and y.*

Proof. To have an asymmetric minimal sense of direction, the nodes should be divided into equivalent classes such that the nodes that belong to different equivalent classes don't have

any common incident pair of labels, and the nodes that belong to the same equivalence class have the same set of pairs of labels (Proposition 4.1.2).

Let $C_1, C_2, ..., C_k$ be all the distinct equivalence classes of G, and let $x_1, x_2, ..., x_k$ be elements of $C_1, C_2, ..., C_k$ respectively. Let $\{(a_{1i}, b_{1i})_{x_i}, (a_{2i}, b_{2i})_{x_i}, ..., (a_{di}, b_{di})_{x_i}\}$ be the set of labels incident on a node x_i where $x_i \in \{x_1, .., x_k\}$. If we have a sense of direction, we should have the condition $c([a_{1i}, b_{1i}]) = c([a_{2i}, b_{2i}]) = ... = c([a_{di}, b_{di}])$ to be verified if we start from any node of the graph. Moreover, starting from the same node x_i, the set of path labelling $\{[a_{2i}, b_{2i}], ..., [a_{di}, b_{di}]\}$ for different value of i have to end in different nodes. Then, there should exist $k-1$ nodes that can be reached from x using d paths of length 2. □

Theorem 4.1.6. *All the equivalence classes that we obtain with asymmetric minimal sense of direction labelling have the same number of elements.*

Proof. Let C_i and C_j be two arbitrary classes that we have obtained with asymmetric minimal sense of direction labelling. Let $x \in C_i$, $y \in C_j$, and assume that y can be reached from x using the path labelling $[a_1, a_2, ..., a_l]$ (see Figure 22). Then, from any $x' \in C_i$, there is a $y' \in C_j$ that can be reached using the same path labelling and starting from x'. We have that $y' \neq y$, because otherwise, there exists a node z in this path that violates local orientation (z would be incident on two different edges with the same pair labelling $(b_i, a_i)_z$). Then, $|C_i| \leqslant |C_j|$, and since the same argument can be applied if we start from any $y' \in C_j$ and use a path labelling to reach a node from C_i, we have $|C_i| = |C_j|$. Hence, all equivalence classes have the same size. □

The following lemmas state some necessary conditions that should be verified to have an asymmetric minimal sense of direction.

Lemma 4.1.7. *In a system (G, λ) where λ is an asymmetric minimal sense of direction, the size of each equivalence class is greater than or equal to 2.*

Proof. In a d-regular graph, assume that we have the labels $\{a_1, a_2, ...a_d\}$ to label this graph and obtain an asymmetric minimal sense of direction. The different pairs of labels that we

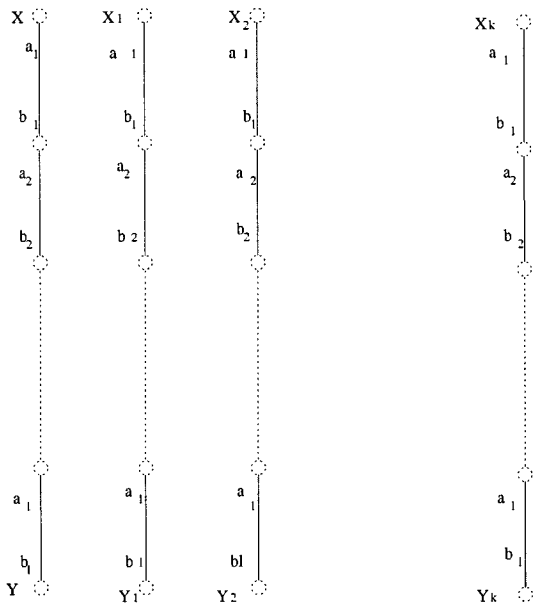

Figure 22: All equivalence classes have the same size.

can use are $\{(a_1, a_1), (a_2, a_1), .., (a_d, a_1), (a_1, a_2), (a_2, a_2), ..., (a_d, a_d)\}$; this set has size $d \times d$. If the size of each equivalence class is one, then we will need to use different pairs of labels to label each node because of Theorem 4.1.2. And since each node is incident on d different pairs of labels, we will need to have $n \times d$ labels. Since the number of pairs of labels needed are more than the available number of pairs of labels, the size of each equivalence class should be greater than 1. □

Lemma 4.1.8. *If the system (G, λ) is an asymmetric minimal sense of direction, then the number of classes, NbClasses, satisfies:*

$$NbClasses \leqslant \min(n/2, d)$$

Proof. The number of different pairs of labels that we need is $Nbclasses \times d$ (d is the degree of G) because each node is incident on d different pairs of labels and the nodes belonging to different classes should be incident on different pairs of labels. On the other hand, the number of labels that we have is d^2 because the set of pairs of labels that we have is

$\{(1,1), (2,1), ..., (d,1), (1,2), (2,2), ..(d,2), .., (d,1), ..., (d,d)\}$.

Thus, we must have

$Nbclasses \times d \leqslant d^2$, which yield: $Nbclasses \leqslant d$. On the other hand, we have $d < n$ and $n = Nbclasses \times size$, where $size$ is the size of one class, and where $Nbclasses$ and $size$ are integers. Thus, $NbClasses \leqslant n/2$. We conclude that: $NbClasses \leqslant \min(n/2, d)$.□

Theorem 4.1.9. *For a system (G, λ), if λ is an asymmetric minimal sense of direction, then the number of nodes in G is not a prime number.*

Proof. In a system (G, λ) where λ is an asymmetric minimal sense of direction, the nodes can be divided into at least two equivalence classes. Let us denote by NbClasses the number of equivalence classes in (G, λ), and Size be the size of each equivalence class (all the equivalence classes have the same size "Size").

The number of nodes n can be expressed as follows:

$$n = NbClasses \times Size$$

where:

$2 \leqslant NbClasses \leqslant n/2$ and $2 \leqslant Size \leqslant n/2$.

Hence, n should not be a prime number. □

4.2 Applications to Specific Topologies

4.2.1 Topologies Without Asymmetric Minimal Sense of Direction

Ring

For the class of the ring graphs, the only asymmetric minimal sense of direction is the ring of size 4. In fact, a system (G, λ) where G is a graph of two nodes cannot have an asymmetric labelling. If G has three nodes or more than 4 nodes, we cannot find a node y which can be reached from another node x using 2 different paths labelling of length 2. In this case, using Lemma 4.1.4, we cannot have an asymmetric minimal sense of direction. Assume that the labels used by G are a and b. If G has four nodes, the only way of

labelling G and getting an asymmetric sense of direction is by using the four pairs of labels $(a,a),(b,a),(b,b),(a,b)$ in this order.

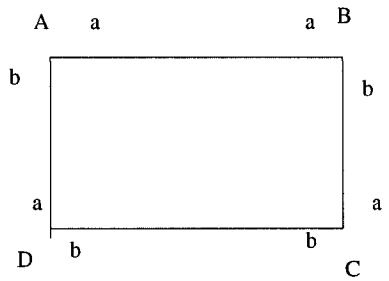

Figure 23: The only asymmetric sense of direction in the class of rings.

Hypercube

A hypercube of dimension n consists of 2^n vertices, labelled by n-bit binary strings, with edges between vertices whose labels differ in exactly one bit position. An example of a hypercube of dimension 3 is given in Figure 24.

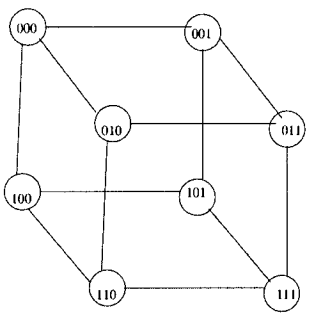

Figure 24: An example of a 3 dimensional hypercube.

In a system (G,λ) where G is a hypercube, G cannot be labelled so to obtain an asymmetric minimal sense of direction. By definition of a hypercube, between any two nodes x,y, there can be at most 2 different paths of length 2. Then, we cannot obtain d different paths between any two nodes x and y (d is the degree of the hypercube).

Torus

In a system (G,λ) where G is a Torus of dimension $M \times N$, $M > 1$ or $N > 1$, G cannot be labelled so to obtain an asymmetric minimal sense of direction. By definition of the

torus, between any two nodes x, y there can be at most 2 different paths of length 2. Then, we cannot obtain d different paths between any two nodes x and y (d is the degree of the torus).

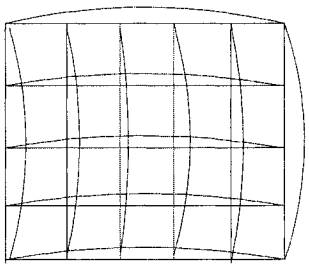

Figure 25: An example of a Torus.

Complete Graphs

A complete graph is a graph where every pair of nodes is connected by an edge. In a system (G, λ) where G is a complete graph, G cannot be labelled so to obtain an asymmetric minimal sense of direction. In fact, every node y can be reached from a node x using exactly $(d-1)$ different paths $P[x, y]$ of length 2 (where d is the degree of the graph). From Lemma 4.1.4, there is no way of labelling G and getting an asymmetric sense of direction .

4.2.2 A Topology With Asymmetric Minimal Sense of Direction

A bipartite graph is a set of graph vertices decomposed into two disjoint sets V_1 and V_2 such that no two vertices within the same set are adjacent. A graph is bipartite if and only if all its cycles are of even length. In the case where every vertex in V_1 is adjacent to every vertex in V_2, G is called a complete bipartite graph.

Theorem 4.2.1. *Given a system (G, λ), where G is a complete bipartite graph. (G, λ) has an asymmetric sense of direction.*

Proof. Let G be a bipartite graph with 2 sets of nodes A and B. We have $|A| = |B|$ because G is a d-regular bipartite graph. Let $A = \{x_1, x_2, .., x_d\}$, $B = \{x'_1, x'_2, .., x'_d\}$, and let $\lambda = \{l_1, l_2, .., l_d\}$ be the set of available labels. We label the edges of G in the following

way (See Figure 26):

$$\begin{cases} \lambda_{x_i}(x_i, x'_j) = l_j \\ \lambda_{x'_i}(x'_i, x_j) = l_j \end{cases}$$

for $1 \leqslant i \leqslant d, \quad 1 \leqslant j \leqslant d.$

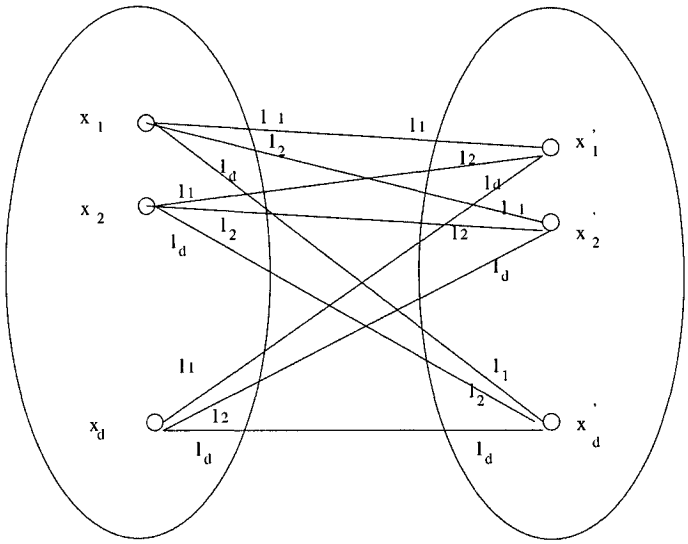

Figure 26: Labelling the bipartite graph using an asymmetric sense of direction.

We show that (G, λ) is a sense of direction.

Let $\Lambda_{y_1}(\pi) = [\lambda_{y_1}(y_1, y_2), ..., \lambda_{y_{m-1}}(y_{m-1}, y_m)]$ where $\pi = [(y_1, y_2), ..., (y_{m-1}, y_m)]$, and $\Lambda_{y_1}(\pi') = [\lambda_{y_1}(y_1, y'_2), ..., \lambda_{y'_{t-1}}(y'_{t-1}, y'_t)]$ where $\pi' = [(y_1, y'_2), ..., (y'_{t-1}, y'_t)]$.

We define the coding function $c : \mathcal{L}^* \to \{\mathcal{L} \cup (\mathcal{L} \times \mathcal{L})\}$ as follows:

$$c(\Lambda_{y_1}(\pi)) = \begin{cases} \lambda_{y_{m-1}}(y_{m-1}, y_m) & \text{if } \Lambda_{y_1}(\pi) \text{ has even length} \\ (\lambda_{y_{m-1}}(y_{m-1}, y_m), \lambda_{y_{m-1}}(y_{m-1}, y_m)) & \text{if } \Lambda_{y_1}(\pi) \text{ has odd length} \end{cases}$$

We prove that:

$$c(\Lambda_{y_1}(\pi)) = c(\Lambda_{y_1}(\pi')) \Longleftrightarrow y_m = y'_t.$$

In other words, we prove that different paths starting from the same node y_1 have the same coding function c if and only if they end up in the same node.

If $c(\Lambda_{y_1}(\pi)) = c(\Lambda_{y_1}(\pi'))$, we have one of the following situations:

1. y_m and y'_t are in A and $\lambda_{y_{m-1}}(y_{m-1}, y_m) = \lambda_{y'_{t-1}}(y'_{t-1}, y'_t)$. By construction of λ, we have $y_m = y'_t$

2. y_m and y'_m are in B and

$$(\lambda_{y_{m-1}}(y_{m-1}, y_m), \lambda_{y_{m-1}}(y_{m-1}, y_m)) = (\lambda_{y'_{t-1}}(y'_{t-1}, y'_t), \lambda_{y'_{t-1}}(y'_{t-1}, y'_t)).$$

   It follows that: $\lambda_{y_{m-1}}(y_{m-1}, y_m) = \lambda_{y'_{t-1}}(y'_{t-1}, y'_t)$. By construction of λ, we have $y_m = y'_t$

If $c(\Lambda_{y_1}(\pi)) \neq c(\Lambda_{y_1}(\pi'))$, we have one of the following situations:

1. y_m and y'_t are on the same set A or B. Since $c(\Lambda_{y_1}(\pi)) \neq c(\Lambda_{y_1}(\pi'))$, we have that: $\lambda_{y_{m-1}}(y_{m-1}, y_m) \neq \lambda_{y'_{t-1}}(y'_{t-1}, y'_t)$. By construction of λ, we conclude that $y_m \neq y'_t$.

2. y_m and y'_t are on different sets. In this case, $y_m \neq y'_t$.

□

Example 4.2.1. Figure 27 provides some examples of bipartite graphs that are asymmetric minimal sense of direction.

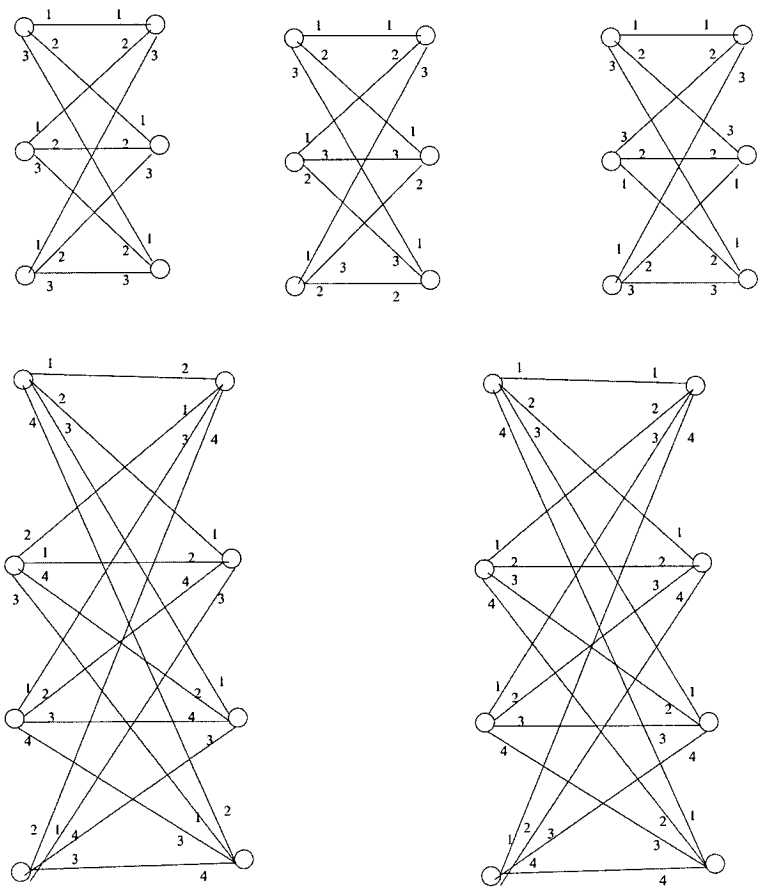

Figure 27: Examples of Asymmetric Sense of Direction.

CHAPTER V

MINIMUM CHORDAL LABELLING

In Section 2.2.2, we have defined the chordal sense of direction by fixing a cyclic ordering of the nodes and labelling each incident link by the distance in the above cycle. Thus, different cyclic ordering of the nodes give different chordal labelling. Figure 28 shows two different chordal labelling in which one uses 5 labels and the other uses 4 labels.

In this section, we solve the problem of minimum sense of direction for the class of chordal labelling by writing two programs that find the minimum number of labels needed in a given graph such that we have a chordal sense of direction. We will do so by using two methods:

- An exhaustive method: the backtrack algorithm
- A heuristic method: the genetic algorithm

The difference between the backtrack and the genetic algorithm is that the backtrack algorithm gives an exact solution to the problem but the genetic algorithm finds just a "close to" optimal solution. Thus, if we want to find a minimum chordal labelling in a small enough graph, the backtrack algorithm is the best choice. But once we start working with graphs with hundreds of nodes and edges, the backtrack algorithm cannot be used and the alternative is the genetic algorithm which finds a near optimal solution.

To solve the minimum chordal labelling in a graph G, we should find a cyclic ordering of the nodes of G that uses the minimum number of labels. The backtrack algorithm described in Section 5.1 tries exhaustively all the cyclic ordering of the nodes and reports the permutation that uses the minimum number of labels. The genetic algorithm described in Section 5.2 gives an approximate solution to the problem of minimum chordal sense of direction by choosing a cyclic ordering of the nodes at random.

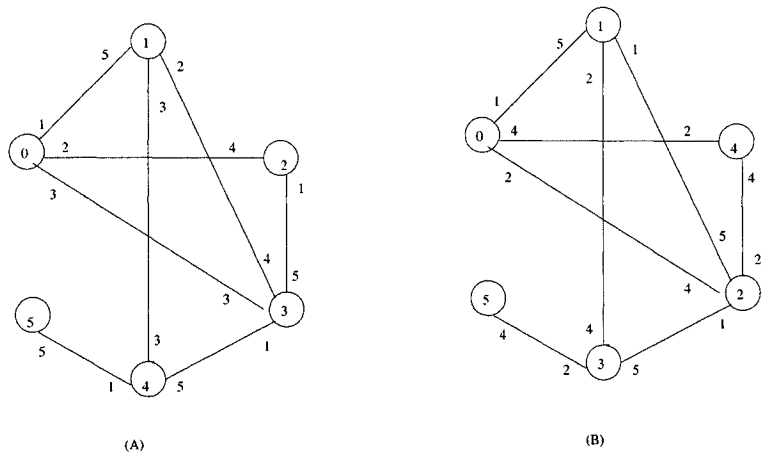

Figure 28: Different chordal labelling arise from different cyclic ordering of the nodes.

5.1 The Backtrack Algorithm

Let $G = (V, E)$ be a graph with $V = \{V_0, V_1, .., V_{n-1}\}$. A feasible solution for the minimum chordal sense of direction is a permutation

$$[lab(V_0), lab(V_1), ..., lab(V_{n-1})]$$

which defines a cyclic ordering of the nodes $\{V_0, V_1, ..., V_{n-1}\}$, and the objective is to minimize the number of labels used in the edges of G such that those labels are obtained by applying the difference of $lab(V_i)$ and $lab(V_j)$ modulo $|V|$ where $(V_i, V_j) \in E$

Any chordal labelling can be represented as a permutation. We use the backtrack algorithm to find the minimum chordal labelling by keeping track of the permutation of the nodes which gives the minimum number of labels needed for the edges.

We have a list of choices to label the nodes. Initially, we have $choiceSet = \{0, 1, 2, .., n-1\}$ where $n = |V|$. Every time we assign a label l to a vertex V_t at an iteration t, we remove l from *choiceSet*, and we calculate $(l - lab[V_i])$ (mod n) where $i < t$ and such that $(V_i, V_t) \in E$. The adjacency matrix facilitates the task of finding all those V_i. In the case where $(l - lab[V_i])$ (mod n) was used before, we do not add it to the list of used labels, but if it was not used before we add the two labels $(lab[V_l] - lab[V_i])$ (mod n) and $(lab[V_i] - lab[V_i])$ (mod n) if they are different and one of them if they are equal. After that, we call the backtrack algorithm with the new *choiceSet*. For pruning, we stop the

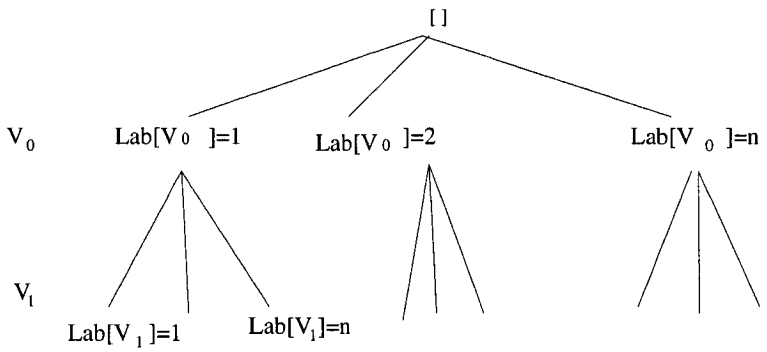

Figure 29: The backtrack algorithm.

algorithm if we find that the number of labels of the edges used in a permutation is equal to the degree of the graph because we know that this solution is an optimal solution. We will prune also if the number of labels of the edges found needed at an iteration l is greater than the best solution found so far. For the whole program, see Appendix A.

Example 5.1.1. Consider the graph of Figure 30. The adjacency matrix of this graph is represented in Table 2. In the written program, we enter the number of nodes in the graph and the adjacency matrix, and the program returns the minimum chordal labelling and a permutation of the nodes which gives this solution. For the example of Figure 30, the input and output is the following:

Give the number of nodes n and the adjacency matrix of the graph

8

1 4 7

0 2 5

1 3 6

2 4 6

0 3 5

1 4 7

2 3 7

0 5 6

The minimum number of labels needed is: 5

A permutation of the nodes which gives this solution is:

0 1 2 3 7 4 6 5

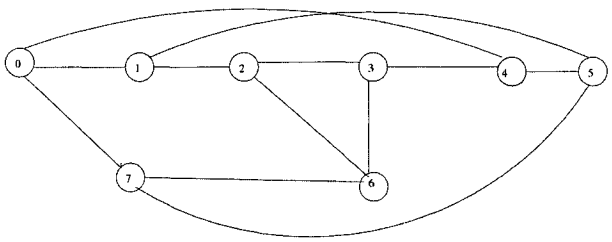

Figure 30: A graph in which we want to find the minimum sense of direction using the chordal labelling.

Vertex	Adjacent	Vertices	
0	1	4	7
1	0	2	5
2	1	3	6
3	2	4	6
4	0	3	5
5	1	4	7
6	2	3	7
7	0	5	6

Table 2: The adjacency matrix for Figure 30.

5.2 The Genetic Algorithm

In this section, we develop a genetic algorithm for the minimum chordal labelling problem.

In a genetic algorithm, we begin with an initial population of feasible solutions. Then, the feasible solutions from this initial population are mated (i.e., recombined in pairs) to produce children. After the children are obtained, some type of mutation are allowed to occur. Usually a mutation is a heuristic based on a neighborhood search. This produces the next generation of the population. The process can be iterated for as many generations as desired. A genetic algorithm must specify how children are produced . A common approach is to take two feasible solutions from the population("parents") , and use a recombination operation to generate two children, which inherits properties of the two parents. One simple recombination operation is called crossover. Suppose that we have an optimization

problem in which the universe $\mathcal{X} = \{0,1\}^n$, and the two parents are $W = [w_1, ..., w_n]$ and $X = [x_1, ..., x_n]$. Choose a crossover point $j \in \{1, ..., n\}$ at random. Then define $Y = [y_1, ..., y_n]$ and $Z = [z_1, ..., z_n]$ as follows:

$$y_i = \begin{cases} w_i & \text{if } 1 \leqslant i \leqslant j \\ x_i & \text{if } \quad j+1 \leqslant i \leqslant n \end{cases}$$

and

$$z_i = \begin{cases} x_i & \text{if } 1 \leqslant i \leqslant j \\ w_i & \text{if } j+1 \leqslant i \leqslant n \end{cases}$$

In other words, Y is formed from the first j entries of W and the last $n-j$ entries of X, and Z is formed from the first j entries of X and the last $n-j$ entries of W. But the crossover operation does not work for all the problems. For example, if we want to generate a permutation from two permutations as it is the case for the problem that we want to solve, crossover cannot be used since it will not produce a feasible solution.

For the minimum chordal labelling problem, a feasible solution is a permutation

$$[lab(v_0), lab(v_1), ..., lab(v_{n-1})]$$

and the objective is to minimize the number of labels used in the graph such that those labels are obtained by applying the difference modulo $|V|$ to $lab(v_i)$ and $lab(v_j)$ where $(v_i, v_j) \in E$

In order to design a genetic algorithm, we need recombination and mutation operations and a method to select an initial population.

We first describe a mutation operation, which will be a heuristic consisting of a sequence of neighborhood searches. Given a permutation $X = [x_0, x_1, ..., x_i, x_{i+1}, ..., x_j, x_{j+1}, ...x_{n-1}]$ we can cut two edges (x_i, x_{i+1}) and (x_j, x_{j+1}) and then reattach the ends of the two resulting paths to create a new permutation. We change our permutation if the number of labels

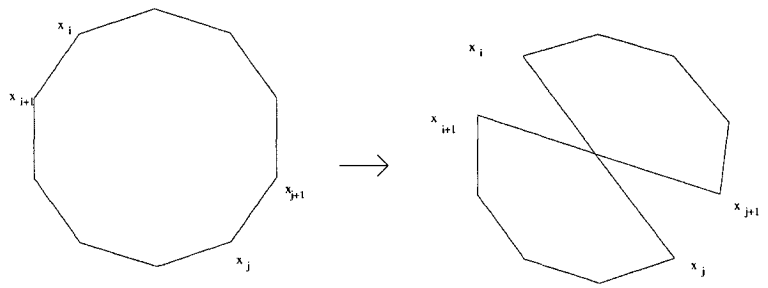

Figure 31: Two-opt moves.

used after the cut and reattach is less than the best we have so far keeping track of the minimum number of labels found so far. Let us call this operation Two-opt moves.

The mutation heuristic consists of a sequence of a Two-opt moves . We iteratively apply two-opt moves until no pair of edges can be found that yield a better solution.

For selecting the initial population, we generate the population size popSize at random each of which is a permutation of $(0, .., n-1)$, where n is the number of vertices of the graph.

For the recombination operations, instead of using the crossover operation that we have already mentioned and which is of no use for our problem, we use the following method: we select at random a length h and a random substring $S = [S_0, ..., S_{h-1}]$ of length h from one of the parents by selecting a starting location j. The string s is first copied to the beginning of a new child. The child is completed to a feasible solution by appending the nodes of the other parent that are not in S in the order in which they appear. The resulting child is then improved to a local minimum by applying the steepest ascent algorithm (the sequence of a two-opt moves). This is then repeated, with the roles of the two parents reversed, to generate a second child. For the whole program, see Appendix B.

Example 5.2.1. Consider the graph of Figure 30. The adjacency matrix of this graph is represented in Figure 2. In the written program, we enter the number of nodes in the graph and the adjacency matrix, and the program returns the minimum chordal labelling and a permutation of the nodes which gives this solution. For the example of Table 30, the input and output is the following:

Give the number of nodes n, the number of iterations, and the population size:

8

50

4

Enter the adjacency list :

1 4 7

0 2 5

1 3 6

2 4 6

0 3 5

1 4 7

2 3 7

0 5 6

The minimum number of labels needed is: 5

A permutation of the nodes which gives this solution is:

1 2 0 7 3 4 6 5

5.3 Examples and Observations

In this section, we test the backtrack and the genetic algorithms on some specific graphs, and provide some information about the running time of both programs, how many times we get an optimal solution in both algorithms, in which cases it is better to use the genetic algorithm and in which cases it is better to use the backtrack algorithm. We also provide some observations about the graphs under study.

In what follows, all the tests done on the genetic algorithm are out of 22 runs.

5.3.1 The Peterson Graph

The backtrack algorithm gives the exact solution. Table 3 reports the minimum chordal labelling needed using the backtrack algorithm and the running time of the program and

Figure 32 shows a permutation of the nodes labelling that gives the minimum chordal labelling.

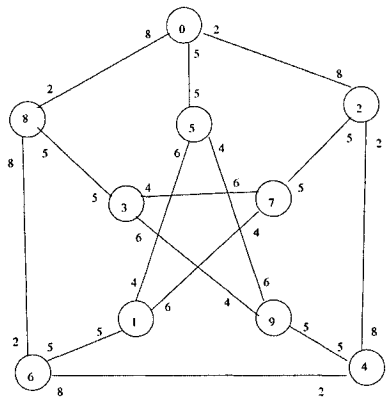

Figure 32: A permutation of the labels that gives the minimum chordal labelling in peterson graph.

optimal solution	average runtime(ms)
5	531

Table 3: The test of minimum chordal labelling using the backtrack algorithm on peterson graph.

The result of the test using the genetic algorithm out of 22 runs is given in Table 4. The population size *PopSize*, and the number of iterations *cmax* done on the genetic algorithm have effects on the solution found (number of labels needed) and on the running time of the program. For this reason, we run the program on different number of population size and different number of iterations. On Table 4, *Min* is the minimum number of labels found by the program, and *Max* is the maximum number of labels found. *Average* is the average number of labels reported, and *nb optimal solution* is the number of times the optimal solution was found out of the 22 runs. *runtime* is the average running time of the program.

The results found on peterson graph show that the backtrack algorithm is better to use than the genetic algorithm. In fact, the backtrack algorithm gives the exact solution and runs in a less time.

PopSize	Cmax	Min	Max	Average	nb optimal sol	runtime
4	50	5	6	5.18	18	78
	100	5	6	5.09	20	156
	200	5	6	5.04	21	297
8	50	5	6	5.09	20	158
	100	5	5	5	22	305
	200	5	5	5	22	578
16	50	5	5	5	22	297
	100	5	5	5	22	578
	200	5	5	5	22	1125
32	50	5	5	5	22	594
	100	5	5	5	22	1140
	200	5	5	5	22	2234

Table 4: The test of minimum chordal labelling using the genetic algorithm on peterson graph.

5.3.2 A Three Regular Graph

It was shown in [10] that the graph of Figure 30 requires 5 labels to have a weak sense of direction. The test done in [10] has used a complicated tool called *optwsod* that serves for making experimental research with weak sense of direction. The authors of [10] have reported that " there is no hope whatsoever to check this claim manually". In this section, we use the backtrack and the genetic algorithm and get not only a minimum chordal labelling, but also a minimum weak sense of direction for the graph of Figure 30.

Table 5 reports the minimum chordal labelling needed using the backtrack algorithm and the running time of the program. The result of the test using the genetic algorithm out of 22 runs is given in Table 6.

optimal solution	average runtime(ms)
5	94

Table 5: The test of minimum chordal labelling using the backtrack algorithm on the graph of Figure 30.

From the results found, both algorithms give the optimal solution all the time. Note from both tables, Table 5 and Table 6, that when the population size equals to 4 and the number of iterations is either 50 or 100, the running time of the genetic algorithm is better than that of the backtrack algorithm.

PopSize	Cmax	Min	Max	Average	nb optimal sol	runtime
4	50	5	5	5	22	47
	100	5	5	5	22	78
	200	5	5	5	22	156
8	50	5	5	5	22	78
	100	5	5	5	22	141
	200	5	5	5	22	296
16	50	5	5	5	22	172
	100	5	5	5	22	297
	200	5	5	5	22	563
32	50	5	5	5	22	296
	100	5	5	5	22	562
	200	5	5	5	22	1078

Table 6: The test of minimum chordal labelling using the genetic algorithm on the graph of Figure 30.

5.3.3 The Torus

The running time of the backtrack algorithm increases as we increase the size of the graph. For this reason, we run the backtrack and the genetic algorithms on different number of nodes for the class of torus graphs increasing the number of nodes used every time and see for which number of nodes the algorithm slows down considerably.

Let *dimension* denotes the dimension of the torus. The test of the backtrack algorithm on different dimensions of the torus is reported in Table 7. Note from the table that if the number of labels needed is equal to the degree of the graph, the backtrack algorithm takes less time because it does not need to go over all the possibilities, it needs just to find a permutation in which the number of labels needed is equal to the degree of the graph. Note also from the table that after using a torus that has 30 nodes and 60 edges (dimension

6×5), the program slows down considerably.

The result of the test using the genetic algorithm out of 22 runs is given in Table 8.

dimension	optimal solution	average runtime(ms)
2×3	3	0
3×3	6	360
4×3	4	125
4×4	6	234438
5×4	4	5750
5×5	6	1963922
6×5	4	80437
6×6	–	–

Table 7: The test of minimum chordal labelling using the backtrack algorithm on the torus.

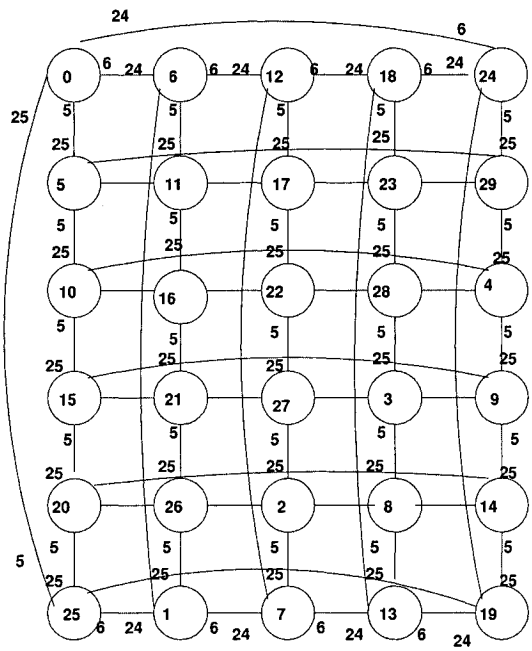

Figure 33: Labelling the tours of dimension 6×5 using 4 labels.

The results found on Table 7 provide an intuition about labelling a torus of dimension $m \times n$, where m and n are relatively prime, and getting a minimal chordal labelling. We can construct the labelling on the way shown in Figure 34. By construction, the labelling uses exactly 4 labels when $m > 2$ and $n > 2$.

dimension	PopSize	Cmax	Min	Max	Average	nb optimal sol	runtime
3×3	4	50	6	6	6	22	47
	4	200	6	6	6	22	281
	16	50	6	6	6	22	282
	16	200	5	6	6	22	1016
4×4	4	50	8	11	8.6	0	797
	4	200	8	9	8.4	0	3047
	16	50	8	9	8.3	0	3125
	16	200	6	9	8.1	2	12203
5×5	4	50	10	18	14.9	0	9013
	4	200	10	18	12.14	0	35462
	16	50	10	18	13.39	0	35682
	16	200	10	18	12.02	0	141414
6×5	4	50	18	22	20.69	0	7015
	4	200	16	22	20.01	0	27547
	16	50	18	22	21.1	0	27937
	16	200	14	18	17.8	0	111281

Table 8: The test of minimum chordal labelling using the genetic algorithm on the torus.

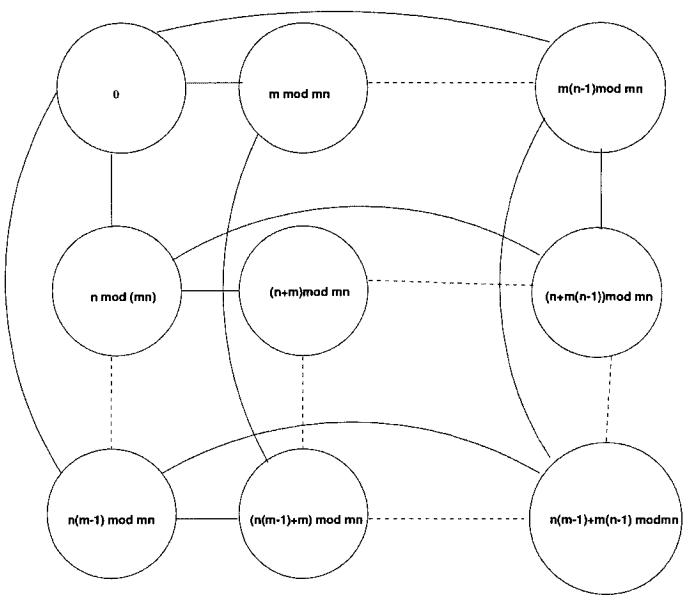

Figure 34: Constructing a minimal chordal labelling in a Torus of dimension $m \times n$ where $gcd(m, n) = 1$.

5.3.4 The Bipartite Graph

We test the backtrack and the genetic algorithm on some bipartite graphs. The test of the backtrack algorithm on the graph of Figure 35 is given in Table 9. The graph of Figure 35 is an interesting graph since it could have an asymmetric minimal sense of direction. It verifies all the conditions that we have already established in Chapter 4, especially the fact that starting from any node x, there exists another node y that can be reached from x using d paths of length 2 (d is the degree of the graph). Moreover, this graph is not a complete bipartite graph. The test done to find the minimum chordal labelling using the backtrack algorithm provides another information about this graph. In fact this graph is cycle symmetric since the minimum number of labels needed to have the chordal labelling is minimal.

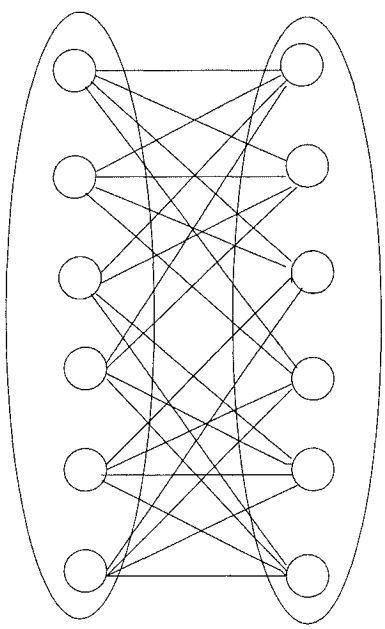

Figure 35: A bipartite graph.

optimal solution	average runtime(ms)
4	328

Table 9: The test of minimal chordal labelling using the backtrack algorithm on the graph of Figure 35.

The result of the test using the genetic algorithm out of 22 runs is given in Table 10.

PopSize	Cmax	Min	Max	Average	nb optimal sol	runtime
4	50	4	6	4.7	14	219
	100	4	6	5	11	406
	200	4	6	4.36	18	797
8	50	4	6	4.8	13	407
	100	4	4	4	22	797
	200	4	6	4.9	18	1547
16	50	4	4	4	22	813
	100	4	6	4.18	20	1578
	200	4	4	4	22	3094
32	50	4	4	4	22	1640
	100	4	4	4	22	3250
	200	4	4	4	22	7047

Table 10: The test of minimal chordal labelling using the genetic algorithm on the graph of Figure 35.

CHAPTER VI

CONCLUSION

In this thesis, we have settled two open problems about the relationship between labelled *cycle symmetry* and minimal sense of direction from one side, and labelled *vertex transitivity* and minimal sense of direction from the other side. We have also provided many new properties of minimal sense of direction using asymmetric labelling in regular graphs as well as several necessary conditions for the existence of such labelling.

A characterization for minimal sense of direction in directed graphs, thus including undirected graphs with asymmetric labelling, has been given in [7] (a directed graph has a minimal sense of direction if and only if it is a semigroup graph. However, this characterization does not describe the topological properties that are necessary and sufficient for having minimal sense of direction. In other words, it is not known what graphs are semigroup graphs. The problem that is still open is to characterize the *topology of graphs* having asymmetric sense of direction. We conjecture that the set of graphs that can be labelled with asymmetric minimal sense of direction coincides with a subset of bipartite graphs. An answer to this conjecture would also provide a characterization of semigroup graphs. In the thesis we have moved several steps toward this goal by giving several necessary conditions and showing that many known topologies cannot be labelled with an asymmetric minimal sense of direction in an undirected graph.

REFERENCES

[1] S.B. Akers et B. Krishamurthy. *A group-theoretic model for symmetric interconnection networks.* IEEE Transactions on Computers, **38(4)**:555-566, 1989.

[2] H. Attiya, J.van Leeuwen, N. Santoro, and S. Zaks. *Efficient elections in chordal ring networks.* Algorithmica, **4**:437-446, 1989.

[3] H. Attiya, M. Snir, and M.K. Warmuth. *Computing on an anonymous ring.* Journal of the A.C.M., **35(4)**:845-875, 1988.

[4] B. Awerbuch, O. Goldreich, D. Peleg, and R. Vainish. *A trade-of between information and communication in broadcast protocols.* Journal of the A.C.M., **37(2)**:238-256, 1990.

[5] G. V. Bochmann, P. Flocchini, and D. Ramazani. *Distributed objects with sense of direction.* In Proc. of the 1st IEEE Workshop on Distributed Data and Structures, 1-15, Orlando, 1998.

[6] G. V. Bochmann, P. Flocchini, and D. Ramazani. *Object naming and object composition.* Technical Report **1135**, University de Montreal, 1998.

[7] S. Foldes and J. Urrutia. *Sense of direction, semigroups, and cayley groups.* Manuscript, 1998.

[8] P. Boldi and S. Vigna. *Coverings that preserve sense of direction.* Information Processing Letters, **75(4)**:175-180, 2000.

[9] P. Boldi and S. Vigna. *On the complexity of deciding sense of direction.* SIAM Journal on Computing, **29(3)**:779-789, 2000.

[10] P. Boldi and S. Vigna. *Lower bounds for weak sense of direction.* In structure, Information and Communication Complexity. In Proc. of the 7th Colloquium SIROCCO 2000, Proceedings in Informatics. Carleton Scientific, 2000.

[11] P. Boldi and S. Vigna. *Minimal sense of direction and decision problems for cayley graphs.* Information Processing Letters, **64(6)**:299-303, 1997.

[12] P. Boldi and S. Vigna. *Computing vector functions on anonymous networks.* In Proc. of 4th International Colloquium on Structural Information and Communication Complexity, 201-214, 1997.

[13] P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. *Symmetry breaking in anonymous networks: Characterization.* In Proc. of 4th Israeli Symposium on Theory of Computing and Systems, 16-26, 1996.

[14] P. Boldi and S. Vigna. *On some constructions which preserve sense of direction.* In Proc. of 3th International Colloquium on Structural Information and Communication Complexity, 47-57, Siena, 1996.

[15] S. Dobrev. *Leader election using any sense of direction.* In Proc. of 6th International Colloquium on Structural Information and Communication Complexity, 93-104, Lacanau, 1999.

[16] S. Dobrev and P. Ruzicka. *Linear broadcasting and $n \log\log n$ election in unoriented hypercubes.* In Proc. of the 4th International Colloquium on Structural Information and Communication Complexity, 53-68, Ascona, 1997.

[17] P. Flocchini, A. Roncato, and N. Santoro. *Symmetries and sense of direction in labeled graphs.* Discrete Applied Mathematics, **87(1-3)**:99-115, 1998.

[18] P. Flocchini, A. Roncato, and N. Santoro. *Computing on anonymous networks with sense of direction.* Theoretical Computer Science, **301(1-3)**:355-379, 2003.

[19] P. Flocchini, B. Mans, N. Santoro. *Sense of direction in distributed computing.* Theoretical Computer Science, **291**:29-53, 2003.

[20] P. Flocchini, B. Mans, and N. Santoro. *Sense of direction: definition, properties and classes.* Networks, **32(3)**:165-180, 1998.

[21] P. Flocchini. *Minimal sense of direction in regular networks.* Information Processing Letters, **61(6)**:331-339, 1997.

[22] P.Flocchini, B. Mans, and N. Santoro. *On the impact of sense of direction on message complexity.* Information Processing Letters, **63(1)**:23-31, 1997.

[23] P. Flocchini and B. Mans. *Optimal election in labelled hypercubes.* Journal of Parallel and Distributed Computing, **33(1)**:76-83, 1996.

[24] P. Flocchini, B. Mans, and N. Santoro. *Distributed traversal and broadcasting in arbitrary network with distance sense of direction.* In Proc. of 9th International Symposium on Computer and Information Sciences, 196-203, 1994.

[25] J-L Fouquet et G.Hahn. *Cycle regular graphs need not be transitive*, Discrete Applied Mathematics, **113**:261-264, 2001.

[26] A. Israeli, E. Kranakis, D. Krizanc, and N. Santoro. *Time-message trade-offs for the weak unison problem.* Nordic Journal of Computing, **4**:317-329, 1997.

[27] T.Z. Kalamboukis and S.L. Mantzaris. *Towards optimal distributed election on chordal rings.* Information Processing Letters, **38**:265-270, 1991.

[28] E. Korach, and S. Moran. *A modular technique for the design of efficient distributed leader finding algorithms.* A.C.M. Transactions on Programming Languages and Systems, **12(1)**:84-101, 1990.

[29] E. Kranakis and D. Krizanc. *Distributed computing on anonymous hypercubes.* Journal of Algorithms, **23**: 32-50, 1997.

[30] E. Kranakis and N. Santoro. *Distributed computing on anonymous hypercube with faulty components.* In Proc. of 6th International Workshop on Distributed Algorithms, 253-263, Haifa, 1992.

[31] I. Lavallée and G. Roucairol. *A fully distributed (minimal) spanning tree algorithm.* Information Processing Letters, **23**:55-62, 1986.

[32] M.C. Loui, T.A. Matsushita, and D.B. West. *Election in complete networks with a sense of direction.* Information Processing Letters, **22**:185-187, 1986.

[33] B. Mans. *Optimal distributed algorithms in unlabelled tori and chordal rings.* Journal on Parallel and Distributed Computing, **46(1)**:80-90, 1997.

[34] Y. Metivier, A. Muscholl, and P.A. Wacrenier. *About the local detection of termination of local computing in graphs.* In Proc. of the 4th International Colloquium on Structural Information and Communication Complexity, 188-200, Ascona, 1997.

[35] G.L. Peterson. *Efficient algorithms for elections in meshes and complete networks.* Technical Report TR-140, Dept. of Computer Science, Unive. of Rochester, Rochester, NY-14627, 1985.

[36] S. Robbins and K. A. Robbins. *Choosing a leader on a hypercube.* In Proc. of International Conference on Databases, Parallel Architectures and their Applications, 469-471, 1990.

[37] N. Santoro. *Sense of direction, topological awareness and communication complexity.* SIGACT NEWS, **2(16)**:50-56, 1984.

[38] G. Tel. *Linear election in hypercubes.* Parallel Processing Letters, **5(1)**:357-366, 1995.

[39] G. Tel. *Sense of direction in processor networks.* In Proc. of 22nd Seminar on Current Trends in Theory and Practice of Informatics, 50-82, 1995.

[40] G. Tel. *Network orientation.* International Journal of Foundations of Computer Science, **5(1)**:1-41, 1994.

[41] M. Yamashita and T. Kameda. *Computing on anonymous networks, part 1: characterizating the solvable cases.* I.E.E.E. Transaction on Parallel and Distributed Computing, **7(1)**:69-89, 1996.

[42] Yi Pan. *A near-optimal multi-stage distributed algorithm for finding leaders in clustered chordal rings.* Information Sciences, **76(1-2)**:131-140, 1994.

APPENDIX A

THE BACKTRACK ALGORITHM IN JAVA

```
import java.io.*;
import java.util.*;
public class p0{
List1 adjacent[];//the adjacency list of the graph
int n;  //nb of nodes in the graph
int degree; //the degree of the graph
Object labelUsed[];//the set of labels used
int mincolor; //the minimal number of labels
Object minPer[];//the permutaion which gives the best solution
int temp[];

public p0(List1 adjacent[],int n,int degree) {
  this.n=n;
  this.adjacent=new List1[n];
  this.adjacent=adjacent;
  this.degree=degree;
  labelUsed=new Object[n];
  minPer=new Object[n];
  mincolor=n;
  temp=new int [n];
}

void backtrack(int l,List1 vertexPer,ListNode point,int labelofl){
  int nblabel=labelofl;
  List1 choiceSet=new List1();
  ListNode p1=vertexPer.firstNode;
  ListNode p2=null;
  while(p1 != null){
      choiceSet.insertAtBack(p1.data);
      if((p1.data).equals((point.data))){
          p2=choiceSet.lastNode;
      }
      p1=p1.next;
  }
  if(l==n){
      labelUsed[l-1]=p2.data;
      choiceSet.removeFromCurrent(p2);
```

```
        nblabel=calculate(labelUsed,n,nblabel);
        if(nblabel<mincolor){
            mincolor=nblabel;
            for(int u=0;u<n;u++)
                minPer[u]=labelUsed[u];
        }
    }
    if(l==0){
        for(int i=0;i<n;i++)
            choiceSet.insertAtBack(new Integer(i));
    }
    if(l!=n){
        if((l!=0)&&(!choiceSet.isEmpty())){
            labelUsed[l-1]=p2.data;
            choiceSet.removeFromCurrent(p2);
            if(l>1){
                nblabel=calculate(labelUsed,l,nblabel);
            }
        }
        ListNode pointer1=choiceSet.firstNode;
        ListNode pointer2=choiceSet.firstNode;
        while(pointer1 !=null){
            if(nblabel>mincolor){
                return;}
            pointer2=pointer1;
            pointer1=pointer1.next;
            backtrack(l+1,choiceSet,pointer2,nblabel);
        }
    }
}


int calculate(Object labelUsed[],int l,int nblabel){
    ListNode po=adjacent[l-1].firstNode;
    while((po!=null)&&((Integer.parseInt((po.data).toString()))<l)  )
    {
        int lab1=Integer.parseInt((labelUsed[l-1]).toString())-Integer.parseInt(
        (labelUsed[(Integer.parseInt((po.data).toString()))]).toString());
        int lab2=Integer.parseInt((labelUsed[
        (Integer.parseInt((po.data).toString()))]).toString())-
        (Integer.parseInt((labelUsed[l-1]).toString()));
        if(lab1<0)lab1=lab1+n;
        if(lab2<0)lab2=lab2+n;
```

```
        boolean used=false;
        for(int k=0;k<nblabel&&!used;k++){
            if((temp[k]==lab1)||(temp[k]==lab2))used=true;}
        if(!used){
            temp[nblabel]=lab1;
            nblabel++;

            if(lab1 !=lab2){
                temp[nblabel]=lab2;
                nblabel++;
            }
        }

        po=po.next;
    }
    return nblabel;
}

public static void main(String[] args){
    int n,d=0;
    List1 adjacent[];
    int degree=0;
    String myString;
    BufferedReader inputStream=new BufferedReader(
    new InputStreamReader(System.in));
    try{
        System.out.println("Give the number of nodes n\\
        and the adjacency matrix of the graph ");
        n=Integer.valueOf(inputStream.readLine().trim()).intValue();
        adjacent=new List1[n];
        for(int i=0;i<n;i++){
            adjacent[i]=new List1();
            d=0;
            myString=inputStream.readLine();
            StringTokenizer st=new StringTokenizer(myString);
            while( st.hasMoreTokens()){
                String s=st.nextToken();
                adjacent[i].insertAtBack(new Integer(Integer.parseInt(s)));
                d++;
            }
            if(degree<d)
                degree=d;
        }
    }
    catch(IOException e){
```

```
        e.printStackTrace();
        return;
    }

    // Get current time
    long start = System.currentTimeMillis();
    p0 project=new p0(adjacent,n,degree);
    List1 vertexPer=new List1();
    project.backtrack(0,vertexPer,null,0);
    // Get elapsed time in milliseconds
    System.out.println("The elapsed time of this program is :
    "+(System.currentTimeMillis()-start)+" ms");
    System.out.println("The minimum number of labels needed is :
    "+project.mincolor);
    System.out.println("A permutaion of the nodes which
    gives this solution is :");
    for(int i=0;i<n;i++){
        System.out.print("     "+project.minPer[i]);

    }
}
}



//Both algorithms uses the class List which has as code:
// Class List definition
public class List1 {
    public ListNode firstNode;
    public ListNode lastNode;
    public List1()
    {
        firstNode = lastNode = null;
    }

    // Insert an Object at the front of the List
    // If List is empty, firstNode and lastNode will refer to
    // the same object. Otherwise, firstNode refers to new node.

    public synchronized void insertAtFront( Object insertItem )
    {
        if ( isEmpty() )
            firstNode = lastNode=new ListNode( insertItem );
        else
```

```
        firstNode = new ListNode( insertItem, firstNode, null);
}

// Insert an Object at the end of the List
// If List is empty, firstNode and lastNode will refer to
// the same Object. Otherwise, lastNode's next instance
// variable refers to new node.
public synchronized void insertAtBack( Object insertItem )
{
    if ( isEmpty() )
        firstNode = lastNode= new ListNode( insertItem );
    else
    lastNode = lastNode.next = new ListNode( insertItem,null,lastNode );
}

// Remove the first node from the List
public synchronized Object removeFromFront()throws EmptyListException
{
    Object remove=null;
    if(isEmpty())
        throw new EmptyListException();
    remove=firstNode.data;
    if(firstNode.equals(lastNode))
        firstNode=lastNode=null;
    else
        firstNode=firstNode.next;
    return remove;
}

public synchronized Object removeFromCurrent(ListNode pointer1)
throws EmptyListException{
    Object remove=null;
    if(isEmpty())
        throw new EmptyListException();
    else if(firstNode.equals(lastNode)){
        firstNode=lastNode=null;
    }
    else if(firstNode==pointer1){
        remove=firstNode.data;
        pointer1=firstNode.next;
        firstNode=firstNode.next;
        firstNode.previous=null;
    }
    else if(lastNode==pointer1){
        pointer1=lastNode.previous;
```

```
            lastNode.previous.next=null;
        }

        else{
            pointer1.previous.next=pointer1.next;
            pointer1.next.previous=pointer1.previous;
        }
        return remove;
    }
   // Return true if the List is empty
    public synchronized boolean isEmpty()

    {
        return firstNode == null;
    }
}

class ListNode {
    Object data;
    ListNode next;
    ListNode previous;
   ]
       // Constructor: Create a ListNode that refers to Object o.
       ListNode( Object o ) { this( o,null,null); }

   // Constructor: Create a ListNode that refers to Object o and
   // to the next ListNode in the List.
   ListNode( Object o, ListNode nextNode ,ListNode previousNode)
   {
       data = o;          // this node refers to Object o
       next = nextNode;  // set next to refer to next
       previous=previousNode;
   }
}
public class EmptyListException extends RuntimeException {
    public EmptyListException ()
    {
        super( "The list is empty" );
    }
}
```

APPENDIX B

THE GENETIC ALGORITHM IN JAVA

```
import java.io.*;
import java.util.*;
public class p33 {
    int n;//number of nodes
    int Xbest[];//the best permutation found
    int pop[][];
    int popSize;//the population size
    int table[];
    int mincolor;//the minimum nb of labels found
    List1 adjacent[];//the adjacency list of the graph
    public p33(List1 adjacent[],int n,int popSize) {
        this.adjacent=new List1[n];
        for(int i=0;i<n;i++)
            this.adjacent[i]=new List1();
        this.adjacent=adjacent;
        this.n=n;
        table=new int[2*popSize];
        this.pop=new int[2*popSize][n];
        this.popSize=popSize;
        Xbest=new int[n];
        for(int i=0;i<n;i++)
            Xbest[i]=0;
    }

    void Rec(int p1[],int p2[],int i){
        int h=2+(int)(Math.random()*(n/2));
        int j=(int)(Math.random()*(n));
        boolean exist[]=new boolean[n];
        int k=0;
        for(int v=0;v<n;v++){
            exist[v]=false;
        }
        for(k=0;k<h;k++){
            pop[popSize+2*i+1][k]=p2[(k+j)%n];
            exist[pop[popSize+2*i+1][k]]=true;
        }
```

```
    for(j=0;j<n;j++){
        if(!exist[p1[j]]){
            pop[popSize+2*i+1][k]=p1[j];
            k++;
        }
    }

    table[popSize+2*i+1]=calculate(pop[popSize+2*i+1]);
    steepestAscentTwoOpt(popSize+2*i+1);
    j=(int)(Math.random()*(n));
    for(int v=0;v<n;v++){
        exist[v]=false;
    }

    for(k=0;k<h;k++){
        pop[popSize+2*i][k]=p1[(k+j)%n];
        exist[pop[popSize+2*i][k]]=true;
    }

    for(j=0;j<n;j++){
        if(!exist[p2[j]]){
            pop[popSize+2*i][k]=p2[j];
            k++;
        }
    }
    table[popSize+2*i]=calculate(pop[popSize+2*i]);
    steepestAscentTwoOpt(popSize+2*i);
}

int calculate(int pop1[]){
    int nblabel=0;
    int temp[]=new int[n];
    for(int x=0;x<n;x++){
        temp[x]=0;}
    for(int i=0;i<n;i++){
        ListNode po=adjacent[i].firstNode;
        while((po!=null)&&(Integer.parseInt((po.data).toString())<i))
        {
            po=po.next;
        }

        while((po!=null)&&((Integer.parseInt((po.data).toString()))<n)  ){
            int lab1=pop1[i]-pop1[(Integer.parseInt((po.data).toString()))];
            int lab2=pop1[(Integer.parseInt((po.data).toString()))]-pop1[i];
            if(lab1<0)lab1=lab1+n;
```

```
            if(lab2<0)lab2=lab2+n;
            boolean used=false;
            for(int k=0;k<nblabel&&!used;k++){
                if((temp[k]==lab1)||(temp[k]==lab2))used=true;}
            if(!used){
                temp[nblabel]=lab1;
                nblabel++;
                if(lab1 !=lab2){
                    temp[nblabel]=lab2;
                    nblabel++;
                }
            }
            po=po.next;
        }
    }
    return nblabel;
}

void steepestAscentTwoOpt(int index){
    int g0;
    int g;
    int label1;
    int label2=0;
    int temp[]=new int [n];
    g0=0;
    for(int i=0;i<n;i++){
        for(int j=i+2;j<n;j++){
            label1=table[index];
            for(int k=0;k<=i;k++){
                temp[k]=pop[index][k];
            }
            for(int k=i+1;k<=j;k++){
                temp[k]=pop[index][j-k+i+1];
            }
            for(int k=j+1;k<n;k++){
                temp[k]=pop[index][k];
            }
            label2=calculate(temp);
            g=label1-label2;
            if(g>g0){
                g0=g;
                for(int r=0;r<n;r++){
                    pop[index][r]=temp[r];
```

```
                }
                table[index]=label2;
            }
        }
    }
}

int[] getpermutation(){
    int pop[]=new int [n];
    int temp[]=new int[n];
    int randomindex;
    for(int k=0;k<n;k++)
        temp[k]=k;
    for(int h=n-1;h>=0;h--){
        randomindex=(int)(Math.random()*(h+1));
        pop[n-h-1]=temp[randomindex];
        for(int u=randomindex;u<h;u++){
            temp[u]=temp[u+1];
        }
    }
    return pop;
}

void Select(){
    for(int i=0;i<popSize;i++){
      pop[i]=getpermutation();
      table[i]=calculate(pop[i]);
      steepestAscentTwoOpt(i);
    }
}

void swap(int i){
    int hold;
    int hold2[]=new int[n];
    hold=table[i];
    table[i]=table[i+1];
    table[i+1]=hold;
    hold2=pop[i];
    pop[i]=pop[i+1];
    pop[i+1]=hold2;
}

void sort(int n){
    for(int pass=1;pass<n;pass++)
```

```
        for(int i=0;i<n-1;i++)
            if(table[i]>table[i+1])
                swap(i);
}

int[] Genetic(int cmax){
    int c=1;
    int curCost=0;
    Select();
    sort(popSize);
    Xbest=pop[0];
    minfcolor=table[0];
    while(c<=cmax){
        for(int i=0;i<=(popSize/2)-1 ;i++)
        {
            Rec(pop[2*i],pop[2*i+1],i);
        }
        sort(2*popSize);
        curCost=table[0];
        if(curCost<mincolor){
            Xbest=pop[0];
            mincolor=curCost;
        }
        c++;
    }
    return Xbest;
}

public static void main(String[] args){
    int n=0;
    List1 adjacent[];
    int value=0;
    int cmax;
    int popSize;
    String myString;
    BufferedReader inputStream=new BufferedReader(new
     InputStreamReader(System.in));
    try{
        System.out.println("Give the number of nodes n,
        the number of iterations, and the population size:");
        n=Integer.valueOf(inputStream.readLine().trim()).intValue();
        cmax=Integer.valueOf(inputStream.readLine().trim()).intValue();
        popSize=Integer.valueOf(inputStream.readLine().trim()).intValue();
        System.out.println("Enter the adjacency list : ");
```

```
            adjacent=new List1[n];
            for(int i=0;i<n;i++){
                adjacent[i]=new List1();
                myString=inputStream.readLine();
                StringTokenizer st=new StringTokenizer(myString);
                while( st.hasMoreTokens()){
                    String s=st.nextToken();
                    adjacent[i].insertAtBack(new Integer(Integer.parseInt(s)));
                }
            }
        }
        catch(IOException e){
            e.printStackTrace();
            return;
        }
        p33 project=new p33(adjacent,n,popSize);
        long start = System.currentTimeMillis();
        project.Genetic(cmax);
        System.out.println("The elapsed time of this program is :
        "+((System.currentTimeMillis())-start)+" ms");
        System.out.println("The minimum number of labels needed is:
         "+project.mincolor);
        System.out.println("A permutation of the nodes which
        gives this solution is: ");
        for(int i=0;i<n;i++){
            System.out.print("     "+project.Xbest[i]);
        }

    }
}
```