UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

ON SUBGRAPHS OF MINIMAL CAYLEY GRAPHS

Autor: Álvaro Soto Gómez

Director:	**Dr. Kolja Knauer**
Realitzat a:	**Departament de Matemàtiques i Informàtica**
Barcelona,	**15 de gener de 2025**

Abstract

In this paper, we aim to identify graphs that are not subgraphs of minimal Cayley graphs. Specifically, we will focus our study on cubic graphs. To this end, we first examine the fundamental properties of minimal Cayley graphs and establish two necessary conditions for a graph to be a subgraph of one. We then provide a proof of Spencer's theorem. Later on, we analyze the subcase of the family of Generalized Petersen graphs by showing which of them are minimal Cayley graphs and demonstrating that, for certain parameters, generalized Petersen graphs are subgraphs of a minimal Cayley graph of the semidirect product of two cyclic groups. Finally, after covering the necessary theoretical groundwork, we will present and explain the algorithm developed for identifying prohibited graphs.

Resum

En aquest text, el nostre objectiu és identificar grafs que no són subgrafs de grafs de Cayley mínims. Concretament, centrarem el nostre estudi en els grafs cúbics. Per assolir aquest objectiu, primer examinarem les propietats fonamentals dels grafs de Cayley mínims i establirem dues condicions necessàries perquè un graf sigui un subgraf d'un d'aquests. A continuació, proporcionarem una demostració del teorema de Spencer. Posteriorment, analitzarem el subcàs de la família dels grafs generalitzats de Petersen, mostrant quins d'aquests són grafs de Cayley mínims i demostrant que, per a certs paràmetres, els grafs generalitzats de Petersen són subgrafs d'un graf de Cayley mínim del producte semidirecte de dos grups cíclics. Finalment, després de cobrir les bases teòriques necessàries, presentarem i explicarem l'algoritme desenvolupat per identificar grafs prohibits.

2020 Mathematics Subject Classification. 05C60, 05C99, 20C99

Acknowledgments

I would like to express my deepest gratitude to Kolja Knauer for his patience, dedication, and support over the past few months. I am especially thankful for the opportunity to work on such an intriguing and challenging topic, as well as for his guidance throughout this journey.

Contents

Chapter 1

Introduction

A *Cayley graph* is a structure whose vertices correspond to the elements of a group, which are connected by edges defined by the generating set. As a consequence of this definition, it appears the concept *minimal Cayley graph*, which is the term used when the generating set of a given Cayley graph is irreductible, i.e. does not properly contain another generating set.

László Babai studied these kinds of graphs extensively in his research. In fact, this project is motivated by the publication [1]. In the cited chapter, Babai wonders if there exists a cubic girth 5 graph which is not subgraph of any minimal Cayley graph. This question arises as a consequence of Spencer's Theorem, which states that for every girth greater than 2, there is a finite graph with maximum degree at most 100 that is not subgraph of any minimal Cayley graph. Babai's question can be seen as a wondering of how much Spencer's Theorem can be improved. Specifically, what is the best possible minimum degree for a given girth? To simplify, one might restrict the focus to regular graphs, where all vertices have the same degree.

For girth 3, there are cubic graphs that are not subgraphs of minimal Cayley graphs (see Section 2.2 and Corollary 2.13). But, for girth 4 or greater, no such examples have been identified. This raises another question: Is there any subcubic graph (a graph with a maximum degree of three) of girth at least 4 that is not a subgraph of any minimal Cayley graph? Answering this would automatically solve Babai's question, as we have seen in the Proposition 2.16. However, we have not been able to answer this question neither.

In any case, we have proved a series of properties that will help us in that matter. For starters, we will see in the Proposition 2.9 that the *no lonely color property*, introduced by Babai in [2], is a necessary condition to be a minimal Cayley graph. Additionally, since Cayley graphs are directed, we introduce a simple criterion in Proposition 2.29 to determine whether a directed graph can be a subgraph of a directed Cayley graph. These two criteria together will be used in the computational test to see whether a graph (can be oriented such that it) can be sub(di)graph of a minimal (directed) Cayley graph. To end the chapter, we will show in the Theorem 2.43 that every minimal Cayley graph of a group G is locally isomorphic to a minimal Cayley graph of the group G/Φ, where Φ is the Frattini subgroup of G.

Afterwards, we show the proof due to Spencer in the Theorem 3.5. Later on, we will study the family of Generalized Petersen graphs, which are cubic graphs. We will expose the results of [22] characterizing the minimal Cayley graphs among them in the Section 4.2. As a new result, Proposition 4.16 demonstrates that for certain parameters, generalized Petersen graphs are subgraphs of a minimal Cayley graph of the semidirect product of two cyclic groups.

Finally, in the concluding chapter, we report on our computational efforts to analyze some given graphs. The cubic graphs used in these analyses are obtained from [16]. For the minimal Cayley graphs, we will be using the list of minimal Cayley graphs on up to 255 vertices produced by Rhys J. Evans, but we can also obtain these graphs up to 95 vertices in [17]. To obtain the orientations of a given graph and the subcubic graphs, we will use *nauty* (See [20]). The whole program is in *Github* (click here to see it), but we can see some fragments of the code in the appendix.

1.1 Notation and basic concepts

In this project, we will be especially interested in simple connected graphs without multiple edges. So, unless otherwise stated, we will always refer to these types of graphs. Having said this, we can start with the basic definitions that we will be using during the next pages.

Definition 1.1. A *simple graph* is a pair $X = (V, E)$, where V is a finite set of vertices and E is a finite set of edges that satisfy the following condition:

$$E \subseteq \{\{u, v\} \mid u, v \subseteq V \text{ and } u \neq v\}$$

Notation 1.2. As seen in the previous definition, we will use the notation $\{u, v\}$ to denote the edge that connects the vertices u, v of the graph.

Definition 1.3. A *directed graph* (or *digraph*) is a pair $X = (V, A)$, where V is a finite set of vertices and A is a finite set of edges (or arcs) that satisfy the following condition:

$$A \subseteq \{(u,v) \mid u,v \subseteq V \text{ and } u \neq v\}$$

Notice that the order of the edges is important because it indicates the direction. An edge can also have both directions. Only in this case will we use multiple edges, by placing two edges in A with the same vertices but reversed: (u,v) and (v,u).

To proceed, we introduce some definitions to distinguish between different ways of traversing the edges and vertices of a graph:

Definition 1.4. Given a graph $X = (V,E)$, we define the following concepts:

i) A *walk* $W = (v_1, e_1, v_2, ..., v_k, e_k, v_{k+1})$ is a sequence of vertices and edges such that $e_i = \{v_i, v_{i+1}\}$ for every $1 \leq i \leq k$. If $v_1 = v_{k+1}$, the walk is said to be *closed*.

ii) A *path* is a walk in which no vertex is repeated.

iii) A *cycle* is a closed path, i.e. a path in which the first and last vertices coincide.

For convenience, we include both vertices and edges in the notation for a walk. However, depending on the context, we may use only one of them to simplify the representation.

Now, regarding the connectivity of graphs, we will define the following two concepts:

Definition 1.5. Let $X = (V,E)$ be a graph. Then:

i) X is said to be *connected*, if and only if, for every two vertices $u,v \in V$, there is a path from u to v in X. Otherwise, we say that X is *disconnected*.

ii) The *connected components* of X are its maximal connected subgraphs.

Definition 1.6. Let $X = (V,E)$ be a graph, $S \subseteq V$ a subset of vertices and $F \subseteq E$ a subset of edges of X.

i) The graph $Y = (S,F)$ is a *subgraph* of X, if every edge in F has both of its endpoints in S.

ii) The *induced subgraph* $X[S]$ is the graph whose vertex set is S and whose edge set consist of all the edges in E that have both endpoints in S.

Remark 1.7. In this project, when we use the term *subgraph*, we will always refer ourselves to the concept defined in the Definition 1.6 (i). So, whenever we talk about induced subgraphs, we will say it explicitly.

Chapter 2

On minimal Cayley graphs

2.1 Definition of the main concepts

In this section, we will present the most important definitions that concern us. In addition, we will introduce the main question, which will be deeply treated in this project.

Definition 2.1. We say that a graph X is a *Cayley graph* $X = Cay(G, S)$ of the group G with respect to the generating set S if its vertices V and its edges E are defined as follows:

i) $V(X) = G$

ii) $E(X) = \{\{g, gs\} \mid g \in G,\ s \in S\}$

On the other hand, if we are considering the *directed Cayley graph*, we will denote it as $X = \overrightarrow{Cay}(G, S)$. In this case, the form of its edges (or arcs) will be the following:

ii) $A(X) = \{(g, gs) \mid g \in G,\ s \in S\}$

Definition 2.2. Let $X = Cay(G, S)$ be a (directed) Cayley graph. Then, if no proper subset of S generates G, we say that X is a *minimal (directed) Cayley graph*.

Definition 2.3. A graph is said to be *cubic* if all its vertices have degree three. In other words, a cubic graph is a 3-regular graph.

Definition 2.4. The *girth* of an undirected graph is the length of the shortest cycle contained in the graph.

In this text, our main goal will be to study which cubic graphs are subgraphs of minimal Cayley graphs. Nevertheless, what will take most of our time, will be the study of the following question:

Question 2.5. Is there a cubic girth 5 graph which is not a subgraph of a minimal Cayley graph?

That inquiry was laid on the table by László Babai in [1]. But, what was the reason that made him think about the girth 5 property? What was that chain of reasoning that brought him there?

Before entering into the following section, we consider that it is important to mention a theorem proved by Babai and Sós in [4], which predates the formulation of the central question of this project.

Theorem 2.6. *Every graph is an induced subgraph of some Cayley graph of any sufficient large group.*

Releated to this Theorem, we also have the papers [3, 14]. However, in this project, in this project, we do not limit our study to induced subgraphs and we focus exclusively on minimal Cayley graphs, rather than general Cayley graphs.

2.2 The no lonely color property

To continue, we will present one of the most important results of our project. Right now, the question proposed above is very abstract and may seem difficult to answer. However, we will reveal a necessary condition that will transform this abstraction into a simple coloring problem.

In order to write the following definitions and propositions, we have followed the paper [2] written by László Babai.

Definition 2.7. An undirected graph X is called a *no lonely color graph* if its edges can be colored such that:

i) each vertex is incident with at most two edges of any color.

ii) for any edge of any cycle, there is another edge of the same color in that cycle.

Remark 2.8. To refer ourselves to the neutral element of a group, we will use the notation e.

Proposition 2.9. *Let Y be a subgraph of a minimal Cayley graph $X = Cay(G,S)$. Then, Y is a no lonely color graph.*

Proof. To start, we take S as the set of colors and we assign the color $s \in S$ to the edge $\{g, gs\}$, where $g \in G$. Observe that the edges of color s incident with g can only be $\{g, gs\}$ and $\{g, gs^{-1}\}$. These two edges will always be different unless if $s^2 = e$. With this, we have just seen that the condition i) of the Definition 2.7 is satisfied for X, so it will also be satisfied for the subgraph Y.

To continue, let us consider a cycle in Y having colors $s_1, s_2, ..., s_k$ in this order. This means that $s_1^{\pm 1} s_2^{\pm 1} \cdots s_k^{\pm 1} = e$. So, if we suppose that, for example, s_1 is a lonely color, then we would be able to express s_1 using the rest of the colors in the cycle: $s_1^{\mp 1} = s_2^{\pm 1} \cdots s_k^{\pm 1}$, which means that $S \setminus \{s_1\}$ would be a generating set for G. However, since S is irreducible, this is not possible. As a consequence, no color in a cycle can be lonely, which means that the condition ii) of the definition is also satisfied. □

We have just seen that it is a necessary condition for a graph to be a no lonely color graph, so that it can have a chance to be a subgraph of a minimal Cayley graph. To carry on, the following lemma will serve us as an example of how we can proceed to see that a graph is a no lonely color:

Lemma 2.10. *A no lonely color graph does not contain any $K_{3,5}$.*

Proof. Assume, to the contrary, that there exists a no lonely color graph which contains the bipartite graph $K_{3,5}$ as a subgraph. This implies that $K_{3,5}$ is a no lonely color graph too by the Proposition 2.9.

Now, suppose that $K_{3,5}$ has a no lonely coloring. Let $V(K_{3,5}) = A \cup B$, $|A| = 3$, $|B| = 5$ with all the edges going between A and B. To continue, consider the vertex $b \in B$. By i), in the Definition 2.7, we can say that there are at least two members $a_1, a_2 \in A$ such that the edges $[b, a_1]$ and $[b, a_2]$ have different colors. We will denote by γ_i the color of the edge $[b, a_i]$.

According to the first condition of the previous definition, there must be at most 3 edges joining a_i to $B \setminus \{b\}$ whose color belongs to $\{\gamma_1, \gamma_2\}$. Therefore, there exists a vertex $b' \in B \setminus \{b\}$ such that none of the edges $\{b, a_i\}$ with $i = 1, 2$ has any of the colors γ_1, γ_2. We will assume that the edges $\{b', a_1\}$ and $\{b', a_2\}$ have the same color α (if it isn't like that, the result is the same).

However, now the cycle (b, a_1, b', a_2, b) does not satisfy the condition ii) of the Definition 2.7, because γ_1 and γ_2 only appear one time in the cycle (see Figure 2.1). So, we have arrived to a contradiction. □

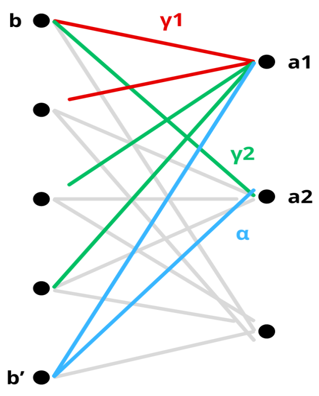


Figure 2.1: Coloring of $K_{3,5}$.

Similar to this lemma, Babai also proves in his paper [2] the same result for the complete graph K_4 minus an edge (usually denoted as K_4^-).

Definition 2.11. Let X be a graph. Then, we will say that X is a *prohibited* graph if it is not a subgraph of any minimal Cayley graph.

Then, according to this last definition, we can say that $K_{3,5}$ and K_4^- are both prohibited graphs. And, as a direct consequence, K_4 is prohibited too. More importantly, it follows from this, that the number of prohibited cubic graphs with girth 3 is infinite. To show that, we will define the following family of graphs:

Definition 2.12. Consider the family of graphs that is formed by taking a cycle of length n and n copies of K_4^-. Then, to unify these graphs, we will replace the vertices of the cycle for each one of the copies of K_4^-. Lastly, we will join the vertices of degree two with the rest of the copies forming a cycle. We will denote this family as F_n. The result is the following graph:

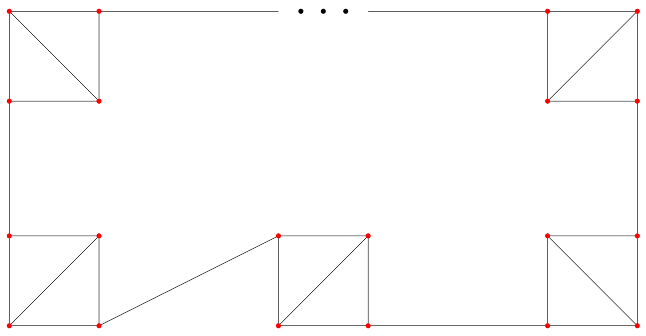


Figure 2.2: Representation of the family of graphs F_n.

Therefore, since every graph of this family contains K_4^- as a subgraph, we have the following property that shows that there is an infinite number of prohibited connected cubic graphs of girth 3:

Corollary 2.13. *All graphs that belong to the family F_n are prohibited.*

To continue, we will now present a result that will be very useful in the future to determine whether a coloring is a no lonely coloring or not:

Proposition 2.14. *Consider an undirected graph $X = (V, E)$ colored such that:*

i) $E = C_1 \cup \cdots \cup C_k$, where C_i represents the set of edges that have color c_i.

ii) every vertex $v \in V$ is adjacent with at most two edges of the same color.

Then, X is a no lonely color graph, if and only if for every color c_i, $X \setminus C_i$ is disconnected and no edge in C_i has both endpoints sharing one of the resulting components.

Proof. Suppose that there exists a graph X, satisfying these conditions, which is not a no lonely color graph. Then, since by hypothesis no vertex is adjacent with more than two edges of the same color, we can suppose that in this graph there will be a cycle that has an edge $e \in C_r$ which does not share color with any other edge of the cycle.

Now, imagine that we start the cycle in a component A of the disconnected graph $X \setminus C_r$. As there will be no edge in C_r that has both endpoints in A, if the cycle contains an edge of C_r, it is necessary to leave the component A. But, to close the cycle, we will have to return to the initial connected component A, so we will have at least two edges of C_r in the cycle, because $X \setminus C_r$ is disconnected. Hence, we have arrived to a contradiction.

For the proof of the opposite direction, we will see that if $X \setminus C_i$ is connected or there is an edge in C_i that has both endpoints sharing one of the resulting components, then X is not a no lonely coloring.

To start, observe that if $X \setminus C_i$ is connected, this means that all the edges in C_i will have both endpoints in the same component. Then, for the purpose of the proof, we only have to consider the case where the is an edge e_0 of C_i that has both endpoints in the same component of $X \setminus C_i$. If this happens, necessarily there will be a cycle in $X \setminus C_i \cup \{e_0\}$, where e_0 will be a lonely color edge. Then, X will not be a no lonely coloring. □

From this last result we obtain the following Theorem proved in [23]. This charactherization of prohibited graphs, will be specially useful for the proof of Spencer's Theorem:

Theorem 2.15. *Let $Y = (V, E)$ be a graph, with $|V| = n$, which satisfies that for each subgraph Z that contains $t \leq \frac{n}{2}$ vertices, there are more than $2t$ edges connecting Z to Z^c in Y. Then, Y is a prohibited graph.*

Proof. We will start considering a minimal Cayley graph $X = Cay(G, S)$ such that $Y \subseteq X$. Then, by the Proposition 2.9, Y has a no lonely coloring. Then, we can take a set of edges $C_i \subseteq E(Y)$ colored with c_i and, by the Proposition 2.14, $Y \setminus C_i$ will be disconnected and no edge in C_i will have both endpoints in the same component. Therefore, we can take any component Z of $Y \setminus C_i$ which satisfies $|Z| \leq \frac{n}{2}$ and, since we are considering a no lonely coloring of Y, there will be at most two neighbors in Z^c for every vertex of Z.

In other words, we have seen that there will be at most $2|Z| = 2t$ edges from Z to Z^c in Y, which is in contradiction with the hypothesis of the theorem. Hence, this concludes the proof. □

Notice that we have seen some properties of the no lonely color graphs which can be helpful to find prohibited graphs. So, what if we were able to transform the procedure used in the proof of the Lemma 2.10 combined with the Proposition 2.14 into an algorithm? This could allow us to check which graphs are not subgraphs of minimal Cayley graphs. Although, it is true that there are a lot of possible colorings to check.

Now, to conclude this section, we will make a few comments related with the subcubic graphs. In the introduction, we mentioned that a possible question that we could ask ourselves is if there is a subcubic prohibited graph (i.e. graph of maximum degree three). We know that for the girth 3 ones, K_4^- answers the question positively. Nonetheless, for girth greater than 4, we do not know any prohibited graph.

However, notice that if a 3-regular graph X is subgraph of a minimal Cayley graph, then any of its connected subgraphs $Y \subseteq X$ will clearly be a subgraph of the same minimal Cayley graph. Or, equivalently:

Proposition 2.16. *Let X be a graph and Y a connected subgraph of X. Then, if Y is prohibited, X will be prohibited too.*

This property tells us that if we find a subcubic prohibited graph Y then, we can ensure the existence of a 3-regular prohibited graph. So, in case of not finding a cubic graph that is not subgraph of any minimal Cayley graph, it may be worthwhile to consider subcubic graphs as an alternative approach.

2.3 No lonely colorings via perfect matchings

In the previous section, we mentioned that there are a lot of possible colorings to consider for each graph. But, what if we could discard a big number of graphs by checking just a few of those colorings? Notice that, in cubic graphs, the condition (i) of the Definition 2.7 is clearly satisfied by perfect matchings. Hence, it could be a good idea to study first these types of colorings.

In this section, we will analyze the structure of the perfect matchings of a graph by presenting some relevant results from the literature, in particular, Petersen's Theorem.

Definition 2.17. An edge of a graph is called *bridge* or *cut-edge* if its deletion increases the graph's number of connected components. Consequently, we say that a graph is *bridgeless* if it contains no bridges.

Remark 2.18. An edge of a graph is a bridge, if and only if it is not contained in any cycle.

Definition 2.19. Given a simple undirected graph $X = (V, E)$, a *matching* M in X is a subset of edges of E with the condition that the edges do not share any endpoint.

Then, we say that M is a *perfect matching* if every vertex of X is adjacent to exactly one edge in M.

Once we have these preliminary definitions, we can now formulate Petersen's Theorem. This is a very important result in graph theory and in this project in particular:

Theorem 2.20. **(Petersen)** *Every cubic bridgeless graph contains a perfect matching.*

We can observe that, if we consider a graph $X = (V, E)$ without the edges of a perfect matching M, what we get is a set of cycles. That is because, in a cubic graph, when we remove a perfect matching, the degree of all vertices decrease to 2. So, the only possibility is that $X \setminus M$ is a set of cycles.

Then, if we color the edges of M differently from the rest, we get a good candidate for a no lonely edge coloring, which, as stated before, is a necessary condition to be a subgraph of a minimal Cayley graph. Nevertheless, it is not necessarily good, as we can see in Figure 2.3:

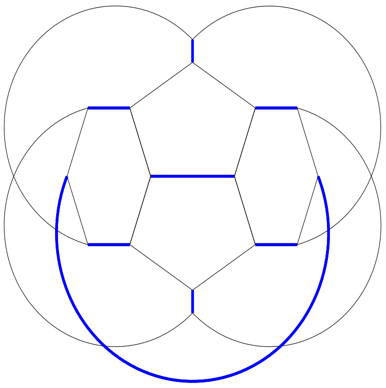


Figure 2.3: Perfect matching M_1 such that the partition $(M_!, E \setminus M_1)$ is not a no lonely coloring.

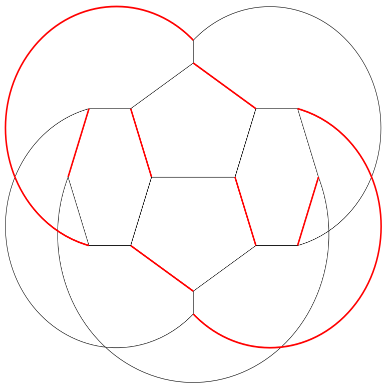


Figure 2.4: Perfect matching M_2 such that the partition $(M_2, E \setminus M_2)$ is a no lonely coloring.

Now, someone may be asking himself if all cubic graphs have a perfect matching which can be used as a no lonely coloring. However, we can quickly answer that question negatively by checking the perfect matchings of the graph K_4. In fact, K_4 is not a no lonely color graph. That is because it contains the graph K_4^-, which, as we have already mentioned in the previous section, can not be contained in any no lonely color graph.

To conclude this section, observe that here we are only considering perfect matchings. However, can we go a little further?

Definition 2.21. Let X be a graph. Then, a *k-factor* of X is a subgraph Y that contains all vertices of X and which satisfies that $deg(v) = k$ for every vertex $v \in V(Y)$.

Moreover, we have that a *k-factorization* is a partition of the edges of the graph into disjoint k-factors.

Remark 2.22. A 1-factor of a graph is a perfect matching.

So, maybe, instead of only considering perfect matchings (or 1-factors), we could try to search no lonely colorings using 1-factorizations, since they satisfy the degree constraints. In fact, we could also try 2-factors as possible candidates. Nevertheless, this would be more appropriate if we were not using cubic graphs, because these colorings would be equivalent to the ones obtained with the 1-factors.

2.4 Isomorphic walks on Cayley graphs

In this section, we will present a second necessary condition that, additionally to the no lonely color property, will help us to show that a given colored digraph is not a subgraph of a directed Cayley graph.

With this in mind, some results or definitions that we will be using have not been stated before. Still, this part of the text is based on elementary ideas that can be found in any standard book on algebraic Graph Theory [13, 18].

Definition 2.23. Let X, Y be two graphs. Then, given a mapping $\phi : V(X) \longrightarrow V(Y)$, we define the following concepts:

i) ϕ is a *graph homomorphism* if it preserves the edges. That is to say:

$$\{u,v\} \in E(X) \Longleftrightarrow \{\phi(u),\phi(v)\} \in E(Y)$$

ii) ϕ is an *isomorphism* if the mapping is a bijective homomorphism such that ϕ^{-1} is also an homomorphism. Then, we will say that X and Y are *isomorphic* ($X \cong Y$).

For the next definitions, suppose that $X = Y$ and, therefore, $\phi : V(X) \longrightarrow V(X)$:

iii) ϕ is an *automorphism* if it is an isomorphism from an object to itself. Their collection is called the *automorphism group* $Aut(X)$.

iv) ϕ is an *endomorphism* if it is an homomorphism from an object to itself.

Notation 2.24. Let $X = (V, A)$ be a directed graph with arcs colored with the colors $c_1, \dots, c_k$ and W a walk of the graph.

i) We will denote the underlying directed graph of the walk W as $\underline{W}$.

ii) If W has a sequence of colors $c_1^{s_1} c_2^{s_2} \cdots c_k^{s_k}$ where, for every $1 \leq i \leq k$, $s_i \in \{+1, -1\}$ indicates whether the arc is outgoing or incoming. We will denote this sequence of the walk as $c(W)$. When $s_i = +1$, we will normally omit the exponent for a simpler notation.

Example 2.25. Given the drawn graph in Figure 2.5, we will denote by b, g and r the colors blue, green and red, respectively. Now, consider the walks $W_1 = (v_1, v_2, v_3, v_4, v_7)$ and $W_2 = (v_2, v_5, v_6, v_7)$. Then,

$$c(W_1) = bg^{-1}b^{-1}g, \qquad c(W_2) = grr^{-1}$$

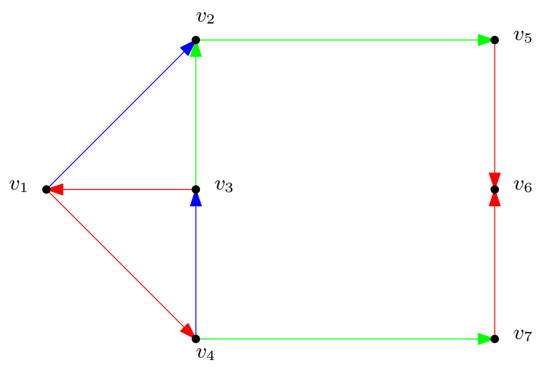


Figure 2.5: Colored directed graph.

Lemma 2.26. *Consider the directed Cayley graph $X = \overrightarrow{Cay}(G, S)$. If we define $\lambda_g : V(X) \longrightarrow V(X)$ where $\lambda_g(v) = gv$ for an element $g \in G$, then λ_g is an automorphism of X.*

Proof. To begin, λ_g is an endomorphism because, given two arbitrary elements $h \in G$ and $s \in S$, if $(h, hs) \in A(X)$ then $(\lambda_g(h), \lambda_g(hs)) = (gh, ghs) \in A(X)$. We can affirm that this last edge is in $A(X)$, since by the properties of a group $gh \in G$ and, by applying the generator s, we get to $ghs \in G$.

Now, we have to prove that λ_g is an isomorphism. That is to say that λ_g is bijective and that its inverse function is also an homomorphism, as we can see in the Definition 2.23.

We will start by seeing that it is a bijection. To do so, we have to see that λ_g is injective and surjective:

- Injectivity: Suppose that $\lambda_g(v_1) = \lambda_g(v_2)$ for any $v_1, v_2 \in V(X)$. Then, $gv_1 = gv_2$ and since $g \in G$ and G is a group, we can affirm that g has an inverse element $g^{-1} \in G$. So, multiplying by g^{-1} to the left of the previous equality, we get $v_1 = v_2$. This proves that λ_g is injective.

- Surjectivity: Now, for any $v' \in G$, we have to find an element $v \in G$ such that $\lambda_g(v) = v'$, i.e. $gv = v'$. Then, we can get v as $v = g^{-1}v'$. Hence, λ_g is surjective.

As we have already proven that λ_g is a bijection, we can continue with the second condition.

What we want to see now is that the inverse function λ_g^{-1} is also an homomorphism. To prove this, we can observe that $\lambda_g^{-1}(v) = g^{-1}v$, which is the same as saying that $\lambda_g^{-1} = \lambda_{g^{-1}}$. Therefore, applying the same argument that we have used in the first paragraph of this proof to $\lambda_{g^{-1}}$, we obtain that λ_g^{-1} is an homomorphism.

Thus, we have proven that λ_g is an automorphism of X for any $g \in G$. □

Proposition 2.27. *Let Y be a subdigraph of a directed Cayley graph X. If we have two walks W, W' in Y satisfying that $c(W) = c(W')$, then $\underline{W}$ and $\underline{W'}$ are isomorphic.*

Proof. To start, we will denote the vertices of Y that use W as v_i and the ones of W' as u_i. Moreover, suppose that $W = (v_0, \dots, v_i, (v_{j\in\{i,i+1\}}, v_{k\in\{i,i+1\}})_{j\neq k}, v_{i+1}, \dots, v_{n+1})$ and $W' = (u_0, \dots, u_i, (u_{j\in\{i,i+1\}}, u_{k\in\{i,i+1\}})_{j\neq k}, u_{i+1}, \dots, u_{n+1})$ and that their sequence of colors is $c(W) = c(W') = c_o^{s_0} \cdots c_n^{s_n}$, where the color c_i correspond to the arc $(w_{j\in\{i,i+1\}}, w_{k\in\{i,i+1\}})_{j\neq k}$ and the value of s_i depends on the order of the vertices.

$$s_i = \begin{cases} +1 & \text{if } j < k, \\ -1 & \text{if } j > k \end{cases}$$

Now, consider the mapping $\lambda_{u_0 v_0^{-1}} : V(X) \longrightarrow V(X)$ defined by $\lambda_{u_0 v_0^{-1}}(v) = (u_0 v_0^{-1})v$, where v_0 and u_0 are the vertices of Y from where the walks W and W' start respectively.

In addition, consider that in the walk W we have the edge (v_j, v_k) of color c_r, where $r = min\{j,k\}$. This means that $v_r = v_0 c_0^{s_0} \cdots c_{r-1}^{s_{r-1}}$ and $v_{r+1} = v_0 c_0^{s_0} \cdots c_{r-1}^{s_{r-1}} c_r^{s_r}$ are vertices of Y. Then,

$$\lambda_{u_0 v_0^{-1}}(v_r) = (u_0 v_0^{-1})v_r = (u_0 v_0^{-1})v_0 c_0^{s_0} \cdots c_{r-1}^{s_{r-1}} = u_0 c_0^{s_0} \cdots c_{r-1}^{s_{r-1}}$$

$$\lambda_{u_0 v_0^{-1}}(v_{r+1}) = (u_0 v_0^{-1})v_{r+1} = (u_0 v_0^{-1})v_0 c_0^{s_0} \cdots c_{r-1}^{s_{r-1}} c_r^{s_r} = u_0 c_0^{s_0} \cdots c_{r-1}^{s_{r-1}} c_r^{s_r}$$

This implies that $\lambda_{u_0 v_0^{-1}}(V(\underline{W})) = V(\underline{W'})$.

Finally, as we have seen in the previous lemma that $\lambda_{u_0 v_0^{-1}}$ is an automorphism of X, we can say that the restriction $\lambda_{u_0 v_0^{-1}} : V(\underline{W}) \longrightarrow V(\underline{W'})$ is an isomorphism. This is enough to show that $\underline{W} \cong \underline{W'}$. □

Now, we will return for a moment to the no lonely color graphs. Recall that in the section 2.2, we have only considered undirected graphs. But, what happens if the graph is directed?

Definition 2.28. A directed graph $X = (V, E)$ is called a *no lonely color graph* if its edges can be colored such that:

i) this coloring is a no lonely coloring for the undirected graph of X.

ii) for every color: $deg^+(v) \leq 1$ and $deg^-(v) \leq 1$, for every $v \in V$.

Where $deg^+(v)$ is the number of outgoing edges from v and $deg^-(v)$ represent the number of the incoming ones.

To continue, we will now enunciate two results that, even though they are almost direct using the previous results, they will be very useful in the future when we try to implement an algorithm on directed graphs.

Proposition 2.29. *Let Y be a directed subgraph of a minimal Cayley graph $X = \overrightarrow{Cay}(G, S)$ and W, W' two walks in Y such that $c(W) = c(W')$. Then, Y is a no lonely color graph and $\underline{W} \cong \underline{W'}$.*

Proof. To show that Y is a no lonely color graph, the condition i) of the previous definition is satisfied by the Proposition 2.9. To see ii), suppose we have a vertex $v \in V(Y)$ which $\deg^+(v) > 1$ for the color s. This color represents the generator $s \in S$. Then, we would be able to go to two different vertices that would represent the same element of G: vs. Like this, we arrive to a contradiction. The argument for the incoming edges is the same but using s^{-1}.

Furthermore, since Y is a subgraph of a minimal Cayley graph, the condition of the walks is directly satisfied by the Proposition 2.27. □

Lemma 2.30. *Let Y be a (directed) subgraph of a minimal (directed) Cayley graph $X = Cay(G, S)$. Then, all monochromatic cycles on Y have the same length.*

Proof. We will just proof the directed case, because the undirected one follows straightaway from it.

To start, notice that, if we find a monochromatic cycle, it will have all edges in the same direction, meaning that we will not be having any vertex of the cycle with two outgoing or incoming edges. This is by the Definition 2.28.

Now, consider that we have two monochromatic cycles, of color s, with length n and m respectively. Recall that the color s represents a generator $s \in S$. Then, we have that $s^n = s^m = e$. But, this implies that $n = m$, because in a cycle there is no vertex that is visited twice. Therefore we have arrived to a contradiction. □

2.5 The Frattini subgroup

In this part of the text, our main goal will be to show that every minimal Cayley graph is locally isomorphic to the minimal Cayley graph of a group with trivial Frattini group. While this is yet another property of minimal Cayley graphs, we will not make use of this result later on. To write this section we have served ourselves, principally, of the sources [11, 21].

Definition 2.31. Let G and H be two groups. Then, given a function $\phi : G \longrightarrow H$, we define the following concepts:

i) ϕ is a *group homomorphism* if, and only if:

$$\phi(g \cdot g') = \phi(g) \cdot \phi(g'), \ \forall g, g' \in G$$

ii) ϕ is an *isomorphism* if the application is a bijective homomorphism. If such a function exist, we say that G and H are *isomorphic* ($G \cong H$).

Now, consider the function $\varphi : G \longrightarrow G$, then we have the following definition:

iii) φ is an *automorphism* if the function is an isomorphism from a group to itself. Their collection is called the *automorphism group* $Aut(G)$.

Definition 2.32. A *maximal subgroup* H of a group G is a proper subgroup such that no proper subgroup K of G contains H strictly.

Lemma 2.33. *Let $f : G \longrightarrow G$ be an automorphism of a group G. Then, if M is a maximal subgroup of G, $f(M)$ will be a maximal subgroup of G as well.*

Proof. To the contrary, assume that there exists a subgroup K of G ($K \leq G$) such that $f(M) \leq K$. Since f is an automorphism, we know that $f^{-1}(K)$ will also be a subgroup of G because an automorphism maps a subgroup to another subgroup. Then, applying f^{-1} to $f(M) \leq K$, we get that $M \leq f^{-1}(K) \leq G$, which is a contradiction because M is maximal. $\square$

Definition 2.34. The *Frattini subgroup* $\Phi(G)$ of a group G is the intersection of all maximal subgroups of G. If G has no maximal subgroups then, we set $\Phi(G) = G$.

Definition 2.35. Let G be a group. We say that an element $g \in G$ is a *non-generator* of G if having $G = \langle g, S \rangle$ implies that $G = \langle S \rangle$, where S is a subset of the elements of the group.

Lemma 2.36. *If a group G is non-trivial, then the Frattini subgroup $\Phi(G)$ is the set formed by all the non-generators of G:*

$$\Phi(G) = \{g \in G \mid g \textit{ is in no minimal generating set of } G\}$$

Proof. We will start considering an element $g \in G$ and a maximal subgroup M of G such that $g \notin M$. Then, if we add to M the element g, we will get $\langle M, g\rangle = G$, since M is maximal. In addition, because of that same reason, we will have $|M| < |G|$, which means that g is an essential generator in $\langle M, g\rangle = G$. Hence, the non-generators of G must be elements of its maximal subgroups. So, as by definition, $\Phi(G) = \bigcap M$, where M represents the different maximal subgroups of G, we can affirm that the non-generators elements are contained in $\Phi(G)$.

To prove that $\Phi(G) \subseteq \{g \in G \mid g$ is in no minimal generating set of $G\}$, take an element $\varphi \in \Phi(G) = \bigcap M$. Now, suppose that φ is a generator of G. Then, there must exist a maximal subgroup M which does not contain φ. Like this, we would get again $\langle M, \varphi\rangle = G$. But, this is a contradiction because $\varphi \in \bigcap M$.

In conclusion, we have seen that $\Phi(G)$ is formed by all the non-generators of G. □

Definition 2.37. Let H be a subgroup of a group G. Then, we have the following definitions:

i) We say that H is a *characteristic subgroup* of G if for every $f \in Aut(G)$, we have that $f(H) = H$.

ii) Furthermore, H is a *normal subgroup* of G if it is invariant under conjugation. That is to say, that $g \cdot H \cdot g^{-1}$ will be in H for every $g \in G$.

Remark 2.38. In fact, although in the definition is stated that if N is a normal subgroup of G then $g \cdot H \cdot g^{-1} \subseteq H$, it is equivalent to say that $g \cdot H \cdot g^{-1} = H$. That is because, if we take an element $g \in G$ and its inverse element, we have:

$$g \cdot H \subseteq H \cdot g \qquad\qquad g^{-1} \cdot H \subseteq H \cdot g^{-1}$$

So, if we multiply on each side of H by g in the second expression:

$$g^{-1} \cdot H \subseteq H \cdot g^{-1} \implies g \cdot g^{-1} \cdot H \cdot g \subseteq g \cdot H \cdot g^{-1} \cdot g \implies H \cdot g \subseteq g \cdot H$$

Since this last inclusion is opposite to the first one we wrote down, we have arrived to the conclusion that $g \cdot H \cdot g^{-1} = H$.

Lemma 2.39. *If H is a characteristic subgroup of G, then it is also normal.*

Proof. It is a direct proof, as we just have to take an automorphism $f : G \longrightarrow G$, which we know that will satisfy that $f(H) = H$. Then, we define f as $f(h) = g \cdot h \cdot g^{-1}$ for any $g \in G$. So, since $f(H) = H$, we will clearly have $g \cdot h \cdot g^{-1} \in H$, for any $h \in H$. □

Proposition 2.40. *Let G be a group. Then, its Frattini subgroup $\Phi(G)$ is a normal subgroup of G.*

Proof. We only need to prove that $\Phi(G)$ is a characteristic subgroup of G, because that will imply the normality of the subgroup by the previous lemma. In order to see this, consider a mapping $f : G \longrightarrow G \in Aut(G)$. We will have to prove that $f(\Phi(G)) = \Phi(G)$. We can do that applying the Lemma 2.33:

$$f(\Phi(G)) = f\left(\bigcap_{M \text{ maximal in } G} M\right) = \bigcap_{f(M) \text{ maximal in } G} f(M) = \Phi(G)$$

With this we have seen that the Frattini subgroup is characteristic and, therefore, normal by Lemma 2.39. □

Definition 2.41. A connected graph $Y = (V_2, E_2)$ is a *k-covering* of a graph $X = (V_1, E_1)$ if there exists a surjective graph homomorphism $f : Y \longrightarrow X$ such that for every $v \in V_1$, the preimage $f^{-1}(v)$ has size k.

Definition 2.42. Let X and Y be two graphs. Then, we say that Y is *locally isomorphic* to X if there exists a k-covering from Y to X.

Theorem 2.43. *If a group G has Frattini subgroup Φ, then every minimal Cayley graph of G is locally isomorphic to a minimal Cayley graph of the group G/Φ.*

Proof. To start, notice that, by the Proposition 2.40, Φ is normal in G, so we can affirm that the quotient G/Φ is a group.

To continue, consider the function $\pi : G \longrightarrow G/\Phi$ defined by $\pi(g) = g \cdot \Phi$. Now, we will see that this function is a group homomorphism. First, we define the cosets as follows:

$$g \cdot \Phi = \{g' \in G \mid \exists p \in \Phi \text{ such that } g' = g \cdot p\}$$

In addition, as Φ is a normal subgroup of G, we have that $g \cdot \Phi = \Phi \cdot g$ (by Remark 2.38). Therefore, π is an homomorphism, because $\pi(g) \cdot \pi(g') = \pi(g \cdot g')$:

$$\pi(g) \cdot \pi(g') = (g \cdot \Phi)(g' \cdot \Phi) = (g \cdot \Phi)(\Phi \cdot g') = g \cdot \Phi \cdot g' = g \cdot (g' \cdot \Phi) = (g \cdot g') \cdot \Phi$$

Moreover, by the definition of $g \cdot \Phi$ we can see that π is surjective. That is because its image is equal to the set of arrival G/Φ, since for any coset $g \cdot \Phi$ in G/Φ there is the element $g \in G$ that satisfies $\pi(g) = g \cdot \Phi$.

To carry on, consider S as an irreducible generating set for G and let $Cay(G,S)$ be the corresponding minimal Cayley graph. Likewise, let $S' := \pi(S)$ be the generating set of G/Φ and let $Cay(G/\Phi, S')$ be the corresponding minimal Cayley graph. Now, we will try to show that $Cay(G,S)$ is a k-covering of $Cay(G/\Phi, S')$ where $k = |\Phi|$:

- Surjectivity: We consider $\phi : Cay(G,S) \longrightarrow Cay(G/\Phi, S')$ defined as $\phi(g) = g \cdot \Phi$. So, as shown before, this is a surjective graph homomorphism.

- Size of the preimage: We know that all cosets $g \cdot \Phi$ have the same cardinality, since we can easily define a bijection between them (the left multiplication by yx^{-1} sends $x \cdot \Phi$ to $y \cdot \Phi$). So, as $\Phi = e \cdot \Phi$ is also a coset with the neutral element of G, we can affirm that $|\Phi| = |g \cdot \Phi|$ for any $g \in G$.

  Furthermore, we know that for every $v \in V(Cay(G,S))$, $\phi^{-1}(v)$ is a set that contains all the elements $g \in Cay(G,S)$ such that $\phi(g) = v = g \cdot \Phi$.

  In addition, as $v = g \cdot \Phi$:

  $$\phi^{-1}(v) = \{h \in Cay(G,S) \mid \phi(h) = v\} = \{h \in Cay(G,S) \mid h \cdot \Phi = g \cdot \Phi\}$$

  But, $h \cdot \Phi = g \cdot \Phi$ if, and only if, $h \in g \cdot \Phi$. Which means that $\phi^{-1}(v) = g \cdot \Phi$. So, as all the cosets have the same cardinality, we have shown that $|\phi^{-1}(v)| = |\Phi|$.

In conclusion, taking $k = |\Phi|$, we have arrived to the result that $Cay(G,S)$ is a k-covering of $Cay(G/\Phi, S')$. Hence, by the Definition 2.42, we can affirm that $Cay(G,S)$ is locally isomorphic to $Cay(G/\Phi, S')$, proving like this the proposition. □

Chapter 3

On Spencer's Theorem

In one of the papers in our bibliography [1], László Babai formulates a theorem previously proved by Joel Spencer in the following way:

Theorem 3.1. (Spencer) *For every $g \geq 3$, there exists a finite graph Y of girth g such that Y is not a subgraph of any minimal Cayley graph.*

In addition to this formulation, he also mentions that it is not known whether such excluded graphs of girth 5 and degree 3 exist. It is at this point, that he suggests the question that we are working on in this project (see Question 2.5).

So, this theorem is a very important result for us because it tells us that there are prohibited finite graphs for every girth. In fact, we will see later that these prohibited graphs that we find for each girth have a bounded degree of 100. This means that, the only thing that would be left for us to know, is if we can say the same with cubic graphs (specially with the ones of girth 5). Therefore, we will attempt to go through the proof ourselves. With that purpose, we will make use of Spencer's paper [23].

First of all, observe that, the first of the two theorems that Joel Spencer formulates in his paper, is the Proposition 2.15. So, since we have already proved this preliminary theorem, we have to continue with the proof of Spencer's Theorem. To proceed, as we will be using probabilistic techniques and other concepts that we have not used yet, we will start off with some properties.

Lemma 3.2. *Let n and k be two integers such that $0 \leq k \leq n$. Then,*

$$\binom{n}{k} < \left(\frac{ne}{k}\right)^k$$

Proof. To begin, we have the following inequality:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!} \leq \frac{n^k}{k!}$$

Then, we will also use this bound for e:

$$e^k = \sum_{j=0}^{\infty} \frac{k^j}{j!} > \frac{k^k}{k!}$$

Which implies that,

$$\frac{1}{k!} < \frac{e^k}{k^k}$$

In conclusion, we have what we wanted:

$$\binom{n}{k} \leq \frac{n^k}{k!} < \left(\frac{ne}{k}\right)^k$$

□

Another Lemma that we will be needing for the proof of Spencer's Theorem is the following. We can refer to the proof in [6].

Lemma 3.3. *As before, let n and k be two integers such that $0 \leq k \leq n$. Then,*

$$\binom{n}{k} \sim 2^{H(\frac{k}{n})\cdot n}$$

Where $H(p) = -p \cdot log_2 p - (1-p) \cdot log_2(1-p)$ is the entropy function.

Finally, before starting with the proof, we will show an inequality that will be needed afterwards:

Proposition 3.4. *Let n be an integer such that $n \geq 2$. Then,*

$$1 - \left(1 - \frac{1}{n-1}\right)^{100} < \frac{100}{n}$$

Proof. For starters, let $x = 1 - \frac{1}{n-1}$. Applying this change of variable, we have $n = \frac{2-x}{1-x}$. So, we have to see that:

$$1 - x^{100} < \frac{100(1-x)}{2-x} \iff x^{101} - 2x^{100} + 99x - 98 < 0$$

Now, we can use *Octave* to see that for $x \in [0,1]$ the maximum value of $f(x) = x^{101} - 2x^{100} + 99x - 98$ takes place in $x = 1$, where $f(1) = f'(1) = 0$. Also, we can see that $f''(1) < 0$. Then, since $f(x)$ is continuous and $f'(x)$ is strictly positive in $x \in [0,1)$, we can affirm that for $0 \le x < 1$:

$$1 - x^{100} < \frac{100(1-x)}{2-x} \implies 1 - \left(1 - \frac{1}{n-1}\right)^{100} < \frac{100}{n} \text{ , for } n \geq 2$$

□

Theorem 3.5. (Spencer) *For all g, there exist a prohibited graph Y satisfying:*

i) $deg(v) \le 100, \forall v \in V(Y)$.

ii) $girth(Y) > g$.

Proof. First of all, we fix the value of g. Now, we take X to be a random 100-regular multigraph defined as follows. Take n even and satisfying:

$$n > 10^6 \cdot \sum_{s=1}^{2g-1} \frac{1}{s!} \cdot \binom{\binom{s}{2}}{s+1} \cdot 100^{s+1}$$

Then, we define the set of vertices as $V(X) = \{1, \dots, n\}$. Also, we take an arbitrary 1-factorization on $V(X)$ independently and uniformly chosen. We will refer to the 1-factors of the 1-factorization as $E_1, \dots, E_{100}$. Therefore, $E = E(X)$ is the union of the different E_r. Like this, we have built a model for a random regular graph of degree 100, even though we may have multiple edges.

To continue, we will calculate a few probabilities that will be helpful later. Notice that, for each pair of distinct vertices i, j, the probability of having $\{i,j\} \in E_r$, for any r, is $\frac{n/2}{\binom{n}{2}} = \frac{1}{n-1}$. Therefore, the probability of $\{i,j\} \in E$ is:

$$P(\{i,j\} \in E) = 1 - P(\{i,j\} \notin E) = 1 - (1 - P(\{i,j\} \in E_r))^{100} = 1 - \left(1 - \frac{1}{n-1}\right)^{100} \tag{3.1}$$

Which is slightly less than $\frac{100}{n}$ (see Prop. 3.4). So, we will be using the following bound:

$$P(\{i,j\} \in E) < \frac{100}{n}$$

In addition, we will need to know the probability of the edges $\{i_1, j_1\}, \dots, \{i_s, j_s\}$ all being in E. To do this calculation, we will consider this events as mutually independent. However, this is not exactly true (specially when s is big). Observe that if $s = 101$ and $i_1 = i_2 = ... = i_s$, the probability of $P(\{i_s, j_s\} \in E) = 1 - \left(1 - \frac{1}{n-1}\right)^{100}$.

But, $P(\{i_s, j_s\} \in E \mid \{i_1, j_1\}, \ldots, \{i_{s-1}, j_{s-1}\} \in E) = 0$, because the maximum degree of the graph is 100. This shows that the events are not independent because $P(\{i_s, j_s\} \in E) \neq P(\{i_s, j_s\} \in E \mid \{i_1, j_1\}, \ldots, \{i_{s-1}, j_{s-1}\} \in E)$. However, this effect is minimal enough to treat the events as approximately independent for the purpose of the analysis. Moreover, although the bigger it is s, the bigger will we be the effect on the calculations, the only time that we will be using this, will be with the value of s relatively small. So, we do not need to worry about this. Then, we get by 3.1:

$$P(\{i_1, j_1\}, \ldots, \{i_s, j_s\} \in E) = P(\{i_1, j_1\} \in E) \cdots P(\{i_s, j_s\} \in E) \leq \left(\frac{100}{n}\right)^s \tag{3.2}$$

Calculation of P(ISC):

To continue, we will denote by **ISC** the event of X containing two cycles of length at most g that intersect at least at one vertex. Now, so that we can calculate $P(\mathbf{ISC})$, imagine that **ISC** is true in X. Then, this means that there are two cycles C_1 and C_2 of length at most g that share at least one vertex. Therefore, the subgraph $C_1 \cup C_2$ has at most $g + g = 2g$ vertices. We will denote by $s < 2g$ the number of vertices of $C_1 \cup C_2$. Then, in $C_1 \cup C_2$ we will have at least $s + 1$ edges.

Now, we will calculate the probability that these two cycles exist. To do so, notice that there are $\binom{n}{s}$ possible sets of s vertices in X. Then, once we have chosen the vertices, there will be $\binom{s}{2}$ possible edges connecting the s vertices in X. And, from these edges we want to take the $s + 1$ edges of $C_1 \cup C_2$. So, we have $\binom{\binom{s}{2}}{s+1}$ possible sets of $s + 1$ edges connecting the s vertices of $C_1 \cup C_2$. Furthermore, these fixed edges would have a probability of being in X of at most $\left(\frac{100}{n}\right)^{s+1}$ by 3.2. So,

$$P(\mathbf{ISC}) < \sum_{s=1}^{2g-1} \binom{n}{s} \cdot \binom{\binom{s}{2}}{s+1} \cdot \left(\frac{100}{n}\right)^{s+1} < \sum_{s=1}^{2g-1} \frac{1}{s!} \cdot \binom{\binom{s}{2}}{s+1} \cdot \frac{100^{s+1}}{n}$$
$$< \frac{1}{n} \cdot C_g \quad \text{, where } C_g = \sum_{s=1}^{2g-1} \frac{1}{s!} \cdot \binom{\binom{s}{2}}{s+1} \cdot 100^{s+1}$$

Then, with the objective of having $P(\mathbf{ISC}) < 10^{-6}$, we set $n > 10^6 \cdot C_g$ as stated in the beginning of the proof.

Calculation of P(FE):

Now that we have seen that the event **ISC** is highly unlikely for large enough values of n, our goal is to show that X can not be split into two different parts so that there are few edges between them. In order to prove that, we will denote as **FE** the event of the existence of a set Z with $t \leq \frac{n}{2}$ vertices such that there are less than $4t$ edges connecting Z to Z^c. Thus,

$$P(\mathbf{FE}) \leq \sum_{t=1}^{\frac{n}{2}} \binom{n}{t} \alpha_t \tag{3.3}$$

Where we are using α_t as the representation of the probability that, for a particular Z such that $V(Z) = \{1, \dots, t\}$, there are fewer than $4t$ edges between Z and Z^c. Moreover, from now on, we will think of each random 1-factor E_r as being constructed in the following way:

1. For every available element $1 \leq i \leq t$ taken in order, we will select an uniformly chosen partner from those elements of X that have not been paired off yet.

2. To continue, as we can connect i with any vertex of X (not only with the ones of Z), at least $\frac{t}{2}$ of the vertices of Z will not be already linked when we get to them.

3. When we reach such an i, let u ($u < i$), denote the number of points $j < i$ that have already been paired off with a vertex of $x \in Z^c$.

Then, at this point, we will have $n - t - u$ available vertices in Z^c and at most $n - 2u$ altogether. So, the probability of the vertex i being connected with an element $x \in Z^c$ is at least $\frac{n-t-u}{n-2u}$, which can be bounded as follows:

$$P(\{i, x\} \in E_r) \geq \frac{n-t-u}{n-2u} = \frac{n-2u-t+u}{n-2u} = 1 + \frac{u-t}{n-2u} \geq 1 - \frac{t}{n-2u} \geq 1 - \frac{t}{n}$$

Then, what we will have is 100 independent random 1-factors on which, we will have at least $100 \cdot \frac{t}{2} = 50t$ occasions in which an edge, linking Z with Z^c, will be created with a probability of $1 - \frac{t}{n}$ or more, as we have just seen. Hence, what we have now is that $\alpha_t \leq P(B < 4t)$, where B represents the number of edges going from Z to Z^c. B follows a binomial distribution: $B \sim B(50t, 1 - \frac{t}{n})$, since we have $50t$ independent repetitions of the Bernoulli experiment consisting in having an edge connecting Z to Z^c or not. Therefore, we have that $\alpha_t < \binom{50t}{4t} \cdot (\frac{t}{n})^{46t}$, since:

$$\begin{aligned} P(B < 4t) &= \sum_{k=0}^{4t-1} \binom{50t}{k} \cdot \left(1 - \frac{t}{n}\right)^k \cdot \left(1 - \left(1 - \frac{t}{n}\right)\right)^{50t-k} \\ &= \sum_{k=0}^{4t-1} \binom{50t}{k} \cdot \left(1 - \frac{t}{n}\right)^k \cdot \left(\frac{t}{n}\right)^{50t-k} \\ &< \binom{50t}{4t} \cdot \left(\frac{t}{n}\right)^{46t} \end{aligned}$$

Before continuing, we will show that this last inequality is true:

$$d = \frac{\sum_{k=0}^{4t-1}\binom{50t}{k}\cdot\left(1-\frac{t}{n}\right)^{k}\cdot\left(\frac{t}{n}\right)^{50t-k}}{\binom{50t}{4t}\cdot\left(\frac{t}{n}\right)^{46t}} = \frac{\sum_{k=0}^{4t-1}\binom{50t}{k}\cdot\left(1-\frac{t}{n}\right)^{k}\cdot\left(\frac{t}{n}\right)^{4t-k}}{\binom{50t}{4t}}$$

Now, applying that $t \leq \frac{n}{2}$ and simplifying the binomial coefficients, we obtain what we wanted:

$$d \leq \sum_{k=0}^{4t-1}\frac{(4t)!\cdot(46t)!}{k!\cdot(50t-k)!}\cdot\left(\frac{1}{2}\right)^{4t} \leq 4t\cdot\frac{(4t)!\cdot(46t)!}{(4t)!\cdot(46t)!\cdot 2^{4t}} = \frac{4t}{2^{4t}} < 1$$

To go on, we will also use the following two bounds that we get from the Lemmas 3.2 and 3.3:

$$\binom{n}{t} < \left(\frac{ne}{t}\right)^{t}, \quad \binom{50t}{4t} \sim 2^{H(0.08)\cdot 50t} < 2^{21t}$$

Then, applying to 3.3, these last two bounds in addition to $\alpha_t < \binom{50t}{4t}\cdot\left(\frac{t}{n}\right)^{46t}$ and $t \leq \frac{n}{2}$, we get:

$$P(\mathbf{FE}) < \sum_{t=1}^{\frac{n}{2}}\left(\frac{ne}{t}\right)^{t}\cdot 2^{21t}\cdot\left(\frac{t}{n}\right)^{46t} = \sum_{t=1}^{\frac{n}{2}}\left[e\cdot 2^{21}\left(\frac{t}{n}\right)^{45}\right]^{t} \leq \sum_{t=1}^{\frac{n}{2}}\left(e\cdot 2^{-24}\right)^{t} < 10^{-6}$$

Finally, the existence of two intersecting cycles is independent of the existence of a set Z with $t \leq \frac{n}{2}$ vertices such that there are less than $4t$ edges connecting Z to Z^c. Then, we have obtained that the events **ISC** and **FE** have a probability of 0.999998 of being both false. Which means that, we can ensure that it is very probable that there exists a graph X that satisfy both of the following properties:

i) X does not have two cycles of length at most g that intersect in a vertex.

ii) In X, all sets Z with $t \leq \frac{n}{2}$ vertices have at least $4t$ edges connecting Z to Z^c.

Now, let Y be a subgraph of X with all edges in cycles of length smaller or equal to g removed. This clearly implies that the girth of Y is greater than g. To continue, we can take a set Z of $t \leq \frac{n}{2}$ vertices, which will have at least $4t$ edges connecting with Z^c, by ii). Moreover, for every vertex $v \in V(Y)$, there will be at most two edges from v that are in X but not in Y. This is because the cycles that have been removed in Y do not share any vertex, by i). This implies that we have deleted at most two edges for every vertex. As a consequence, in Y there will be at least $4t - 2t = 2t$ edges connecting Z to Z^c. So, by the Theorem 2.15, Y is prohibited. □

After taking a look at this demonstration, one may be thinking that instead of bounding the degree by 100, we could use 3 and that would solve our problems.

However, this argument does not work with the degree bounded by 3. Even if we try to bound the degree of the graph by 4, we are not able to bound the probability of α_t by any relevant value. In fact, if we go back to the calculation of $P(B < 4t)$, observe that since in this case we only have 4 independent random 1-factors, we have at least $4 \cdot \frac{t}{2} = 2t$ occasions in which an edge linking Z and Z^c will be created with probability $1 - \frac{t}{n}$. So, using this minimum value of $2t$ occasions, we will clearly have that $P(B < 4t) = 1$ because $2t < 4t$. If we use 3 instead of 4, it happens the same thing, but we have used this case to avoid non-integer values, since we would obtain $\frac{3t}{2}$ occasions in the case of degree bounded by three.

Even if we try to bound the degree by 50 we only obtain that $P(\mathbf{FE}) \leq 0,21$. This gives us a probability of 0,79 of being **ISC** and **FE** both false, and this value is not comparable with the one obtained in the proof. Also, we could try to reduce the maximum degree to 98 (to avoid fractions with 99), and we would obtain a probability of 0.997. So, the only problem with these last two cases is that the probability is not as strict as it is in the shown proof.

The question now is: can we follow a different strategy to ensure the maximum degree is bounded by 3, or at least by a significantly smaller value than 100? This is likely the question Babai asked himself in the referenced paper [1]. Perhaps he believed that the bound used in the theorem could be improved.

Chapter 4

Generalized Petersen graphs

We have mentioned several times that Babai asks in [1] whether all cubic graphs of girth 5 are subgraphs of minimal Cayley graphs. Right after this, he also mentions that the Petersen graph is subgraph of a minimal Cayley graph of group order 20. Motivated by this example, we study his question with respect to the family of generalized Petersen graphs.

4.1 Definition and properties

We start this section with the definition of the family of generalized Petersen graphs. Although the standard notation for this family is $G(n,k)$, here we will use $P(n,k)$ to avoid confusions with groups.

Definition 4.1. The *generalized Petersen graphs* are a family of graphs formed by connecting the vertices u_i of a regular polygon to the corresponding vertices v_i of a star polygon. We denote these graphs by $P(n,k)$ and we define them as follows:

$$V = \{u_0, u_1, \dots, u_{n-1}, v_0, v_1, \dots, v_{n-1}\}$$

$$E = \{\{u_i, u_{i+1}\}, \{u_i, v_i\}, \{v_i, v_{i+k}\} \mid 0 \leq i \leq n-1\}$$

Where the subscripts of the vertices should be read modulo n. Also, k is defined so that it satisfies the condition $k < \frac{n}{2}$.

Remark 4.2. All generalized Petersen graphs are cubic.

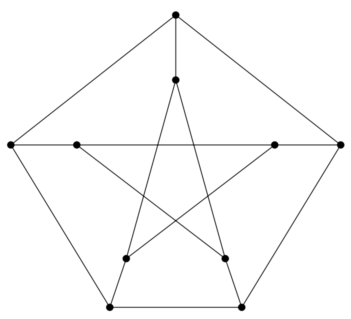


Figure 4.1: Petersen graph or $P(5,2)$.

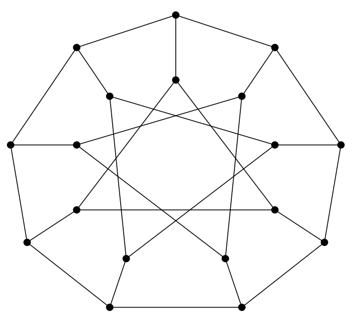


Figure 4.2: $P(9,3)$.

Previously, in Section 2.3, we asked ourselves if every cubic graph has a perfect matching which can be used as a no lonely edge coloring. Although we discarded that option easily, observe that we can use this type of coloring to quickly obtain a no lonely coloring for all generalized Petersen graphs. This is the strategy that we will use to prove the following result:

Proposition 4.3. *If $X = P(n,k)$ is a generalized Petersen graph, then X is a no lonely color graph.*

Proof. To start, we will denote C_n as the outer cycle of X. That is to say, the regular polygon to which we refer in the previous definition. In addition, we will denote by $S_1, \dots, S_r$ where $r \geqslant 1$, the different cycles that are usually represented in the inner part of X and which are referred in the definition as the star polygons. Finally, we will also denote by E' the edges that connect the cycle C_n to $S_1, \dots, S_r$.

In order to show that X is a no lonely color graph, we will paint the edges with two different colors: h_1 and h_2. All edges in E' will have the color h_1 and the rest will be painted with h_2. Now, observe that E' is a perfect matching and, since X is a cubic graph, the condition that every vertex has to be incident with at most two edges of any color is satisfied.

Now, observe that the subgraph of X that only contains the edges of E' is disconnected and all its edges have their endpoints in different components, since E' is a perfect matching. And it happens the same with the subgraph that only contains the edges $E(X) \setminus E'$. So, by the Proposition 2.14, the defined coloring is a no lonely coloring of X. $\square$

Nevertheless, remember that this is not a sufficient condition to be a subgraph of a minimal Cayley graph. A good example is the Dürer graph: $P(6,2)$. If we use the coloring of the proof, we obtain 3 monochromatic cycles: one of length 6 and the other two of length 3, all colored with the same color. Then, by the Lemma

2.30, this coloring is not good for a minimal Cayley graph. And it happens the same, for example, with $P(8,2)$ or $P(10,2)$.

So, at the end, this result does not help us to answer our main question. This means that we will have to deal with this matter with a different approach.

4.2 The Generalized Petersen Graphs that are minimal Cayley graphs

To start, a good option is to show which generalized Petersen graphs are minimal Cayley graphs, which may seem a simpler option in comparison to check if they are subgraphs. To do that, we will follow a proof given by Marko Lovrečič Saražin in [22]. But, we will start with some definitions:

Definition 4.4. A graph X is said to be *vertex-transitive* if, for every vertex v_1 and v_2 of the graph, there is an automorphism $\varphi \in Aut(X)$ such that:

$$\varphi(v_1) = v_2$$

Definition 4.5. Let X be a graph. Then, a subgroup K of $Aut(X)$ is *fixed-point-free* if the only automorphism in it with a fixed point is $\{id\}$.

Definition 4.6. Let $X = (V, E)$ be a graph and K a subgroup of $Aut(X)$. Then, K is a *regular subgroup* if it is fixed-point-free and acts transitively on V.

Now, once we have these definitions, we can continue with the formulation of Sabidussi's Lemma. This result will be essential to arrive to the consequences that we see in [22]. For the understanding of this theorem's proof, we have used the book [13].

Lemma 4.7. (Sabidussi) *A connected graph $X = (V, E)$ is a Cayley graph if and only if, $Aut(X)$ contains a regular subgroup acting on V.*

Proof. For the first direction, assume that $X = Cay(G, S)$ for any group G and any generating set S. This implication is a consequence of the Lemma 2.26, since for every two elements $g, h \in G$, we always have the automorphism $\lambda_{hg^{-1}}$, which maps g to h:

$$\lambda_{hg^{-1}}(g) = (hg^{-1})g = h$$

Then, we can affirm that the family of the left-multiplication mappings λ_g, where $g \in G$, is a subgroup that acts transitively on the vertices of X.

Now, to see that this subgroup is also fixed-point-free, we will suppose that there exists some arbitrary elements $g, h \in G$ such that $\lambda_g(h) = h$. Then,

$$gh = h \iff g = hh^{-1} \iff g = e$$

But, λ_e is the identity function id. So, we have seen that the mentioned subgroup is fixed-point-free and, hence, regular by the Definition 4.6.

For the converse direction, assume that we have a regular subgroup G of $Aut(X)$. Also, fix an arbitrary vertex $e \in V$ as the identity element of G. Now, since G is regular, we can affirm that, for any $v \in V$, there is an unique automorphism $\phi_v \in G$ that maps e to v: $\phi_v(e) = v$. Then, we can define a set S as:

$$S = \{\phi_v \mid \phi_v(e) = v\}$$

As a consequence, we can identify each vertex of the graph X with the elements of G and we obtain the Cayley graph $X = Cay(G, S)$, where S is the generating set and two elements $x, y \in X$ will be adjacent if and only if $\phi_x^{-1} \circ \phi_y \in S$. $\square$

As we said, this will be the main theorem needed to prove which generalized Petersen graphs are minimal Cayley graphs and which are not. Therefore, we need to bring our attention to the automorphism groups $Aut(P(n,k))$. Observe that α and β are clearly two elements of the group:

$$\begin{aligned} \alpha(x_i) &= x_{i+1} & \qquad \beta(x_i) &= x_{-i} \\ \alpha(y_i) &= y_{i+1} & \qquad \beta(y_i) &= y_{-i} \end{aligned}$$

In addition, we can define a permutation of vertices γ such that $\gamma(x_i) = y_{ki}$ and $\gamma(y_i) = x_{ki}$. This mapping will also be an automorphism of $P(n,k)$ if $k^2 \equiv \pm 1 \pmod{n}$. To prove that γ is an automorphism, we have to see first that is a bijection. With this purpose, we can easily obtain the injectivity. Then, the surjectivity is a direct consequence of having $k^2 \equiv \pm 1 \pmod{n}$, since this implies that $gcd(n,k) = 1$. Also, taking the inverse function $\gamma^{-1}(x_i) = y_{ki}$ and $\gamma^{-1}(y_i) = x_{ki}$ for the case $k^2 \equiv +1 \pmod{n}$ and $\gamma^{-1}(x_i) = y_{-ki}$ and $\gamma^{-1}(y_i) = x_{-ki}$ for the case $k^2 \equiv -1 \pmod{n}$, we can easily check that γ^{-1} is also an homomorphism.

Example 4.8. Here we have two examples of the transformation obtained after applying the automorphism γ:

i) Example for $k^2 \equiv 1 \pmod{n}$

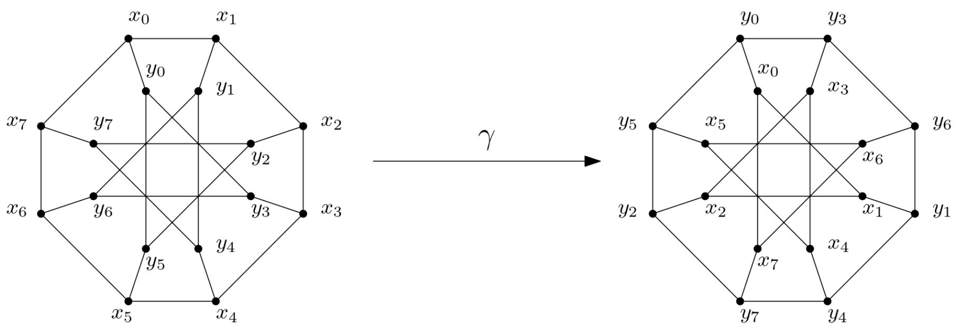

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7
γ
y_0 y_3 y_6 y_1 y_4 y_7 y_2 y_5
x_0 x_3 x_6 x_1 x_4 x_7 x_2 x_5

Figure 4.3: Application of γ to the vertices of the graph $P(8,3)$.

ii) Example for $k^2 \equiv -1 \pmod{n}$

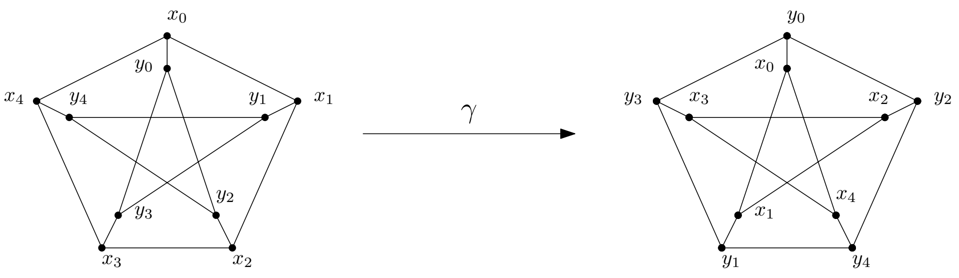

x_0 x_1 x_2 x_3 x_4
y_0 y_1 y_2 y_3 y_4
γ
y_0 y_2 y_4 y_1 y_3
x_0 x_2 x_4 x_1 x_3

Figure 4.4: Application of γ to the vertices of the Petersen graph ($P(5,2)$).

To continue, we will formulate a theorem that is proved in [8]:

Theorem 4.9. *A generalized Petersen graph $P(n,k)$ is vertex-transitive if and only if, either $k^2 \equiv \pm 1$ (mod n) or $(n,k) = (10,2)$.*

Then, if (n,k) is not one of the following seven cases: $(4,1)$, $(5,2)$, $(8,3)$, $(10,2)$, $(10,3)$, $(12,5)$ or $(24,5)$, we have the following automorphism group:

i) If $k^2 \equiv 1$ (mod n):

$$Aut(P(n,k)) = \langle \alpha, \beta, \gamma \mid \alpha^n = \beta^2 = \gamma^2 = id,\ \beta\alpha\beta = \alpha^{-1},\ \gamma\beta = \beta\gamma,\ \gamma\alpha\gamma = \alpha^k \rangle \tag{4.1}$$

ii) If $k^2 \equiv -1$ (mod n):

$$Aut(P(n,k)) = \langle \alpha, \gamma \mid \alpha^n = \gamma^4 = id,\ \gamma\alpha\gamma^{-1} = \alpha^k \rangle \tag{4.2}$$

As we can see, the Theorem 4.9 tells us which generalized Petersen graphs are candidates for being minimal Cayley graphs according to the lemma of Sabidussi.

In addition, it also describes its automorphism groups and leaves us with a set of seven exceptions that we will have to study separately.

To do so, we can observe that the minimal Cayley graph of $P(4,1)$ can be identified with the dihedral group D_4. Furthermore, we can take the minimal Cayley graph of $P(8,3)$ as the Pauli group G_1 as stated in [12]. Then, as we can see in [7], $P(12,5)$ is the minimal Cayley graph of the symmetric group of permutations on four elements S_4. And, finally, $P(24,5)$ is the minimal Cayley graph of the group $GL(2,3)$ as seen in [5].

On the other hand, the remaining three graphs are not minimal Cayley graphs. The proof concerning the Petersen graph $P(5,2)$ is shown in the Theorem 2.4 of [9], for $P(10,3)$ we have the proof in [24] and, lastly, we can find the proof of $P(10,2)$ in the same paper that we are following [22]. This leaves us with the following proposition:

Proposition 4.10. *The generalized Petersen graphs $P(4,1)$, $P(8,3)$, $P(12,5)$ and $P(24,5)$ are minimal Cayley graphs. Contrary to the graphs $P(5,2)$, $P(10,3)$ and $P(10,2)$, which are not (minimal) Cayley graphs.*

Once, we have arrived at this point, it is the moment to prove the main result that characterize some generalized Petersen graphs as minimal Cayley graphs. Observe that the Lemma of Sabidussi (Lemma 4.7), only allow us to know if a graph is Cayley, but it does not give us information about if it is minimal. However, this will not stop us from obtaining the following result:

Theorem 4.11. *Let $X = P(n,k)$ be an arbitrary generalized Petersen graph. Then, X is a minimal Cayley graph, if and only if, $k^2 \equiv 1$ (mod n).*

Proof. To start, assume that X is not one of the previous exceptions (if it is, the case has been considered in the Proposition 4.10). Then, we have the automorphism group that is represented on the Theorem 4.9. Now, consider the subgroup $B = \langle \alpha, \gamma \rangle$. According to (4.1), $\alpha\gamma = \gamma^{-1}\alpha^k$. So, we can affirm that $|B| = 2n$, which are the number of vertices of the graph, if we keep in mind that $\alpha^n = \gamma^2 = id$.

Therefore, we will only need to consider the images of the vertex x_o because, by Theorem 4.9, X is vertex-transitive:

$$\alpha^i(x_0) = x_i \qquad\qquad \alpha^i(\gamma(x_0)) = y_i$$

And, since $|B| = 2n$, we can confirm that all the images are different.

Then, B is a regular subgroup of $Aut(X)$, so the graph X is a Cayley graph of the form $Cay(G,S)$, by the Theorem of Sabidussi. To continue, we will show that this Cayley graph is in fact minimal.

Notice that the group representation of the Cayley graph $X = Cay(G, S)$ is:

$$\langle \alpha, \gamma \mid \alpha^n = \gamma^2 = e,\ \gamma\alpha\gamma = \alpha^k \rangle$$

In this case S is of order two. Hence, any subset of S would generate a cyclic group. However, G is not cyclic, which implies that S is irreductible. Thus, X is a minimal Cayley graph. □

4.3 Some generalized Petersen graphs that are subgraphs of minimal Cayley graphs

As we have mentioned before, it is not an easy question to show that a graph is a subgraph of a minimal Cayley graph. So, we will not be able to formulate a general theorem that includes all generalized Petersen graphs. However, we can try to show it for a certain selection of that family.

With that objective, we will start defining the semidirect product, which will help us in our question.

Definition 4.12. Let H and K be any two groups and $\phi : K \longrightarrow Aut(H)$ a group homomorphism. This homomorphism characterizes an action of the group K on H given by $\phi_k(h) = khk^{-1}$, for all $k \in K$ and $h \in H$.

Then, we call the *semidirect product* of H and K with respect to ϕ (denoted as $H \rtimes_\phi K$) to the group formed by the pairs:

$$\{(h,k) \mid h \in H,\ k \in K\}$$

Under the operation defined by: $(h_1, k_1) \cdot (h_2, k_2) = (h_1 \cdot \phi_{k_1}(h_2), k_1 \cdot k_2)$.

Before the formulation and the demonstration of the proposition which is the main goal of this section, we will recall briefly the Euler's phi function and the Euler's theorem, so that we can use them later. We can see a proof of this Theorem, for example, in [15].

Definition 4.13. The *Euler's phi function* φ counts the natural numbers up to a given $n \in \mathbb{N}_{\geq 2}$ that are coprime with n.

Theorem 4.14. (Euler) *If $k \in \mathbb{Z}$, $n \in \mathbb{N}_{\geq 2}$ and $gcd(k, n) = 1 \Longrightarrow k^{\varphi(n)} \equiv 1$ (mod n), where φ is the Euler's phi function.*

In addition, we formulate a lemma that will also help us in the proof of the main result of the section. To prove this statement, we have helped ourselves of the book [19] of our bibliography.

Lemma 4.15. *Let n, $k \in \mathbb{N}^+$ such that $gcd(n,k) = 1$. In addition, let C_n and $C_{\varphi(n)}$ be two cyclic groups with generators a and b, respectively. Then, under these conditions, the group representation of the semidirect product of C_n and $C_{\varphi(n)}$ is:*

$$C_n \rtimes C_{\varphi(n)} = \langle a, b \mid a^n = e,\ b^{\varphi(n)} = e,\ bab^{-1} = a^k \rangle \tag{4.3}$$

Proof. To start, since $gcd(n,k) = 1$, we know that a^k also generates C_n. So, we can take an automorphism $\alpha \in Aut(C_n)$, defined as $\alpha(a^i) = a^{ki}$ for all $0 \leq i < n$. Then, by the Theorem of Euler (Thm 4.14), we have that $k^{\varphi(n)} \equiv 1 \pmod{n}$. So, we obtain $\alpha^{\varphi(n)}(a^i) = a^{k^{\varphi(n)}i} = a^i$, which means that $\alpha^{\varphi(n)} = id$.

Now, we can define a group homomorphism $\theta : C_{\varphi(n)} \longrightarrow Aut(C_n)$ defined as $\theta_b = \alpha$. This is the homomorphism of the semidirect product $C_n \rtimes C_{\varphi(n)}$. Therefore, by the Definition 4.12, if we write a for (a, e) and b for (e, b) (where e is the neutral element), we have the following equality:

$$ba = (e, b) \cdot (a, e) = (\theta_b(a), b) = (\alpha(a), b) = (a^k, b)$$

$$a^k b = (a^k, e) \cdot (e, b) = (a^k, b)$$

Therefore, $ba = a^k b$. Hence, we have seen that $C_n \rtimes C_{\varphi(n)}$ is generated by a and b satisfying the relations:

$$a^n = e, \qquad b^{\varphi(n)} = e, \qquad bab^{-1} = a^k$$

□

Proposition 4.16. *Let $X = P(n,k)$ be a generalized Petersen graph with $gcd(n,k) = 1$. Then, X is an induced subgraph of the minimal Cayley graph:*

$$C_n \rtimes C_{\varphi(n)} = \langle a, b \mid a^n = e,\ b^{\varphi(n)} = e,\ bab^{-1} = a^k \rangle \tag{4.4}$$

Proof. In this proof, we will have to show that $P(n,k) \subseteq Cay(C_n \rtimes C_{\varphi(n)}, \{a, b\})$. To start, we know that we have a cycle of length n that is generated by a. That is because in the group representation we have that $a^n = e$.

Besides, if we apply b to each one of the vertices of the cycle a^i, $i = 0, \ldots, n-1$, we arrive to $a^i b$.

Now, we will take a look at these vertices. Observe that, for $i = 0$, we will be in the vertex $a^0 b = b$. But, what happens if we apply the generator a to b? That, since the third condition of the group representation is equivalent to $ba = a^k b$, we can see that from b we will get to $a^k b$ (see Figure 4.5). And, if we continue applying a to $a^k b$ we get to $a^{2k} b$ for the same reason. And so on, until we arrive to $a^{nk} b = b$, which will close the cycle.

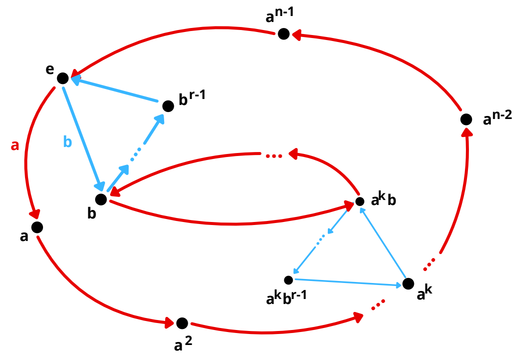


Figure 4.5: Simplified representation of $C_n \rtimes C_{\varphi(n)}$, where $r = \varphi(n)$.

We can see that, this last cycle, will correspond to the star polygon inside the generalized Petersen graph. Hence, we have already found $P(n,k)$ in $Cay(C_n \rtimes C_{\varphi(n)}, \{a, b\})$. □

In Figure 4.6, we can see an example of a Cayley graph of the semidirect product of two cyclic groups.

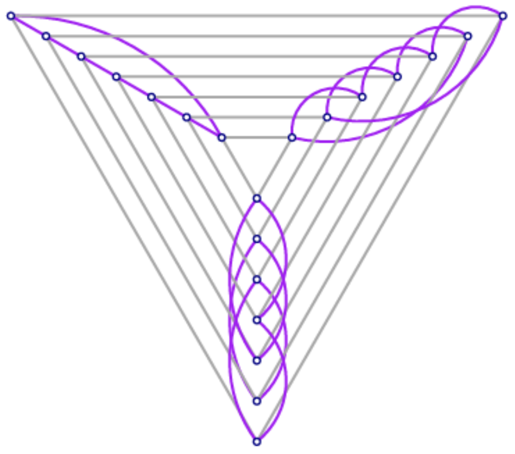


Figure 4.6: Representation of the Cayley graph $Cay(C_3 \rtimes C_7, \{(0,1),(1,0)\})$ obtained from [11].

As a consequence of the last proof, we can now affirm that all generalized Petersen graphs $P(n,k)$, which have $gcd(n,k) = 1$, are subgraphs of a minimal

Cayley graph. But, what about the other generalized Petersen graphs? In the section 5.4 we will show some more generalized Petersen graphs that are subgraphs of minimal Cayley graphs. For now, however, it is time to readjust our focus to all cubic graphs. In the next chapter, we will describe the methods we have used to search for prohibited cubic graphs.

Chapter 5

In search of prohibited cubic graphs

At this point, once we have deeply studied minimal Cayley graphs and their properties, we can now start the search of prohibited cubic graphs. With this objective of finding these cubic graphs that are not subgraphs of minimal Cayley graphs, we have coded a few programs that are based on some lists of graphs that we have found on the internet. The first files that we have used, consist on cubic graphs divided by its girth and number of vertices. We have obtained them from the web *House of Graphs* [16]. Then, we have also used a list of files, produced by Rhys J. Evans, that contain all minimal Cayley graphs on up to 255 vertices. These results have been independently verified by Kolja Knauer up to 127 vertices. We can obtain these graphs up to 95 vertices on [17].

However, your question may now be: which strategy did we follow to search these prohibited graphs? Well, since it is not easy to ensure whether a graph is prohibited or not, what we have done is to check if these graphs have a no lonely coloring. Remember that, according to the Proposition 2.9, the no lonely color property is an essential condition to be a subgraph of a minimal Cayley graph.

Nevertheless, searching for a no lonely coloring is not an easy task either, because there is a huge amount of possible colorings for each graph. So, how can we make it more efficient? Our strategy has been to first consider the perfect matchings as colorings. This is a method that we mentioned in the section 2.3. This strategy has helped a lot to increase the efficiency of the algorithm and, although we know that if we do not find a no lonely coloring with the perfect matchings, it is not enough to set this graph as prohibited, it will help to decrease the number of graphs to check.

The main functions of the code, programmed with *SageMath*, are in the section A.1 of the appendix. But the idea of the algorithm is the following:

- For every perfect matching, we apply a DFS method to check all the cycles of the graph. Recall that, we do not need to check the condition (i) of the Definition 2.7 by the way a perfect matching is constructed.
- Once we are checking the cycles, if we find two edges of the perfect matching, we stop the search on that path. This is because, if we find a lonely color edge in a cycle, that edge has to be part of the matching. The same argument has been used in the proof of the Proposition 4.3. This helps a lot to increase the efficiency of the program, because an algorithm that goes through all the cycles of a graph has a high complexity in terms of time.
- If we find a no lonely coloring, we can discard that graph and continue with the next one.
- If we find a cycle with a lonely color edge, we will have to check another perfect matching. But, if all the perfect matchings have been examined without finding a no lonely coloring, we will save the graph for a later exhaustive search on all possible colorings.

With this algorithm, we have managed to find some candidates of being prohibited graphs. But, with each one of these graphs, we will have to check which one of them is really prohibited. Because remember that we have just tried some possible colorings. It will be once we have checked all the possibilities that we will have the right to affirm that it is not a no lonely color graph. But, how can we do that? Because the computational complexity of checking all possible colorings is very high.

5.1 Going through all the colorings

Once we have checked all the perfect matchings, it is time to go through all the possible colorings of the graphs that have been set as candidates. To do that, we have used a backtracking algorithm. The recursive function `checkcolorings_bactrack` is the one that contains the code of the backtracking and, to make it as efficient as possible in terms of time and memory, we have used the following strategy.

First, it is important to mention that our strategy focuses on coloring the edges one color at a time. Once we determine that a color might be correct, we proceed to the next one. However, we do not use multiple colors simultaneously.

We are using three different graphs in the algorithm:

- `G`: `Graph` object that saves the information of the graph. Including the edges that are colored (with the label).

- `G_aux`: `Graph` object that only contains the edges that are not colored with the current color.
- `G_coloredEdges`: `Graph object` that only contains the colored edges.

In the algorithm, instead of going through all the edges that are uncolored, we check if there is a cycle that contains a lonely uncolored edge. If we do not find one, we search for lonely color edges in a cycle that have the current color. The following scheme helps to clear out this step of the algorithm. However, the hole algorithm can be seen in the section A.1.

- `If we find a lonely uncolored edge in a cycle:`
  We paint it with the current color. Remember that it is not possible that we need to paint it with a previous one because, before continuing with a different label, we check if that color is correct based on the Proposition 2.14.
- `Else:`
  We examine the cycles that contain a lonely edge painted with the current color.
  - `If we find such a cycle:`
    We return the uncolored edges so that we only consider those in the next backtracking step. Observe that, if it finds such a cycle with all edges colored, the algorithm will return no edges and the backtracking will consider this coloring as false.
  - `Else:`
    If the current color is marked as correctly colored, we change of color. If not, we just start the next step of the backtracking with all the uncolored edges.

This procedure helps a lot to reduce the number of edges that we check in the backtracking at each state of the graph. The algorithm that goes through the cycles of the graph in this step, is very similar to the one that we used before with the perfect matchings.

Then, for each one of these edges that we have selected, if we have not explored that coloring before, we verify if the degree of the endpoints is correct in terms of color (see (i) of Definition 2.7). Also, we verify if there has been a monochromatic cycle created and we check the lengths of those cycles according to the Proposition 2.30. Finally, we examine how the next step of the backtracking will be:

- If the edges that have the current color are well colored based on the Proposition 2.14 and Lemma 2.30, the next step will be to try to use a different color with the remaining edges.

- If that is not the case, we will have to continue trying with the same color.

Issues that we have encountered

Although with almost all graphs we can find a solution in less than 1 second, there is a point where we start having some issues with the memory. The thing is that, once we get to the graphs with 22 vertices, we have encountered a few isolated graphs to which the algorithm is not able to find a solution before running out of memory.

Remember that the objective is to find a graph that does not have one of these solutions, but if it runs out of memory before finding one, it means that the algorithm will never end with such big graphs. Nevertheless, this does not mean that we have to surrender. On the contrary. This suggests that the graph may not have any solution. So, in this case, we have a few options to try:

1. One thing that we can try to do is to color it ourselves by hand. We have used this method a couple of times using two colors and, once we have found a solution, we use a simple algorithm to check if the solution is correct. This algorithm is not in the appendix, but is relatively simple and is based on the Proposition 2.14.

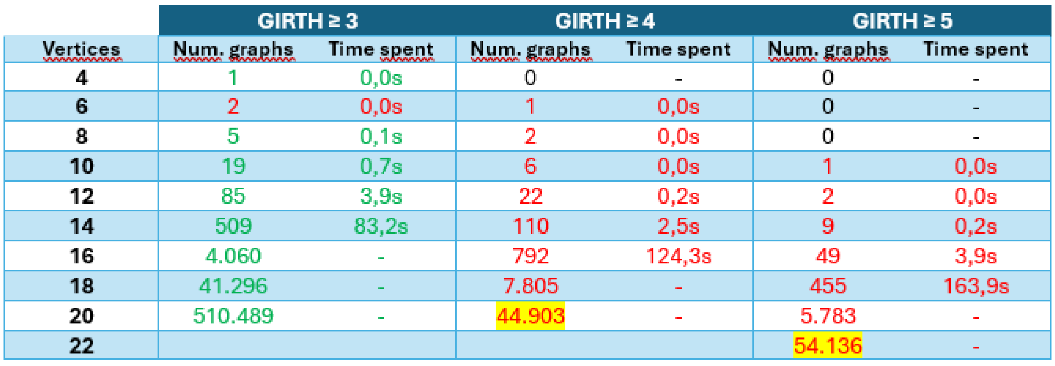
2. Another option is to check directly if this graph is a subgraph of a minimal Cayley graph. We can do that using the list of minimal Cayley graphs up to 255 vertices mentioned before.

3. Finally, if none of these previous options work, another option is to color it just using two different colors. Like this, we reduce the number of possible colorings.

With this strategy, we have been able to find prohibited graphs with girth 3, without any problem. One example is the K_4 graph, which has been mentioned before many times (see Section 2.2).

However, we have not had the same success with graphs of girth greater than 3. Despite examining all graphs, with girth 4 or greater, up to 18 vertices, each had at least one solution. Additionally, we analyzed all graphs with 20 vertices and girth 5, yielding the same result. The number of graphs analyzed is summarized in Figure 5.1.

In this picture we can see the number of graphs checked for each girth and number of vertices. Next to that, we have the time that the algorithm has spent checking those graphs. The numbers appear in green if we have found a prohibited graph and in red otherwise. Also, we see some numbers highlighted



	GIRTH ≥ 3		GIRTH ≥ 4		GIRTH ≥ 5	
Vertices	Num. graphs	Time spent	Num. graphs	Time spent	Num. graphs	Time spent
4	1	0,0s	0	-	0	-
6	2	0,0s	1	0,0s	0	-
8	5	0,1s	2	0,0s	0	-
10	19	0,7s	6	0,0s	1	0,0s
12	85	3,9s	22	0,2s	2	0,0s
14	509	83,2s	110	2,5s	9	0,2s
16	4.060	-	792	124,3s	49	3,9s
18	41.296	-	7.805	-	455	163,9s
20	510.489	-	44.903	-	5.783	-
22					54.136	-

Figure 5.1: Graphs checked with the algorithm.

with yellow. This is to emphasize that we have not checked all existing graphs with such properties. Finally, the values of time that are not written down is because we have partitioned the algorithm in two parts. First, we have saved in `possible_lonely_color_graphs.g6` the graphs that do not have a no lonely coloring using the perfect matchings. Then, we have checked directly the rest of the colorings on those remaining graphs.

So, once we have seen these results, is it possible that all cubic graphs with girth 4 or more have a no lonely coloring? Can we use other methods to find a prohibited graph with these characteristics?

5.2 Checking solutions by hand

As mentioned before, we have encountered two graphs that have been impossible to check with the previous algorithm (see Figures 5.2 and 5.3).

What we have done with these two graphs, is to search a solution by hand. Although it may seem a hard task, it has been surprising for us to see that, using two colors, we have been able to find a coloring very easily in these two cases. Nevertheless, the problem is that we have to confirm that it is in fact a no lonely coloring.

To do that, we have implemented an algorithm based on the Proposition 2.14. The code is similar to the fragment where has been used this same proposition in the previous algorithm. However, in this case, we have coded the algorithm with the objective of checking solutions using just two colors. It could be extended to check more possible colorings but, for our interests, we do not need more.

Nevertheless, the fact that there exists such a coloring, does not mean that these graphs are not prohibited. So, since it has taken so much time running without finding a solution, maybe this means that there are few no lonely colorings for

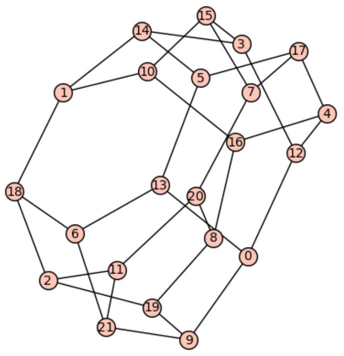


Figure 5.2: Cubic girth 5 graph of 22 vertices with g6 format: `U??????_A?C?e¿_I_?aOAI?L?BC?@@_?@c??d??`

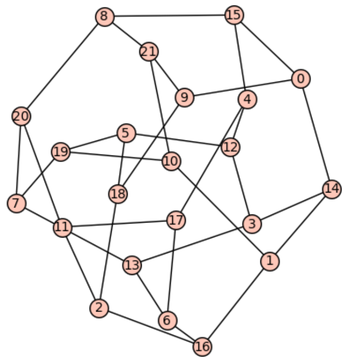


Figure 5.3: Cubic girth 5 graph of 22 vertices with g6 format: `U??????_A?C?F?CoY?CP?W_?IC@G_?IO?@c??M??`

these graphs. So, can we take advantage of that?

Furthermore, this problem has helped us arrive to another conclusion in respect of how easy has been to find a no lonely coloring with 2 colors. Maybe, if we improve the algorithm that was mentioned previously using 2 colors, we do not need to go through all the possible colorings.

5.3 Using two colors

In the section 5.1, we mentioned how we tried to color graphs using two colors to avoid problems with the memory. However, the only thing we did was to put a condition so that the labels would not go further than the number 2. The objective was satisfied and, although it took several hours, it actually found a coloring for the "evil graphs" mentioned in the previous section.

Nevertheless, after seeing how easy it was to find colorings with two colors, we thought that maybe we could try to improve that algorithm. In fact, it was not that hard, because using the functions used in the previous section, we could just check if the coloring is a solution at the moment where we change to the next color. Like this, we only color the graph with the label 1 and, when it is time to change to number 2, we just check the solution considering the uncolored edges as colored with the label 0.

This method has helped a lot to increase the efficiency and, if we apply a few changes to the functions that check the solution, by adapting them to this specific case, we could even improve it more. However, in this case the improvement will not be as noticeable. So, with the objective of not cause difficulties in terms of readability and understanding, we will just apply the changes mentioned in the previous paragraph.

In fact, with only that improvement, we have been able to check the two "evil graphs" in 0,36s. When before, it took us hours to check just one of the two graphs. Like this, this method not just has improved in terms of memory, but also in terms of time.

After observing the efficiency of this algorithm, we have been able to find a solution for all graphs of girth 5 and 22 vertices as a continuation of the results of the Figure 5.1.

In addition, since we have not succeeded in the search of a graph which is not a no lonely coloring, we have decided to examine the subcubic graphs. We have seen in the Proposition 2.16 that if we found a prohibited subcubic graph, this would imply the existence of a cubic prohibited graph. We then applied this program to all subcubic graphs with girth 4 or greater and up to 15 vertices. Additionally, we analyzed subcubic graphs with girth 5 or greater and 16 vertices. However, all the graphs tested admitted a solution with two colors. To obtain these subcubic graphs, we have used the function `geng` of *nauty* [20].

5.4 Checking minimal Cayley graphs

In this case, we will be trying to check directly if a graph is subgraph of a minimal Cayley graph. To do so, we will be using the list of minimal Cayley graphs on up to 255 vertices produced by Rhys J. Evans. These files are available under demand. The code is very simple and just uses a method of the `Graph` object named `subgraph_search`. Observe that this method will never say to us if a graph is prohibited, but it can help us to discard graphs.

In order to take advantage of this code, we have used it to check some generalized Petersen graphs. In particular, we have looked into the Dürer graph $P(6,2)$, which we have found that is subgraph of the cuboctahedron graph. This is the minimal Cayley graph of the alternating group A_4 with the presentation: $\langle a, b \mid a^3 = e, b^3 = e, (ab)^2 = e \rangle$. This Cayley graph is represented in the Figure 5.4.

We have obtained the same results with the other graphs mentioned in the section 4.1: $P(8,2)$ is subgraph of the minimal Cayley graph of $C3 \rtimes Q8$ and $P(10,2)$ of the minimal Cayley graph of the group $C3 \times C3 \times C3$.

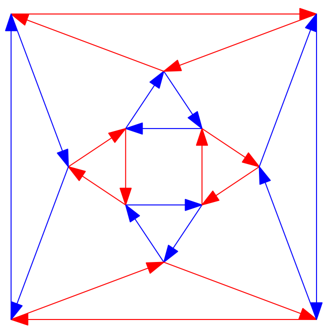


Figure 5.4: Cayley graph of $A_4 = \langle a, b \mid a^3 = e, b^3 = e, (ab)^2 = e \rangle$.

To conclude the subject of generalized Petersen graphs, we show the Petersen plane representing our results corresponding to this family of graphs. This representation has been inspired on the Petersen plane of [10].

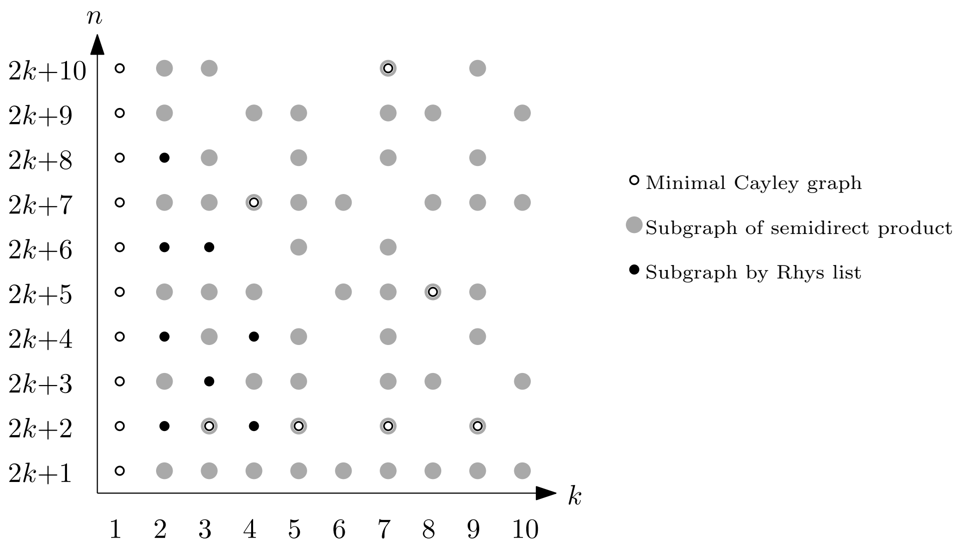


Figure 5.5: Representation of the $P(n, k)$ that are minimal Cayley graphs, subgraphs of the minimal Cayley graph of the semidirect product (see 4.16) or subgraphs of minimal Cayley graph saved in the list of Rhys J. Evans.

5.5 Going through all orientations

Now, since we are not having luck checking the colorings of the undirected graphs, a good option may be to consider its directed graphs. It is important to remember that, the no lonely color property is just a necessary condition, so it is possible that one of these graphs is prohibited despite the previous results.

Nevertheless, so that we can use the results obtained in the Section 2.4, it will be necessary to obtain all no lonely colorings of the graph. Then, we will have to modify some details of the algorithm so that it obtains all the solutions.

Our strategy will consist in, for each no lonely coloring of the undirected graph, we will check the degrees of each vertex in the directed graph as stated in the Definition 2.28. This will discard almost all orientations.

Remark 5.1. In *SageMath*, we can not define an edge that uses both directions at the same time. Instead, we have to use two parallel edges with opposite directions. To do so, we will activate the option of multiple edges whenever we define a directed graph.

Afterwards, once we have checked the degrees, we examine if there is any (monochromatic) cycle of length 2. If there is such cycle, we verify if there is a bigger monochromatic cycle. If this last cycle exists, we can discard the orientation by the Lemma 2.30.

Notice that we only do that if we find a cycle of length 2. This is because those are the only ones missing to check, since in the algorithm we were exploring if there were any monochromatic cycles of different lengths. However, we were considering undirected graphs, so only the cycles with length at least 3 were considered. That is why we have to repeat this process in case we find a cycle of length 2.

Finally, if both of these tests are passed, we arrive to the definitive one, which is based on the Proposition 2.29:

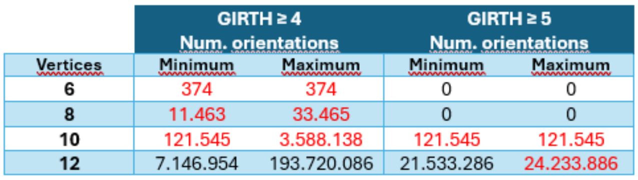
1. For each cycle (using edges in any direction and with no vertices repeated) we save the color sequence of the path. An edge in the opposite direction will be saved with the color in negative.

2. Then, for each vertex and for each cyclic rotation of the color sequence, we check if we can follow that same path.

3. If we have been able to follow that path and we have not obtained another cycle, by the Proposition 2.29, we can discard that orientation.

The problem with this algorithm is the number of possible orientations to examine: $3^{|E|}$, which is terrible in terms of complexity. But, what if we rule out the orientations that are isomorphic? We can do that using *nauty*. This program contains a function, named `directg`, that gives all orientations of a graph, in format `d6`, without considering the ones that are isomorphic to a previous one. This will decrease considerably the number of orientations to explore. However, this will not solve our problem, because the graphs that we need to check have a lot of edges and, therefore, also a lot of orientations to explore.

In total, with this method, we have checked eight girth 4 graphs and two of the girth 5 ones. In the following table, we can not see all the graphs, but it gives us an idea of the number of orientations that *nauty* produces. In the Figure 5.6, the graphs that appear in red are the ones that have been checked and that satisfy the necessary conditions to be a subgraph of a minimal Cayley graph. Also, we represent the minimum and maximum values of the orientations obtained from the different graphs in each file. In other words, since each file is organized by girth and the number of vertices, we have identified the graph in each file with the fewest orientations and the one with the most. This allows us to visualize the range of orientations present in each category:



	GIRTH ≥ 4 Num. orientations		GIRTH ≥ 5 Num. orientations	
Vertices	Minimum	Maximum	Minimum	Maximum
6	374	374	0	0
8	11.463	33.465	0	0
10	121.545	3.588.138	121.545	121.545
12	7.146.954	193.720.086	21.533.286	24.233.886

Figure 5.6: Minimum and maximum number of orientations by vertices and girth.

This high number of orientations causes an extremely high time of execution. This is because we have to go through all of these orientations for each no lonely coloring that we find. So that you have an idea of the magnitude of this problem, we will show the numbers of the only two girth 5 graphs that we have checked:

- Cubic girth 5 graph with 10 vertices: This graph is the Petersen graph $P(5,2)$ and has a total of **66** no lonely colorings and **121.545** orientations to check. For each solution, the program takes around **15s** to go through all orientations. However, it ends before checking all solutions because it finds an orientation that satisfies the stated conditions.

  So, for this case, it is not that much time, but we have to notice that this is the graph that has less orientations of the ones considered in the table with 10 vertices.

- Cubic girth 5 graph with 12 vertices: This graph is the first one of the two graphs of 12 vertices and girth 5. It has a total of **242** solutions and **24.233.886** orientations to inspect. For each solution, the program takes around **1h** to check all orientations. It was not until the solution number 188, after several days running, that we found an orientation. We can see the colored orientation found in Figure 5.7.

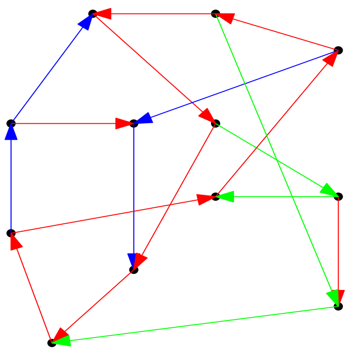


Figure 5.7: Oriented no lonely color solution of a cubic girth 5 graph with 12 vertices.

You may notice that we have not mentioned the other girth 5 graph with 12 vertices. This omission is due to our inability to verify all its orientations for each no lonely coloring. The verification of the graph shown in Figure 5.7 required significant computational effort from my personal computer, leading to internal errors. These interruptions have made the task of checking this other graph more challenging and time-consuming.

However, we have been able to go through the minimal Cayley graphs with this graph. With the program mentioned in the Section 5.4, we have seen that the graph in the Figure 5.8 is not subgraph of any minimal Cayley graph up to 127 vertices. Could this be the prohibited cubic graph of girth 5 that we have been searching for? One way to confirm this would be by applying the algorithm described in this section.

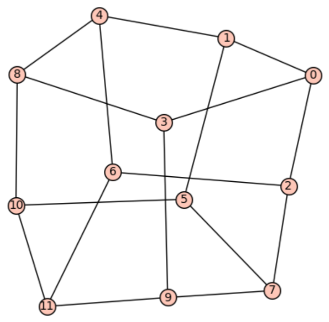


Figure 5.8: Cubic girth 5 graph of 12 vertices with g6 format: `KsP@PGWCOH?R`

In spite of these difficulties, one may be still thinking about those "evil graphs" of the section 5.2. Remember that, in that same section, we observed that if there were such big problems to find a solution considering all possible colorings, maybe it was because there were few of them. With this wondering in mind, we applied a few minor modifications (similar to the ones applied in the program explained in this section) to find all solutions with only two colors.

Nevertheless, in less than a minute, we were able to find more than a thousand solutions for each graph. We did not finish the algorithm to see the total number of solutions because it was not worth it. However, we now know that the option of going through all the orientations of these two graphs of 22 vertices, is inconceivable.

But, what about the subcubic graphs? We have been able to check that almost all subcubic graphs of at least girth 4 up to 9 vertices are subgraphs of minimal Cayley graphs. We have found only four exceptions:

- FCxv?
- GCR`v?
- H?ovE_w
- HCQf@o[

However, with each one of these graphs, we have found an orientation associated to a no lonely coloring that could be a subgraph of a minimal Cayley graph.

In addition, we have also verified that all subcubic girth 5 or greater graphs of 10 and 11 vertices are subgraphs of minimal Cayley graphs.

Conclusion

In this project, we have studied the properties of minimal Cayley graphs and their subgraphs. In particular, we have worked with cubic graphs, giving a lot of importance to the family of the generalized Petersen graphs. Additionally, we have proved Spencer's Theorem and observed that significantly decreasing the bound of maximum degree 100 is not straightforward. However, we have mentioned how Babai's question could be seen as a wondering of how much Spencer's Theorem can be improved.

The central objects were the graphs of girth 5 and Babai's question: *Is there a cubic girth 5 graph which is not a subgraph of a minimal Cayley graph?* Regarding this question, we have not reached a definitive conclusion as to why Babai specifically considered cubic graphs with girth 5. Perhaps, he talks of cubic graphs because is the smallest regular case which is not trivial and, as we have seen in the Proposition 2.16, because they generalize the case of the subcubic graphs. However, the girth 5 condition remains a mystery for us.

To search for such a prohibited graph, we have developed an algorithm that searches no lonely colorings in a given graph, along with another one that verifies the isomorphism of the walks in a no lonely colored digraph. Nevertheless, after going through over 95.000 graphs and millions of orientations, we have not been able to find a prohibited graph with those properties.

This work, however, raises many questions:

- Is it possible that all cubic graphs with girth 4 or greater are no lonely colorings? Observe that we have checked over 145.000 graphs without any luck.

- While our focus has been on girth 5, could there exist a cubic girth 4 prohibited graph? Notice that have only found prohibited graphs with girth 3, but not greater.

- What other methods can we use to search prohibited cubic graphs? Because, it is possible that all cubic graphs with girth 4 or greater have a no lonely coloring and, the complexity of the method that goes through all the orientations, is exponential. So, is there a smarter way or should we just use these algorithms in a much more potent computer?

- Finally, if we forget about regular graphs, the open question is: for which $d < 100$ is it true that for every girth g there is a forbidden graph of maximum degree at most d?

Additionally, even though we have not found a prohibited cubic girth 5 graph, we have proposed a candidate. We have seen that the graph of the Figure 5.8, is not subgraph of any minimal Cayley graph up to 127 vertices. So, if we are able to verify the orientations on all its solutions with the proposed algorithm, we may find a prohibited graph.

In conclusion, while we have not provided definitive answers, we have posed many questions that open the door to further research.

Appendix A

Code of the main algorithms

In our project folder of *Github*, we have 5 Jupyter notebooks that contain the main algorithms and 2 Python files that contain some support functions that are used in more than one notebook. We describe them briefly:

- `noLonelyColorDetector.ipynb`: Jupyter notebook that verifies if a graph is a no lonely color graph.

- `noLonelyColorings_allSolutionsAndOrientations.ipynb`: Jupyter notebook that obtains all no lonely colorings of a graph and that, for each solution, can go through all the orientations of the graph.

- `noLonelyColorDetector_2colorsVersion.ipynb`: Jupyter notebook that verifies if a graph is a no lonely color graph, but that only considers colorings that use two colors.

- `checkingMinCayleyGraphs.ipynb`: Jupyter notebook that, given a graph, goes through all minimal Cayley graphs up to 255 vertices until it finds one that contains the given graph as a subgraph.

- `twoColorSolutionChecker.ipynb`: Jupyter notebook that, given a graph colored using only two colors, checks if the coloring satisfies the no lonely color conditions. The functions that we use here, are saved in the Python file that shares name with this notebook. This method is based in the Proposition 2.14.

- `twoColorSolutionChecker.py`: Python file that contains the functions needed to verify if a coloring satisfies the no lonely color conditions.

- `supportFunctions.py`: Python file that contains the basic functions needed in the notebooks.

In this annex, we will show the most important algorithms that we have used in these programs.

A.1 The no lonely color detector

The code that we have implemented for the detection of no lonely colorings consists in a DFS algorithm for the first verification of the perfect matchings and a backtracking algorithm when we go through all the possible colorings.

With the following two functions, we are able to determine if there is a lonely color in a cycle for the case of the perfect matchings:

```
def isNoLonelyColor_ForPerfectMatchings(G):
    """
    Method that vertifies a colored graph with a perfect matching by calling a
         DFS method for each vertex of the graph.

    PARAMETERS:
        G: Graph which has the edges of the perfect matching colored with the
            label 1. The rest of the edges have label 0

    RETURNS:
        True if the matching is a no lonely coloring
        False if not
    """
    visited = set()
    for v in G.vertices():
        if v not in visited:
            visited.add(v)
            if not isNoLonelyColorDFS_ForPerfectMatchings(G, v, v, visited,
                set(), 0):
                return False

    return True

def isNoLonelyColorDFS_ForPerfectMatchings(G, currentNode, finalNode, visited
    = set(), path = set(), numEdgesMatching = 0):
    """
    DFS method that goes through all the cycles that contain the vertex
        finalNode.

    PARAMETERS:
        G: Graph which has the edges of the perfect matching colored with the
            label 1. The rest of the edges have label 0
        currentNode: Vertex that we are visiting at the moment
        finalNode: Vertex where the cycle ends (and begins)
        visited: set of vertices whose cycles have already been checked
        path: set that contains the vertices that uses the path
        numEdgesMatching: number of vertices of the matching that are in the
            path

    RETURNS:
        True if all the cycles that go through finalNode are okay
        False if not there is a cycle that contains a lonely color edge
    """
    if currentNode not in path:
        path.add(currentNode)

        for nei in G.neighbors(currentNode):
            if nei not in visited and nei not in path:  # Like this we won't
                review again a vertex which has all its cycles checked
                label = G.edge_label(currentNode, nei)
                aux_numEdgesMatching = numEdgesMatching + (1 if label == 1
                    else 0)

                if aux_numEdgesMatching > 1:
                    # It is not necessary to check this path because if we
                        find two edges of the perfect matching in the cycle,
                        it won't be a lonely color cycle
                    continue
                elif not isNoLonelyColorDFS_ForPerfectMatchings(G, nei,
                    finalNode, visited, path, aux_numEdgesMatching):
                    return False

            elif nei == finalNode and len(path) > 2:
                # We have found a cycle
                label = G.edge_label(currentNode, nei)
                aux_numEdgesMatching = numEdgesMatching + (1 if label == 1
                    else 0)

                if aux_numEdgesMatching == 1:
                    # We have found a cycle with a lonely color
                    return False

        path.remove(currentNode)

    return True
```

Then, if there is not a no lonely coloring using the perfect matchings as colorings, it is when we go to the following step. That is to try to search a no lonely coloring considering all the possibilities. To do so, we have the following function that implement a backtracking strategy:

```
def checkColorings_backtrack(G, G_aux, G_coloredEdges, edge, color,
    colorIsOkay, statesChecked, lenMonochromaticCycle):
    """
    Backtracking function that searchs a no lonely coloring of G.

    PARAMETERS:
        G: Graph that we want to color
        G_aux: Graph that contains the edges that are not colored with "color"
        G_coloredEdges: Graph that contains the colored edges
        edge: Edge that is going to be colored next
        color: Color that we will use to color "edge"
        colorIsOkay: Boolean that tells us that, once we color "edge", if the
            edges colored with "color" satisfy the needed conditions to be a
            no lonely color
        statesChecked: Set that contains the colorings that we have already
            discarded.
        lenMonochromaticCycle: If we find a monochromatic cycle, it contains
            the length. If not it is 0.

    RETURNS:
        True if it finds a no lonely coloring in G.
        False if not
    """
    u, v, label = edge[0], edge[1], edge[2]

    # If the edge is uncolored
    if label == 0:
        # We color the edge and update all the graphs
        G.set_edge_label(u, v, color)
        G_aux.delete_edge(u, v)
        G_coloredEdges.add_edge(u, v, color)

        # If this was the last edge to paint and the coloring is okay, we end
        if len(G_coloredEdges.edges()) == len(G.edges()) and colorIsOkay:
            return True

        # DEFINING edges_missing --> WE WANT TO KNOW WHICH EDGES SHOULD BE
            COLORED NEXT
        # We search for lonely uncolored edges in cycles
        if len(G_coloredEdges.edges()) != len(G.edges()): # If G is fully
            colored, there is no need
            checkNonColoredCycles, uncoloredEdges = sf.
                searchOfLonelyEdgesByColor(G, 0)
        else: checkNonColoredCycles = True

        if not checkNonColoredCycles:
            # uncoloredEdges contains a lonely uncolored edge in a cycle
            edges_missing = uncoloredEdges
        else:
            # Else we search for a lonely edge with color "color"
            checkCyclesColor, uncoloredEdges = sf.searchOfLonelyEdgesByColor(G
                , color)

            if not checkCyclesColor:
                # uncoloredEdges contains the edges of the cycle that contains
                     a lonely edge
                edges_missing = uncoloredEdges
            else:
                # If we have not found a lonely edge of color "color"...
                if colorIsOkay:
                    # We continue coloring with another color
                    if checkColorings_backtrack(G, G.copy(), G_coloredEdges,
                        sf.findNextEdge(G), color + 1, False, statesChecked,
                        0):
                        return True

                # If we have no candidates, we just try to color all the
                    uncolored edges
                edges_missing = G_aux.edges()
        # WE EXPLORE THE CANDIDATES TO COLOR
        for edge_to_color in edges_missing:
            u_to_color, v_to_color, label_to_color = edge_to_color[0],
                edge_to_color[1], edge_to_color[2]

            if label_to_color == 0:
                G_coloredEdges.add_edge(u_to_color, v_to_color, color)

                # If we have not discarded this state before
                if tuple(G_coloredEdges.edges()) not in statesChecked:
                    # If the vertices do not have three adjacent edges of
                        color "color"
                    if sf.isDegreeColorGood(G_coloredEdges, edge_to_color,
                        color):
                        lenMonochromaticCycle_currentEdge = sf.
                            searchMonochromaticCyclesInEdge(G_coloredEdges,
                            edge_to_color, color)

                        # If the monochromatic cycles have the same length
                        if (
                            lenMonochromaticCycle_currentEdge == 0
                            or lenMonochromaticCycle == 0
                            or lenMonochromaticCycle_currentEdge ==
                                lenMonochromaticCycle
                        ):
                            if lenMonochromaticCycle == 0:
                                # We have not found a previous monochromatic
                                    cycle. So, we save the new value
                                lenMonochromaticCycle =
                                    lenMonochromaticCycle_currentEdge

                            # We update G_aux
                            G_aux.delete_edge(u_to_color, v_to_color)

                            # If G_aux is disconnected and G_colored edges has
                                both endpoints
                            # of each edge of color "color" in different
                                components
                            if (
                                not G_aux.is_connected()
                                and sf.areEndpointsInDifferentComponents(
                                    G_coloredEdges, G_aux.connected_components
                                    (), color)
                            ):
                                # The edges of color "color" satisfy the
                                    necessary conditions
                                # --> colorIsOkay = True
                                if checkColorings_backtrack(G, G_aux,
                                    G_coloredEdges, edge_to_color, color, True
                                    , statesChecked, lenMonochromaticCycle):
                                    return True

                            # If not, we continue considering that the color
                                is not okay
                            # --> colorIsOkay = False
                            if checkColorings_backtrack(G, G_aux,
                                G_coloredEdges, edge_to_color, color, False,
                                statesChecked, lenMonochromaticCycle):
                                return True

                            # We restore G_aux
                            G_aux.add_edge(u_to_color, v_to_color, 0)

                # We restore G_coloredEdges
                G_coloredEdges.delete_edge(u_to_color, v_to_color)

        # NONE OF THE edges_missing CAN BE COLORED WITH color
        # To save memory, we do not save the cases that are the only option
            after a previous state
        if len(edges_missing) != 1: statesChecked.add(tuple(G_coloredEdges.
            edges()))

        # We restore the graphs
        G.set_edge_label(u, v, 0)
        G_aux.add_edge(u, v, 0)
        G_coloredEdges.delete_edge(u, v)

    return False
```

A.2 Going through all the orientations

In the Jupyter notebook `noLonelyColorings_allSolutionsAndOrientations.ipynb`, we have a first section where we can obtain all the no lonely colorings of each graph that is saved in the selected file. In this case, the code is very similar, since the backtracking algorithm only has a few minor changes.

Then, there is another section at the end of the notebook where, we can obtain all the no lonely coloring solutions for an specific graph. In addition, once we have obtained all the solutions, we will be asked if we want to check the orientations of the graph. For that, we need to save a file named `orientations.d6` in the same folder that contains the notebook. For the inspection of all orientations, the most important algorithm is the one that is based on the Proposition 2.29:

```
def checkCyclePaths(D, G):
    """
    Function that verifies if the same color sequence, starting at different
        vertices, gives an isomorphic path.

    PARAMETERS:
        D: Colored DiGraph
        G: Colored Graph (undirected graph of D)

    RETURNS:
        True if the paths are isomorphic.
        False if we find two paths with the same color sequence that are not
            isomorphic.
    """
    # We obtain all possible cycles in G (also the ones that use an arc in the
         opposited direction in D)
    for cycle in G.to_directed().all_simple_cycles():  # Cycle contains the
        vertex of the origin twice
        if len(cycle) > 3:  # We skip cycles of length 2
            labelsInCycle = set()
            path = []
            for i in range(len(cycle) - 1):
                if D.has_edge(cycle[i], cycle[i+1]):
                    label = D.edge_label(cycle[i], cycle[i+1])[0]
                else:
                    # The edge is in the opposite direction. So, we save the
                        label as negative
                    label = - D.edge_label(cycle[i+1], cycle[i])[0]

                path.append(label)
                labelsInCycle.add(label)

            # For every cyclic rotation of the path
            for i in range(len(path)):
                aux_path = path[i:] + path[:i]

                # For every vertex
                for v in D.vertices():
                    if not follow_path(D, v, aux_path, 0, v):
                        return False

    return True
```

A.3 No lonely color detection using two colors

In this Jupyter notebook, called `noLonelyColorDetector_2colorsVersion.ipynb`, is very similar to the code used in A.1. However, when it is the moment of changing of color in the backtracking function showed above, we just consider the uncolored edges as colored. Then, we use the functions of `twoColorSolutionChecker.py` and one of the methods of `supportFunctions.py`, to verify if the solution is correct. This speeds a lot the process.

A.4 The support functions

The Python file containing the support functions that we are using in almost every notebook is formed of simple methods that help to simplify the code. In special, we want to show the one that is probably the most important function in the backtracking algorithm. This method helps reduce the next edges to color in the algorithm and makes the backtracking much more efficient. We do that with a DFS function that searches lonely color edges in the cycles of the graph:

```
def searchOfLonelyEdgesByColor(G, color):
    """
    Method that vertifies a colored graph by calling a DFS method for each
        vertex of the graph. It also returns some candidates to be colored
        next.

    PARAMETERS:
        G: Graph which has some edges colored

    RETURNS:
        True, None if all the cycles are correctly colored.
        False, uncoloredEdges if we have found a cycle that has a lonely edge
            of color "color" and gives the uncolored edges that should be
            colored next.
    """
    visited = set()
    for v in G.vertices():
        if v not in visited:
            visited.add(v)
            check, uncoloredEdges = searchOfLonelyEdgesByColorDFS(G, v, v,
                color, visited, set(), 0, set())
            if not check:
                return False, uncoloredEdges

    return True, None


def searchOfLonelyEdgesByColorDFS(G, currentNode, finalNode, color, visited,
    path, numEdgesColor, uncoloredEdges):
    """
    DFS method that goes through all the cycles that contain the vertex
        finalNode in the search of a lonely colored edge of color "color".

    PARAMETERS:
        G:              Partially colored Graph
        currentNode:    Vertex that we are visiting at the moment
        finalNode:      Vertex where the cycle ends (and begins)
        color:          Color of which we are searching a lonely color edge in
             a cycle
        visited:        set of vertices whose cycles have already been checked
        path:           set that contains the vertices that uses the path
        numColor:       number of vertices of color "color" that are in the
            path
        uncoloredEdges: set of edges that are uncolored in the current cycle

    RETURNS:
        True, uncoloredEdges if all the cycles are correctly colored.
        False, uncoloredEdges if we have found a cycle that has a lonely edge
            of color "color" and gives the uncolored edges of the cycle.
    """
    if currentNode not in path:
        path.add(currentNode)
        for nei in G.neighbors(currentNode):
            label = G.edge_label(currentNode, nei)
            aux_numEdgesColor = numEdgesColor

            if nei not in visited and nei not in path:  # To not use vertices
                that are already fully checked
                if label == color: aux_numEdgesColor += 1

                if aux_numEdgesColor > 1:
                    # If we find two edges of color "color" in a cycle, it won
                        't be a lonely color cycle
                    continue

                else:
                    if label == 0:
                        uncoloredEdges.add((currentNode, nei, label))

                    check, uncoloredEdges = searchOfLonelyEdgesByColorDFS(G,
                        nei, finalNode, color, visited, path,
                        aux_numEdgesColor, uncoloredEdges)
                    if not check:
                        return False, uncoloredEdges

            elif nei == finalNode and len(path) > 2:
                if label == color: aux_numEdgesColor += 1

                if aux_numEdgesColor == 1:
                    if label == 0:
                        uncoloredEdges.add((currentNode, nei, label))

                    # We have found a cycle with a lonely color
                    return False, uncoloredEdges

            # We delete the edge because if we have arrived here means that
                the cycle was correct
            uncoloredEdges.discard((currentNode, nei, label))

        path.remove(currentNode)

    return True, uncoloredEdges
```

Bibliography

[1] László Babai. "Automorphism groups, isomorphism, reconstruction (Chapter 27 of the Handbook of Combinatorics)". In: *North-Holland–Elsevier* (1995), pp. 1447–1540.

[2] László Babai. "Chromatic number and subgraphs of Cayley graphs". In: *Theory and Applications of Graphs: Proceedings, Michigan May 11–15, 1976*. Springer. 1978, pp. 10–22.

[3] László Babai. *Embedding graphs in Cayley graphs*. English. Problèmes combinatoires et théorie des graphes, Orsay 1976, Colloq. int. CNRS No. 260, 13-15 (1978).

[4] László Babai and Vera T. Sós. "Sidon sets in groups and induced subgraphs of Cayley graphs". English. In: *Eur. J. Comb.* 6 (1985), pp. 101–114. ISSN: 0195-6698. DOI: `10.1016/S0195-6698(85)80001-9`.

[5] H.S.M. Coxeter. "The generalized Petersen graph G(24, 5)". In: *Computers Mathematics with Applications* 12.3, Part 2 (1986), pp. 579–583. ISSN: 0898-1221. DOI: `https://doi.org/10.1016/0898-1221(86)90412-8`. URL: `https://www.sciencedirect.com/science/article/pii/0898122186904128`.

[6] Shagnik Das. "A brief note on estimates of binomial coefficients". In: *URL: http://page. mi. fu-berlin. de/shagnik/notes/binomials. pdf* (2016).

[7] David Eppstein. *The many faces of the Nauru graph*. 2007. URL: `https://11011110.github.io/blog/2007/12/12/many-faces-of.html`.

[8] Roberto Frucht, Jack E. Graver, and Mark E. Watkins. "The groups of the generalized Petersen graphs". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 70.2 (1971), 211â218. DOI: `10.1017/S0305004100049811`.

[9] Ashwin Ganesan. *Cayley graphs and symmetric interconnection networks*. 2017. arXiv: `1703.08109 [math.CO]`. URL: `https://arxiv.org/abs/1703.08109`.

[10] Ignacio García-Marco and Kolja Knauer. "Beyond symmetry in generalized Petersen graphs". In: *Journal of Algebraic Combinatorics* 59.2 (2024), pp. 331–357.

[11] Ignacio García-Marco and Kolja Knauer. “Coloring minimal Cayley graphs”. In: *European Journal of Combinatorics* 125 (2025), p. 104108. ISSN: 0195-6698. DOI: https://doi.org/10.1016/j.ejc.2024.104108. URL: https://www.sciencedirect.com/science/article/pii/S0195669824001938.

[12] Ignacio García-Marco and Kolja Knauer. “On sensitivity in bipartite Cayley graphs”. In: *Journal of Combinatorial Theory, Series B* 154 (2022), pp. 211–238. ISSN: 0095-8956. DOI: https://doi.org/10.1016/j.jctb.2022.01.002. URL: https://www.sciencedirect.com/science/article/pii/S0095895622000028.

[13] C. Godsil and G.F. Royle. *Algebraic Graph Theory*. Graduate Texts in Mathematics. Springer, 2001. ISBN: 9780387952208. URL: https://books.google.es/books?id=pYfJe-ZVUyAC.

[14] Chris D. Godsil and Wilfried Imrich. “Embedding graphs in Cayley graphs”. English. In: *Graphs Comb.* 3 (1987), pp. 39–43. ISSN: 0911-0119. DOI: 10.1007/BF01788527.

[15] A.T. Grau. *Aritmètica*. UB (Universidad de Barcelona). Universitat de Barcelona, 1998. ISBN: 9788483380314. URL: https://books.google.es/books?id=QQ4wjrO_hiQC.

[16] S. D’hondt K. Coolsaet and J. Goedgebeur. *The House Of Graphs: Connected cubic graphs*. 2024. URL: https://houseofgraphs.org/meta-directory/cubic.

[17] S. D’hondt K. Coolsaet and J. Goedgebeur. *The House Of Graphs: Minimal Cayley graphs*. 2024. URL: https://houseofgraphs.org/meta-directory/minimal-cayley.

[18] Ulrich Knauer and Kolja Knauer. *Algebraic Graph Theory: Morphisms, Monoids and Matrices*. 2019. ISBN: 9783110617368.

[19] Saunders Mac Lane and Garrett Birkhoff. *Algebra*. Vol. 330. American Mathematical Society, 2023, pp. 414–415.

[20] Brendan D. McKay and Adolfo Piperno. “Practical graph isomorphism, II”. In: *Journal of Symbolic Computation* 60 (2014), pp. 94–112. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2013.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0747717113001193.

[21] D. Robinson. *A Course in the Theory of Groups*. Graduate Texts in Mathematics. Springer New York, 1996. ISBN: 9780387944616. URL: https://books.google.es/books?id=lqyCjUFY6WAC.

[22] Marko Lovrecic Sarazin. “A Note on the Generalized Petersen Graphs That Are Also Cayley Graphs”. In: *J. Comb. Theory B* 69 (1997), pp. 226–229. URL: https://api.semanticscholar.org/CorpusID:206570808.

[23] Joel Spencer. “What’s not inside a Cayley graph”. In: *Combinatorica* 3 (1983), pp. 239–241.

[24] Mark E. Watkins. *Vertex-Transitive Graphs That Are Not Cayley Graphs*. Ed. by Geňa Hahn, Gert Sabidussi, and Robert E. Woodrow. Dordrecht: Springer Netherlands, 1990, pp. 243–256. ISBN: 978-94-009-0517-7. DOI: 10.1007/978-94-009-0517-7_19. URL: https://doi.org/10.1007/978-94-009-0517-7_19.