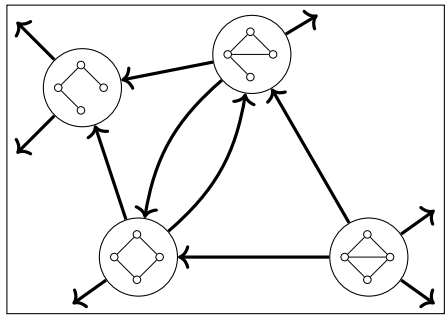
UMONS

Université de Mons
Faculté des Sciences
Département d'Informatique

Faculté
des Sciences

Proofs by Transformation in Extremal Graph Theory

Gauvain Devillez



Dissertation submitted in fulfilment of the requirements for the degree of
Docteur en Sciences

Jury

Pr. Dr. Véronique Bruyère, President	Université de Mons
Pr. Dr. Hadrien Mélot, Director	Université de Mons
Pr. Dr. Alain Hertz, Co-Director	Polytechnique Montréal
Pr. Dr. Jef Wijsen	Université de Mons
Pr. Dr. Bernard Ries	Université de Fribourg
Pr. Dr. Anthony Labarre	Université Gustave Eiffel
Pr. Dr. Jan Goedgebeur	KU Leuven

Thanks

Avec cette thèse, c'est une aventure de plusieurs années qui se termine. Bien sûr, cette aventure n'a pas été de tout repos, sinon ce ne serait pas une belle aventure. Pourtant, je n'ai pas vu ces années passer. Pourquoi ? Parce que comme le dit Izaak Walton : "Good company in a journey makes the way to seem the shorter."

Parmi toutes ces personnes qui m'ont accompagné dans cette aventure, je veux d'abord remercier mes deux directeurs : Alain et Hadrien. Hadrien, avant même le début de mes études, tu as pris de ton temps pour encadrer le stagiaire timide que j'étais. Et, depuis ce jour, tu as toujours été présent, que ce soit pour m'aider quand j'étais bloqué, me remotiver quand je désespérais ou simplement pour boire un verre ensemble. Merci pour tout cela.

Alain, on s'est vus moins souvent, mais je ne te suis pas moins reconnaissant. Quand j'étais à Montréal, loin de chez moi, en plein hiver et toujours très timide, pas une fois, je n'ai eu le mal du pays. Ton optimisme permanent, tes idées intarissables et bien sûr, ta bienveillance, m'ont permis de me sentir à ma place dans ce monde encore inconnu de la recherche.

Hadrien, Alain, vous avez tous les deux été pour moi plus que des directeurs de thèse. Vous avez été et êtes toujours des mentors et des amis. Je n'aurais pas pu rêver mieux.

Mais, il y a un autre avec qui j'ai beaucoup travaillé et que je n'oublie pas. Merci Pierre pour toutes ces conversations extrêmement intéressantes, tous ces verres bus ensemble et bien sûr, pour m'avoir supporté après un verre de trop. Mais, surtout, merci pour tous ces conseils et pour avoir toujours été

celui qui gérait les imprévus quand je n'avais pas pu m'y préparer une semaine à l'avance.

Je veux aussi remercier Anthony Labarre, Bernard Ries, Jan Goedgebeur, Jef Wijsen et Véronique Bruyère pour avoir accepté d'être dans mon jury, pour avoir pris le temps de lire ma thèse et pour toutes vos questions et remarques pertinentes et intéressantes.

Je ne peux malheureusement pas expliquer tout ce qui s'est passé pendant les années qui ont mené à la réalisation de cette thèse. Mais, je peux au moins remercier de tout cur tous ceux qui ont été présents durant ces années. Merci à tout le département de Sciences Informatiques qui n'est composé que de personnes exceptionnelles. Grâce à vous, j'ai pu me sentir à ma place et, pour moi, ça compte énormément. Et merci au département de Sciences Mathématiques. Même si je n'en fais pas partie, vous êtes tout aussi exceptionnels. Merci à Jérémy, qui n'a jamais hésité à me donner un coup de main quand je croulais sous les corrections. Merci à Horacio, Marion et Maximilien qui ont amené tellement de vie et de bonne humeur au travail. Merci Sébastien, je sais que ma relève est assurée. Merci aux élèves-assistants sans qui les TPs seraient tombés dans le chaos. Et merci aux secrétaires qui ont dû se battre avec les horaires, mais qui n'en gardent pas moins le sourire. C'est un problème mathématiquement compliqué, après-tout.

Et comment pourrais-je oublier tous ceux qui m'ont supporté en dehors de l'université ? Merci Julien pour avoir été un véritable ami pendant toutes ces années. J'aurais été bien seul sans notre amitié. Et merci aussi pour ton aide à optimiser mon code C++. Merci aussi à Florent et Sandro pour tous ces bons moments passés ensemble. Vous aussi, vous êtes des véritables amis.

Enfin, merci à ma famille qui m'ont toujours supporté, encouragé, félicité et ont essayé de comprendre ce que je faisais. Sans vous, je ne serais (littéralement) pas là. Papa, maman, vous avez fait tellement pour moi que je n'ai pas assez de place pour vous remercier assez. Et Géraud, merci d'avoir supporté ton grand-frère depuis si longtemps. Tu as définitivement la patience de réaliser tes rêves.

Et, comme je suis parfois distrait, merci à tous ceux que j'ai oubliés !

Contents

Chapter 1

Introduction

Representing real-world applications using mathematical models has allowed scientists many discoveries. One model appears almost everywhere: the graph.

Graphs are mathematical models representing relationships between elements. Graphs can be used to represent many different concepts such as molecules, train networks, social networks and even constraints when making schedules. They all have in common a set of elements and links between pairs of those elements. The atoms in a molecule are linked together, train stations are linked by train lines, users of social networks are linked by friendship, etc. Modeling a schedule using a graph is less obvious. In this context, two events to be scheduled are linked by conflicts such as involving the same people, which means that they cannot be scheduled at the same time.

In a graph, those elements are called the vertices of the graph and the links between them are called the edges. For example, the graph of Figure 1.1 has 10 vertices and 15 pairs of vertices that are linked by an edge.

The graph model has been used to study and solve many problems, from computing the best way to assign tasks to computers to finding the best structure for a metro network. In this last example, the best structure is usually the structure that will have the lower cost or the higher performance. In most cases, this structure must respect constraints, such as not having isolated sta-

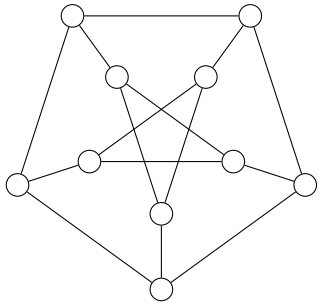


Figure 1.1: The Petersen graph.

tions that cannot be reached or not having two railways crossing. This is a special case of a more general problem of finding a graph that has the best structure according to some objective function for some parameters. While there are many algorithms that can be used to find a good graph in such a problem, finding the best one is often difficult and takes a lot of time, even for a computer.

For this reason, scientists try to identify the best possible graphs with some parameters such as the number of vertices and prove mathematically that these graphs are indeed the optimal ones. This is called *extremal graph theory*. The difficulty of this approach is that there is an infinite number of graphs, and even only considering a subset of those graphs often leaves us with a large number of different graphs. Fortunately, we now have powerful computers able to perform billions of computations in a split second. These computers have been used to help researchers identify candidate graphs that might be the best ones in an extremal graph theory problem. Different computer systems using different approaches exist to do this, and they have all been of a great help when searching for those best graphs.

But most of the time, those tools only provide candidate graphs with no certainty that they are indeed the best possible ones, and researchers need to prove this by hand. A common method, called a proof by transformation, consists in showing that any graph that is not one of those candidates can be transformed into a better graph. Finding those transformations, though,

is often difficult because of the number of graphs and, to the best of our knowledge, there is no generic software aimed at assisting researchers in finding those proofs like there is for finding graph candidates.

This is why, in this document, we present TransProof, a tool able to compute transformations on a large number of graphs in order to assist researchers in testing out the transformations they plan to use in a proof before setting out to write the proof. Such a tool can be very helpful in identifying counter-examples which can show that a proof idea requires some refinement or that it is entirely wrong and that time should not be wasted on it. TransProof was used in scientific research and helped in the writing of proofs for several results, but also produced interesting conjectures that are yet to be proven.

In this thesis, we will first introduce notations and definitions that are important for a good understanding of the subject. Next, Chapter 3 will go into more details about extremal graph theory and how computers help with research in this field. Chapter 4 then presents the common method of the proof by transformation and introduces the metagraph representation that is used by TransProof. TransProof itself is presented in Chapter 5. This chapter explains the ideas used in the writing of this tool in order to achieve good performance despite many problems being known as computationally expensive. Chapters 6 and 7 present results obtained in extremal graph theory when studying the eccentric connectivity index of a graph and the average number of colors in the non-equivalent colorings of a graph respectively. TransProof was used in the study of those two values, showing problematic cases such as counter-examples to the ideas for proofs and helping correct them. Finally, we summarize the results we obtained and discuss them as well as future works and open problems in Chapter 8.

Chapter 2

Definitions and notation

Before we start explaining extremal graph theory and TransProof, there are important notions that need to be defined in order to understand the subject. We start in the first section by giving the different types of graphs and introducing the specific notations and terms. To make finding the definitions of the different concepts in the text, the first time we use a new concept, we also add it in the margin.

2.1 Graphs

A *graph* is an abstract structure used to model relationships between elements. Usually written as $G = (V, E)$, a graph G is defined as a pair of two sets V and E. The set V is called the *vertex set* and contains the vertices of the graph, that is, the elements whose relationships we want to model. This set is sometimes denoted by $V(G)$ for a given graph G and the number of elements of this set is the *order* of the graph (denoted by $|G|$ and often represented by the letter n). The *edge set* E is a set containing pairs of elements of V ($E \subseteq V \times V$). Those pairs are called the edges of the graph. If $(u, v) \in E$, we say that there is an edge between u and v or that u and v are *adjacent*. Also, u and v are called the *endpoints* or *extremities* of the edge. Two edges sharing a common extremity are said to be *incident* to each other. The set

graph

vertex set

order

edge set

adjacent

vertices

extremities

incident

size

E is often denoted by $E(G)$ and its number of elements is the *size* of a graph (denoted by $||G||$ and often represented by the letter m).

In this document, we will only consider simple undirected graphs unless stated otherwise. This means that all the pairs $\{u, v\} \in E$ must be such that u and v are distinct (no edge between a vertex and itself) and that this pair is present at most once in E (only one edge between two vertices). A graph with those properties is a *simple graph*. An *undirected graph* adds the constraint that the edges (u, v) and (v, u) are considered to be the same, that is, the order of the vertices in the pair does not matter. If we differentiate between (u, v) and (v, u), we have a *directed graph* and we use the word *arc* instead of edge and the edge set becomes the arc set. Given a vertex v, all the arcs of the form (v, x) are called the *outgoing arcs* of v since they are leaving v to reach some vertex x and those of the form (x, v) are the *incoming arcs* of v.

simple graph

undirected graph

directed graph

arc

outgoing arc

incoming arc

Figure 2.1 shows a simple undirected graph G and a directed graph H. Both have the same vertex set $\{a, b, c, d\}$ but their edge sets are different. For the graph G, $E(G) = \{(a, b), (a, c), (b, c), (c, d)\}$. It is a simple graph, as every vertex pair appears at most once in its vertex set and each pair contains two distinct vertices. The graph G has order $|G| = 4$ and size $||G|| = 4$. The graph H is a directed graph, as shown by the arrows. Its arc set is $E(H) = \{(a, b), (a, b)', (b, a), (b, d), (c, d), (c, c)\}$. It is not a simple graph as we can see that there are two different arcs from a to b and an arc from c to itself.

Figure 2.1: A simple undirected graph G and a directed graph H

2.2 Structural properties of graphs

In this document, we only study these graphs for their structure. Thus, the notion modeled by the graphs such as train stations or schedules are not important as they do not change the underlying problems. The structure of a graph can be interesting in many ways, and there are many recognizable structures. A widely known structure in graph theory is the path. A *path* on n vertices (denoted by P_n) is simply a set of vertices $v_1, \ldots, v_n$ such that given two vertices v_i and v_j, there is an edge between the two if and only if $j = i + 1$. Note that the *length of a path* is its number of edges and not its number of vertices. If there is also an edge between v_1 and v_n, the structure is a *cycle* on n vertices (denoted by C_n). Figure 2.2 shows an example of a path and a cycle.

path

length of a path

cycle

(a) A path on 4 vertices (P_4). (b) A cycle on 4 vertices (C_4).

Figure 2.2: Examples of a path and a cycle.

If a graph on n vertices is such that there is an edge between every pair of distinct vertices, this graph is called a *complete graph* (denoted by K_n). Conversely, if the edge set is empty, the graph is an *empty graph*. It is the complement of the complete graph.

complete graph

empty graph

Definition 1. Let G and H be two graphs with the same vertex set V, we say that H is the *complement* of G if for every pair (u, v) of vertices, $(u, v) \in E(H)$ if and only if $(u, v) \notin E(G)$. This is also written $H = \overline{G}$.

complement

Because the empty graph is the complement of the complete graph, it will be denoted by $\overline{\mathsf{K}}_n$. An example of both a complete graph and an empty graph can be seen in Figure 2.3.

(a) A complete graph on 4 vertices (K_4). (b) An empty graph on 4 vertices ($\overline{\mathsf{K}}_4$).

Figure 2.3: Examples of a complete and an empty graph.

Those configurations span the whole graphs, but some substructures can be of interest too. For example, a graph could have a subset of its vertices that form a complete graph. Such a subset is called a *clique*. When a subset of vertices forms an empty graph instead, we say that this is a *stable set* (sometimes called an *independent set*). Those are both examples of subgraphs.

clique

stable set

independent set

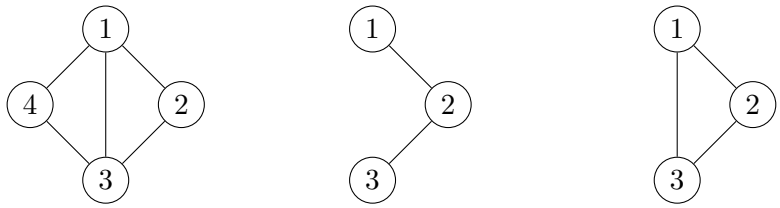
subgraph

supergraph

induced subgraph

Definition 2. A graph G' is a *subgraph* of a graph G if and only if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The graph G is then a *supergraph* of G'. If $E(G') = \{(u, v) \in E(G) \mid u, v \in V(G')\}$, G' is called an *induced subgraph*.

In Figure 2.4, G' and G'' are subgraphs of G but G' is not an induced subgraph as vertices 1 and 3 are adjacent in G but not in G'. The graph G'' is an induced subgraph of G.



G G' G''

Figure 2.4: A graph (left) with one of its subgraphs (middle) and an induced subgraph (right).

Since we only study the structures of the graphs, the way they are drawn or the names of their vertices does not matter. If two graphs have the same

structure but differ only by the name, order or position of their vertices, we will say that these graphs are isomorphic and consider them as the same graph.

Definition 3. Let G and H be two graphs. We say that G and H are *isomorphic*, denoted by $G \simeq H$, if there exists a bijection $f : V(G) \to V(H)$ such that $\forall u, v \in V(G), (u, v) \in E(H) \Leftrightarrow (f(u), f(v)) \in E(H)$. The bijection f is called a *graph isomorphism* between G and H.

isomorphic

graph isomorphism

Given two graphs G and H, which are isomorphic to each other, and the isomorphism f, if a is a vertex of G and b is a vertex of H such that $f(a) = b$, we will say that a is *mapped* to b. We also have that b is mapped to a.

mapped

Figure 2.5 shows two graphs which are isomorphic. Indeed, if we map vertices 1 through 5 in G to the vertices e, b, d, a and c of H in this order, we obtain the same graph.

Figure 2.5: Two isomorphic graphs.

When working with computers and using graphs with unnamed vertices, it is common practice to use the ordered set $(1, \ldots, n)$ as the vertex set of a graph of order n. Thus, each vertex is referred to by an index corresponding to its position in the set. The first vertex will have index 1, the second will have index 2 and so on. An edge between two vertices is represented as the pair of the indices of the two vertices. For example, if there is an edge between the first and the second vertex of the graph, we store this information as the pair $(1, 2)$. In this case, two isomorphic graphs G and G' will have the same vertex set but maybe in a different order.

If two isomorphic graphs have the same vertex set, the isomorphism is called an *automorphism*. In this case, the bijection that is the automorphism

automorphism

is simply a permutation of the vertices. Among all the orderings of the vertex set that can be obtained from an edge-preserving permutation, we can choose one ordering to represent all the other ones. This ordering is called a *canonical ordering* of the graph. Because this ordering can be seen as giving each vertex an index from 1 to n, it is also referred to as a *canonical labeling*.

canonical ordering
canonical labeling

This canonical ordering is chosen in such a way that it can be obtained from any ordering of the vertex set. There already are several tools able to compute such an ordering given any graph, such as Nauty and Traces [74] or bliss [65].

2.3 Graph invariants

With two isomorphic graphs, their canonical labelings are the same but it is not the only property they have in common. Given that two isomorphic graphs have the same structure, many functions on those graphs will produce the same results. Those functions are called graph invariants, as the image of a graph G under those functions is invariant by isomorphism of G.

graph invariant

Definition 4. A *graph invariant* is a function f on graphs such that for any two graphs G and H, $G \simeq H \Rightarrow f(G) = f(H)$.

The number of graph invariants is infinite since mathematical operations between invariants are also invariants. They can derive from properties of the vertices, such as the *degree* of a vertex v in a graph G (denoted by $deg_G(v)$) which is simply the number of edges incident to v. The degree of a vertex is not an invariant, but one could obtain an invariant by aggregating information about degrees, such as the average degree of the vertices or simply by taking the sorted list of the degrees of each vertex. The minimum and maximum degree in a graph are also common graph invariants. This degree is also used to describe different types of vertices, such as *pendant vertices* whose degree is 1 or vertices with degree 0 which are called *isolated vertices*. If a vertex has degree $n-1$ where n is the order of the graph, it is called a *dominant vertex* because it is adjacent to every other vertex of the graph. Another invariant

degree

pendant vertex
isolated vertex
dominant vertex

using the degrees is whether a graph is *regular*, that is, if all of its vertices have the same degree.

regular graph

There are also invariants based on paths between the vertices of a graph. If, for every pair of vertices u and v of a graph G, there is an induced path with those two vertices as its extremities, G is said to be a *connected graph*. If there exists at least two vertices such that there is no path between them, G is *disconnected*. Being connected or not is also a graph invariant. Often, in a disconnected graph, we consider the *connected components* of this graph. They are the maximal subsets of vertices such that the induced subgraph they define is connected. The number of connected components of a graph is also a graph invariant. If there is at least one induced path between two vertices u and v, the length of the shortest path between these two vertices is the distance between u and v.

connected graph

connected components

Using the distances, one can compute the largest distance from a specific vertex u to any other vertex and obtain the *eccentricity* of u. The largest of the eccentricities is called the *diameter* of a graph and is a graph invariant, and in a graph with a diameter λ, an induced path of length λ is a *diametral path*.

eccentricity

diameter

diametral path

Another type of graph invariants is based on coloring the graph. A *coloring* of a graph is simply an assignment of colors to its vertices such that adjacent vertices have different colors. This is equivalent to partitioning the vertices with the constraint that two adjacent vertices must be in two different subsets. When coloring a graph, one will often wish to use as few colors as possible. The minimum number of colors required to color a graph is called the *chromatic number* and is denoted by $\chi(G)$.

coloring

chromatic number

As we saw, invariants can be as simple as the order and size of the graph or more complex, such as the chromatic number. Even though numerical invariants are the most commonly used, there are many other types of invariants such as Boolean values, vectors, matrices or even text. We can cite the sorted list of degrees, whether the graph is connected and even the canonical labeling of a graph as examples of invariants. Note that a graph invariant is not nec-

essarily defined for all graphs. For example, the diameter in a graph cannot be computed for disconnected graphs as there is no path between different connected components[1].

2.4 Graph transformations

Given an invariant and a graph, we might also be interested in the effect on the invariant that changes in the graph could cause. Those changes in the graph are called graph transformations[2].

Transforming a graph is done by selecting edges and vertices on this graph and changing their structure, that is, removing selected edges or vertices or adding edges between two non-adjacent selected vertices or even adding new vertices.

A transformation thus needs two sets of both vertices and edges. We need to know the set V_r of vertices that will be removed, the set E_r of edges that will be removed, the set V_a of vertices to add and the set E_a of edges to add. Of course, there are constraints on these sets:

- $V_a \cap V = \emptyset$ (we cannot add an already existing vertex),
- $E_a \cap E = \emptyset$ (we cannot add an already existing edge),
- $V_r \subseteq V$ (we cannot remove a vertex that is not in the graph),
- $E_r \subseteq E$ (we cannot remove an edge that is not in the graph),
- $\forall v \in V_r, w \in V, (v, w) \in E \Rightarrow (v, w) \in E_r$ (removing a vertex implies removing all its incident edges).

[1] Of course, it is possible to decide that the diameter of a disconnected graph is infinite, but this is not interesting if one wants to compute real world distances.

[2] In the literature, the concept of graph transformations is slightly different, as there is a set of labels assigned to the vertices [82]. Here, we only consider the underlying structure of the graph.

If we have these constraints, we can define

$$V_C = \left\{ v \in V \mid v \in V_r \text{ or } \exists w \text{ such that } (v, w) \in (E_r \bigcup E_a) \right\}$$

and

$$E_C = \{(x, y) \in E \mid x \in V_C \text{ and } y \in V_C\},$$

and create a graph $B = (V_C, E_C)$ containing the set of vertices and edges of G that will be modified by the transformation.

The graph $R = ((V_C \bigcup V_a) \backslash V_r, (E_C \bigcup E_a) \backslash E_r)$ is not necessarily a subgraph of G since it contains vertices and edges that are not part of G (the elements of V_a and E_a) but it is a subgraph of the graph G' obtained after applying the transformation on G. Actually, R contains only the vertices and edges that differ between G and G'.

In Figure 2.6, the graph G is to be transformed by removing the vertex 4 and the edge $(3, 1)$ (in bold) and adding a vertex 5 as well as the edge $(1, 5)$. The set V_C is then the set $\{1, 3, 4\}$ and the graph B as defined above is represented in the middle. The graph R is seen on the right and contains only the elements affected by the transformation and that are not removed. The transformation can be applied by replacing the subgraph B in G by the graph R.

The vertex set of B is a subset of $V(G)$ but we can soften this requirement by saying that we need to have a mapping between the two sets. Of course, the vertices that are present in R but not in B must not be mapped to vertices of G since they are to be added to G by the transformation. This mapping would be a subgraph isomorphism, that is, an isomorphism which maps all the vertices of the subgraph to vertices of the supergraph but does not necessarily map all the vertices of the supergraph (if the supergraph has more vertices than the subgraph).

graph transformation

A *graph transformation* can thus be seen as a pair of graphs B and R with B being a template of a configuration in the graph that will be replaced by a different configuration R. To apply a transformation $T = (B, R)$ to a graph G, we need to find a subgraph H of G (not necessarily induced) that

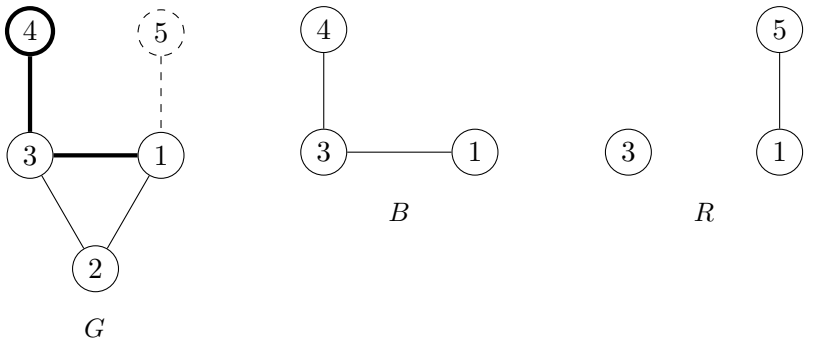


Figure 2.6: A transformation of the graph G represented as a pair of graphs B and R.

is isomorphic to B. Then, we replace this subgraph by R. As an example, the transformation consisting of removing an edge would have an edge (a, b) as its graph B and two isolated vertices a and b as its graph R. Applying this transformation would then consist of finding an edge and removing it.

basis
result
instance
application

Given a graph transformation T, we will say that the graph B is the *basis* of T and R is the *result* of T. An isomorphism between B and a subgraph of a graph G is called an *instance* of T and the graph obtained by applying T on an instance is an *application* of T.

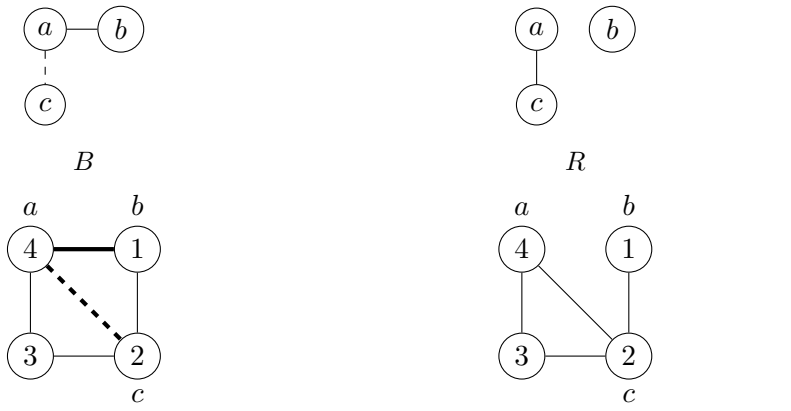
Note that, for the inverse transformation, adding an edge, we need two vertices that are not already adjacent. The definition of subgraph does not allow making a difference between a pair of vertices that can be disjoint and a pair that must be disjoint. Thus, in our graph B, we will label some non-adjacent vertices as non-edges. A *non-edge* is a pair of vertices that are not adjacent in a graph. In a transformation, a non-edge is a pair of vertices that cannot be adjacent in the instance of a transformation.

non-edge

rotation

In Figure 2.7, we apply a transformation called a *rotation*. This transformation consists of removing an edge (a, b) and then adding an edge (a, c) that was not already present in the graph. Given those two edges (a, b) and (a, c), we refer to the rotation removing (a, b) and adding (a, c) as $rot(a, b, c)$. The basis of the transformation is then an edge (a, b) and a non-edge (a, c) that

is shown using a dashed line. But vertices b and c can be adjacent. We first find an instance by mapping a with 4, b with 1 and c with 2. Then, we apply the transformation by removing the edge $(4, 1)$ and adding the edge $(4, 2)$, obtaining an application of the transformation. Note that we only show one instance of the transformation, but there are more.



An instance of the transformation An application of the transformation

Figure 2.7: An application of the rotation $rot(a, b, c)$.

In this document, we will use the notation $\tau(G)$ to represent the set of applications of a transformation τ on a graph G. For example, if a transformation τ consists simply of removing an edge, $\tau(G)$ would be the set of all the graphs that can be obtained from G by removing an edge.

As we will see in chapter 4, these transformations will play a great role when proving theorems in extremal graph theory. But first, we introduce this theory in the next chapter.

Chapter 3

Computers and extremal graph theory

In this document, we wish to exploit computers to help researchers with the writing of proofs by transformation in extremal graph theory. But before we can talk about this, we first need to explain what extremal graph theory is. We do so in Section 3.1. We then survey the existing software that can be used in graph theory in Section 3.2. We present the system PHOEG in more details in Section 3.3 as we are currently developing it and for which TransProof, presented in Chapter 5 will be a module. Finally, we give a short conclusion about what is and is not supported by existing software for extremal graph theory.

3.1 An introduction to extremal graph theory

Extremal graph theory is the study of bounds on numerical graph invariants. For example, one might be interested in having an upper bound on the diameter of a connected graph with n vertices. An upper bound can be trivially obtained considering that we can have at most $\binom{n}{2}$ edges in this graph and thus, a diametral path, which is a path whose length is equal to the diameter, cannot use more than $\binom{n}{2}$ edges. The diameter of a graph with n vertices is

thus bounded by $\binom{n}{2}$.

But this bound is not precise, and the attentive reader might already have a better upper bound: the length of a path on n vertices. Indeed, the diameter of a connected graph G on n vertices is the length of an induced path in G and such an induced path can have at most n vertices. Thus, the diameter of a connected graph is bounded by $n-1$ and this bound is tight as there is at least one graph that reaches it: the path on n vertices. This is why, extremal graph theorists often search for extremal graphs, that is, graphs with the maximum or minimum value of the invariant. Finding such graphs and proving their extremality gives bounds that are directly tight.

A conjecture in extremal graph theory for the problem of finding an upper bound can then be defined using three parameters:

- the set of graphs $\mathcal{G}$ satisfying the constraints of the problem such as having n vertices or being connected,
- the invariant $\mathcal{I}$ that is being studied,
- and the set of conjectured extremal graphs $\mathcal{E} \subseteq \mathcal{G} = \{E \in \mathcal{G} \mid \forall G \in \mathcal{G}, \mathcal{I}(G) \leq \mathcal{I}(E)\}$.

This can be easily made into a lower bound by switching the inequality.

To illustrate this, we consider the invariant called the irregularity of a graph, which is a measure of how far a given graph is from being regular. There are different invariants achieving this goal [10, 34, 50, 89]. The one we will use was introduced by Albertson *et al.* [4], and uses the imbalances of the edges.

Definition 5. Let G be a graph and $(u, v) \in E(G)$ be an edge between vertices u and v of G. We define the *imbalance* of (u, v), denoted by $\text{imb}_{(u,v)}$, as the absolute value of the difference between the degrees of u and v ($\text{imb}_{(u,v)} = |deg_G(u) - deg_G(v)|$).

imbalance

The *irregularity* of a graph G is then simply the sum of the imbalances of its edges. One should note that since the graphs considered here are undirected graphs, the edges (u, v) and (v, u) are the same edge and are only counted once. irregularity

Definition 6. Let $G = (V, E)$ be a graph. The *irregularity* of G, denoted by $\mathrm{irr}(G)$, is the sum of the imbalances of its edges:

$$\mathrm{irr}(G) = \sum_{(u,v)\in E} \mathrm{imb}_{(u,v)} = \sum_{(u,v)\in E} |deg_G(u) - deg_G(v)|.$$

Hansen *et al.* showed the following theorem which is a good example of a result in extremal graph theory.

Theorem 1 ([53])**.** *Given a graph G with n vertices and m edges,*

$$irr(G) \leq d(n-d)(n-d+1) + t(t-2d-1),$$

where

$$d = \left\lfloor n - \frac{1}{2} - \sqrt{\left(n - \frac{1}{2}\right)^2 - 2m} \right\rfloor,$$

and

$$t = m - (n-d)d - \frac{d(d-1)}{2}$$

with equality if and only if G is a fanned split graph.

Definition 7 ([53])**.** Given three integers n, d and k such that $n \geq d$ and $k \leq (n-d-1)$ a *fanned split graph* is a graph composed of a clique of order d and a stable set of order $n-d$ where every vertex of the stable set is adjacent to every vertex of the clique. If $k > 0$, one of the vertices of the stable set will be made adjacent to k other vertices of the stable set. fanned split graph

A fanned split graph can be seen in Figure 3.1. It has 6 vertices and 12 edges. Its clique is of order 3 (round vertices) and its stable set (squared vertices) as well, except vertex 4 is adjacent to vertex 5. Thus, it has a value of 3 for d and 1 for k.

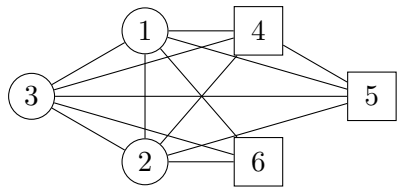


Figure 3.1: A fanned split graph with $n = 6, m = 12, d = 3$ and $k = 1$.

For Theorem 1, if we use the previously introduced notation, $\mathcal{G}$ would be the set of all graphs with n vertices and m edges, $\mathcal{I}$ is the irregularity and $\mathcal{E}$ is the set containing only the fanned split graph of order n and size m.

Another example is Turán's theorem [87] stating that, given two integers n and r ($r \leq n$ and $n = qr + s$), the Turán graph $T(n, r)$ is the graph with the maximum number of edges that does not have K_{r+1} as a subgraph. The Turán graph is the graph composed of s stable sets of order $q + 1$ and $r - s$ stable sets of order q where each vertex is adjacent to every other vertex that is not part of the same stable set.

For this theorem, $\mathcal{G}$ is the set of all graphs of order n that do not contain a clique of order $r + 1$, $\mathcal{I}$ is the size of the graphs and $\mathcal{E}$ is the graph $T(n, r)$.

Given $\mathcal{G}$ and $\mathcal{I}$, the hardest part of conjecturing is to define the set $\mathcal{E}$. This is where computers can be used to speed up the research thanks to their computational power. Different computer systems developed with this intent as well as systems that can be generally useful in this domain are reviewed in the next section.

3.2 Computer-assisted research, a state of the art

In extremal graph theory, it is often difficult to give general bounds on graph invariants, as the number of possible graph configurations infinite. And even fixing some invariant such as the number of vertices might not help: there are already more than 11 million non-isomorphic graphs with 10 vertices and the amount increases exponentially (see Table 3.1).

Just computing some invariant for all those graphs is already no trivial

Table 3.1: Number of non-isomorphic graphs with a given order [63]

order	number of graphs
0	1
1	1
2	2
3	4
4	11
5	34
6	156
7	1 044
8	12 346
9	274 668
10	12 005 168
11	1 018 997 864
12	165 091 172 592
13	50 502 031 367 952
14	29 054 155 657 235 488
15	31 426 485 969 804 308 768
16	64 001 015 704 527 557 894 928
17	245 935 864 153 532 932 683 719 776
18	1 787 577 725 145 611 700 547 878 190 848
19	24 637 809 253 125 004 524 383 007 491 432 768

task. But if we add the fact that many graph invariants are NP-hard to compute [47] and that we need to compare those values to obtain a valid bound, it can become quite tedious to study graph invariants. Fortunately, while computing an invariant for all graphs is impossible, computers nowadays are fast enough to handle many graphs in a relatively short amount of time. This and other more advanced ideas have been exploited over the years by different software in order to produce and study conjectures.

The first one was GRAPH [28] in 1981. This tool allowed its user to consult its database of invariants, compute those for given graphs, compare different graphs, etc. While it does not produce conjectures on its own, this system came with an automated theorem prover using the properties known by the system about graphs, invariants, etc. A new, simplified version called newGraph [12], which does not include the theorem prover component, was developed in 2003 to be easier to use and more efficient. Another system allowing the user to study invariants and the relationships between them is GrinvIn [81] which aims to be used in teaching mathematical reasoning to students while still being useful in research.

For conjecturing, some tools go one step further and try not only to help with the exploration of graph invariants, but also with the generation of conjectures in extremal graph theory. Graffiti [43] produced more than 7,000 conjectures on its first run thanks to a database of graphs used to compare invariants by using heuristics. Later, a simpler variant, Graffiti.pc [30], was built with the objective to be easier to use.

But while producing a large number of conjectures is impressive, it can make it difficult to choose a conjecture to work on Other tools with less automation were built, and different strategies were used. AutoGraphiX [22] uses the variable neighborhood search to try to find conjectures about selected invariants or to refute them. This tool can also prove some trivial conjectures automatically. Another tool using meta-heuristics is Digenes [2] which uses a genetic algorithm to produce conjectures, but this time on directed graphs. GraPHedron [75] and its successor PHOEG [31] (presented in section 3.3) produce conjectures using a different, geometric based, approach presented in Section 3.3.

Because of the high number of NP-hard problems in graph theory, there are also databases of invariant values for all graphs or for interesting ones to avoid having to compute these invariants each time. PHOEG uses a database of invariant values for all the graphs up to order 10 and more for some specific graph classes such as trees or planar graphs. The database House of

Graphs [13] provides researchers with data about many graphs involved in conjectures or theorems, as well as their value for a number of invariants. Some systems rather store relationships between graphs, such as GraphsIn-Graphs [19] which is a database of graphs and their subgraphs.

Most of those tools aim towards the study or generation of conjectures in extremal graph theory. Some, like PHOEG or AutoGraphiX can try to provide counter-examples if a conjecture is wrong. AutoGraphiX can also generate proofs for trivial conjectures. The system GRAPH offers automated proving, but only using logical inference, which requires a full database of facts and properties. This requires to update this database with every new proven result, but also to be able to detect when two properties are equivalent. Hence, despite using the term automated, this system still requires user interaction.

There are already proofs involving computer software. A widely known example is the four color theorem [7]. The authors of this proof used graph theory to reduce the number of possible counter-examples to a finite set of configurations and a computer program to check them. This theorem has also been proved later using the Coq system [48].

Those proofs though require the computer system to be perfectly correct. Indeed, a simple bug in the software could mean that the proof is, in fact, wrong. The proof of the four-color theorem was not accepted by everyone when it was published because of this [92]

General proof assistants exist like Coq [9] or Isabelle [90] which provide specific languages and functionalities to write formal proofs in any domain defined in their language. The proofs are based on modifying, rewriting, decomposing propositions in order to end up with trivial or known results. They can be used both to help in the logical writing of a proof and in checking a proof

In proof assistants, the purely mathematical aspect of these systems can also sometimes hinder the researcher by locking the proof in one direction while some less formal but still correct argument could work. To help with a proof, instead of focussing on the logic of the arguments, one could also try

to look for counter-examples of some proof idea, just like enumerating helps to study conjectures. In extremal graph theory, graph transformations are often used in proofs called "proofs by transformation" which will be explained in more details in Chapter 4. For these proofs, it can be difficult to define a transformation with the properties required by the proof. While there are many tools to study graph invariants, there is little computer support to study transformations.

Transformations are not only used in proofs but also in graph generation [14, 15, 73] and also studied on their own [64, 71]. This is why, we wish to offer a system allowing the user to study graph transformations by using the power of computers in order to efficiently generate these transformations and check their properties.

3.3 PHOEG Helps to Obtain Extremal Graphs

The content of this section comes from the published paper [31] which we wrote to present this system and its ideas. Minor changes have been made to better fit in this document.

As presented in section 3.2, there are many tools to help with research in extremal graph theory with many different ideas and techniques. In 2008, Mélot presented GraPHedron [75], using an exact approach on small graphs. Its central idea is to use all graphs up to some order in the invariant space, and then compute the convex hull of these points. The facets of the hull can be seen as inequalities between the chosen invariants and the vertices of the convex hull as extremal graphs.

While tools such as AutoGraphiX and Graffiti evolved over the years [21, 30], GraPHedron did not. This is why we started a complete overhaul of this tool. This successor, PHOEG[1], contains a set of tools aimed at speeding up the testing of ideas and helping raise new ones. It is mainly composed of a database of graphs enabling fast queries and computations, but also of a module named

[1]PHOEG stands for PHOEG Helps to Obtain Extremal Graphs.

TransProof whose goal is to assist in finding proofs for the conjectures. This module will be presented in details in Chapter 5. PHOEG is already being used in scientific research and has led to several results [32, 55, 59, 60].

In the following sections, we present the different aspects of PHOEG and its new features with respect to GraPHedron and explain some of the main ideas used to help researchers in studying extremal graph theory.

3.3.1 Invariant space

In extremal graph theory, many results are expressed as inequalities between graph invariants [6, 43]. This observation led to the idea used by GraPHedron.

Given at least two graph invariants, GraPHedron represents each graph as a point in the invariant space with their values for the chosen invariants as coordinates. The facets of the convex hull for all those points are then inequalities between the invariants. This is also a different point of view from AutoGraphiX as instead of using meta-heuristics to find an arbitrary large graph, GraPHedron focuses on exact methods but is restricted to a larger amount of smaller graphs (usually, all graphs up to 10 vertices using precomputed values and up to 12 vertices on the fly). Both approaches are thus inexact, as GraPHedron could miss a counter-example if it has too many vertices. Nevertheless, this idea has led to several results [1, 17, 18, 23, 26, 58].

The figure 3.2 shows a visual example of this process for the eccentric connectivity index (ξ^c) which is presented in Chapter 6. The points correspond to the connected graphs of order 7 in the invariants space (the size m and ξ^c) and the color of each point represents the maximum diameter among the graphs with same coordinates, that is, the largest diameter among all graphs whose size and value of ξ^c correspond to the coordinates of the point.

One of the specificities of PHOEG is that it is possible to explore the inner points of the polytope. For example, GraPHedron did not have the possibility to obtain the diameter of the graphs inside the convex hull.

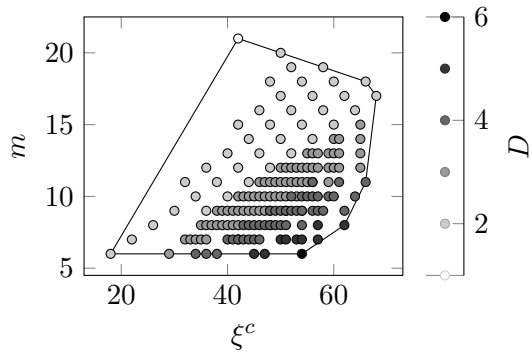


Figure 3.2: Polytope and graph coordinates for the eccentric connectivity index (ξ^c) invariant with fixed order ($n = 7$) and parameterized size.

3.3.2 Invariants database

We stated in section 3.2 that while it is fast and easy to compute some invariants on some classes of graphs, others can be computationally costly. This is especially inconvenient when there are millions of graphs to consider. This problem was tackled in GraPHedron by only computing their values once for each graph and then storing their values in files.

As invariants are constant by isomorphism, each graph is only considered once thanks to its canonical form computed by the Nauty software [74].

In PHOEG, the data storage and query answering are delegated to a relational database management system. Complete finite classes of graphs, alongside their invariants values, are listed in tables, *e.g.,* all non-isomorphic graphs, trees, claw-free connected graphs, up to some order.

The goal of this tool is to be used by researchers in graph theory. As they are not necessarily accustomed to the writing of SQL queries, the addition of a web interface is currently underway to provide ease of use.

3.3.3 Forbidden subgraph characterization

In graph theory, classes of graphs are often described by means of a forbidden graph characterization. Such a characterization of a class $\mathcal{G}$ of graphs

is given by an obstruction set $\mathcal{O}$ containing the forbidden graphs. A graph G is a member of $\mathcal{G}$ if and only if it has no element of $\mathcal{O}$ as substructure (*e.g.,* induced subgraph, graph minor). A new feature in PHOEG (mainly developed by Pierre Hauweele) provides a tool addressing this matter. The substructure relations define a preorder. Given a finite class of graphs or a class membership function and a specific substructure relation, PHOEG computes the minimal graphs (not) in the class for this relation. These minimal graphs are obtained by iteratively removing graphs that are either supergraphs of some other graph in the class or that are not subgraphs of one or more graphs in the class. If the iteration of the class satisfies the natural order relation of the couple (order, size) of the graphs, then the graphs added to the accumulator are actual minimals and therefore do not need to be ruled out. The output set of minimal graphs provides a proposed obstruction set for the forbidden graph characterization of the input class.

3.4 Conclusion

As we saw in this chapter, the first step in extremal graph theory is finding the structure of extremal graphs. This step benefits from existing software that can provide researchers for candidates extremal graphs thanks to different methods. One of these tools is PHOEG, which we are developing, and which uses an exact approach on large quantities of small graphs. This allows researchers to extract information about invariants and structures of graphs.

The next step in extremal graph theory, proving the conjecture, does not have as much tooling. A reason for this is the need to translate formal mathematics into a software usable by all. But restricting ourselves to specific proving methods, it becomes possible to provide assistance in writing a proof by exploiting the speed of computers. To this extent, we will focus on proofs by transformations, which are presented in the next chapter.

Chapter 4

Proofs by transformation

This chapter presents the notion at the heart of the thesis the proof by transformation. We first explain and present the ideas of a proof by transformation. We illustrate this by giving different examples from the literature. We then formalize them using the concept of the metagraph of transformations.

4.1 Analysis of proofs by transformation

Once a conjecture in extremal graph theory has been written, it needs to be proven or disproved. If the conjectured bound is tight, this boils down to proving that the conjectured extremal graphs are indeed extremal.

To prove that a graph is extremal, a common method that often appears in the literature is the proof by transformation. Among the many ways Turán's theorem has been proven, there exists a proof by transformation [3]. This type of proof works by showing that, for any non-extremal graph, we can transform it into a new graph with a value for the invariant that is closer to that of the extremal graphs. Note that, since the invariant value is different, this new graph is not isomorphic to the previous one.

For example, given a connected graph G with n vertices and diameter λ that is not a path, we can take an induced path $v_1, \ldots, v_\lambda$ of length λ in G. Then, we can take any vertex x not part of this path, completely disconnect it

and then add an edge between x and v_1. If this new graph is connected, it has diameter $\lambda + 1$ and thus, G is not extremal with respect to the diameter. If this works for any connected graph other than the path, we have shown that all the other connected graphs have a smaller diameter than the path.

But of course, this transformation is not always correct. While the length of the path will increase, we could obtain a disconnected graph where the diameter is not defined. In general, finding transformations that will always work is no trivial matter.

Let us consider another problem: finding an upper bound on the irregularity of a graph with n vertices and m edges.

Using one of the tools for computer-assisted conjecturing mentioned in section 3.2, we can find that the extremal graphs are fanned split graphs as stated by Theorem 1 (p. 19). This theorem was proven using two transformations that are specific rotations.

Let G be a graph of order n and size m that is not a fanned split graph. Let us start by defining D as the set of vertices of degree $n-1$ in G and d the number of such vertices ($d = |D|$). We call the vertices in D the *dominating vertices* of G.

Let w be a vertex of maximum degree in $V(G) - D$. We know this vertex exists since G is not a fanned split graph and therefore, $D \cup V(G) \neq \emptyset$. Since w is not a dominant vertex, G also contains a vertex u that is not adjacent to w and not dominant.

Hansen *et al.* showed in their proof of Theorem 1 that, if u has a neighbor v that is not dominant, the graph obtained by the rotation $rot(u, v, w)$ has a higher irregularity than G. This shows that, for any graph G to which the transformation can be applied, we can obtain a new graph with a higher irregularity and that G is therefore not extremal. An example of this rotation is show in Figure 4.1.

Now, if u is only adjacent to dominant vertices and G is not a fanned split graph, there is at least one edge (v_1, v_2) ($deg_G(v_1) \geq deg_G(v_2)$) such that $\{v_1, v_2\} \not\subseteq D \cup \{w\}$. Otherwise, every non-dominant vertex that is not w is

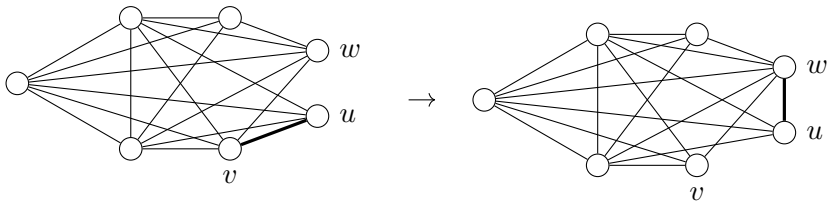


Figure 4.1: The first rotation of the proof of Theorem 1.

adjacent to only dominant vertices and possibly w and G would be a fanned split graph.

In this case, if we applied the rotation $rot(v_1, v_2, u)$, the graph G' produced by this rotation fits the conditions to apply the rotation $rot(u, v_1, w)$ and we would obtain a new graph G'' with a higher irregularity than G'. This time, G' might not have a higher irregularity than G but Hansen *et al.* showed that $\text{irr}(G) \leq \text{irr}(G')$ and thus, $\text{irr}(G) < \text{irr}(G'')$. Again, all the graphs where this second rotation applies cannot be extremal, since it was shown that there exists a graph G'' with a higher irregularity but with the same order and size. This second rotation is illustrated in Figure 4.2. One might notice that the result of this second rotation is exactly the graph being transformed by the first rotation in Figure 4.1.

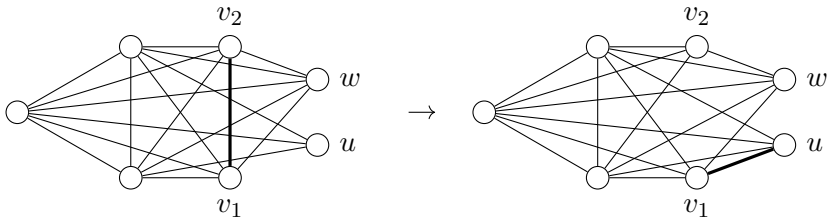


Figure 4.2: The second rotation of the proof of Theorem 1.

From those two rotations and their properties, every graph with n vertices and m edges that is not a fanned split graph fits the conditions of one of the two. Thus, for every graph G with order n and size m that is not a fanned split graph, there exists a graph G' with same order and size but higher irregularity. This proves that the maximum irregularity can only be reached by a fanned

split graph on n vertices and m edges.

This example shows that we cannot simply choose a generic transformation, but that we also need to see when this transformation works. While some cases can be solved by applying the same transformation several times, others might require different transformations depending on the situation.

In their paper, Hertz *et al.* use 42 different transformations [57]. If $\mathcal{G}$ is the set of all the graphs covered by their theorem and $\mathcal{E}$ is the set of the extremal graphs ($\mathcal{E} \subseteq \mathcal{G}$), we denote by $\mathcal{G}_i$ the set of graphs that were not proven to be non-extremal after applying transformations 1 to i. We also define $\mathcal{G}_0 = \mathcal{G}$. The way these transformations work is by restricting the set of possibly extremal graphs until this set only contains the extremal ones, that is, for some transformation i, $\mathcal{G}_i \subset \mathcal{G}_{i-1}$ and $\mathcal{G}_{42} = \mathcal{E}$. Finding the special cases of a transformation and solving them can thus be a complex task.

But even with those cases identified, simple transformations are not always enough. Some invariants might require more than adding an edge or applying a rotation. Sometimes they can be as complex as replacing a part of a graph with an entirely different subgraph. Hansen *et al.* show that the average distance between two vertices of a connected graph is at most half of the maximum order of an *induced forest*, that is, an induced subgraph graph without a cycle [51]. They do so using a proof by transformation with 5 transformations. An example of each of these transformations is shown in Figure 4.3. Transformations 1, 3 and 5 are quite simple. Transformation 1 consists in removing an edge, Transformation 3 is a rotation and Transformation 5 is the removal of a vertex. Transformation 4 is already more complex as it involves removing all the neighbors of a vertex and then adding an edge between this vertex and all the vertices of a clique. The second transformation is completely different from what we saw up to now as it does not consist in removing and adding specific edges or vertices, but it removes a subgraph entirely to replace it with a different structure. In Figure 4.3, the C_4 on the left is replaced by a C_3 with a pendant vertex.

induced forest

Therefore, some tool to check as many configurations as possible can be

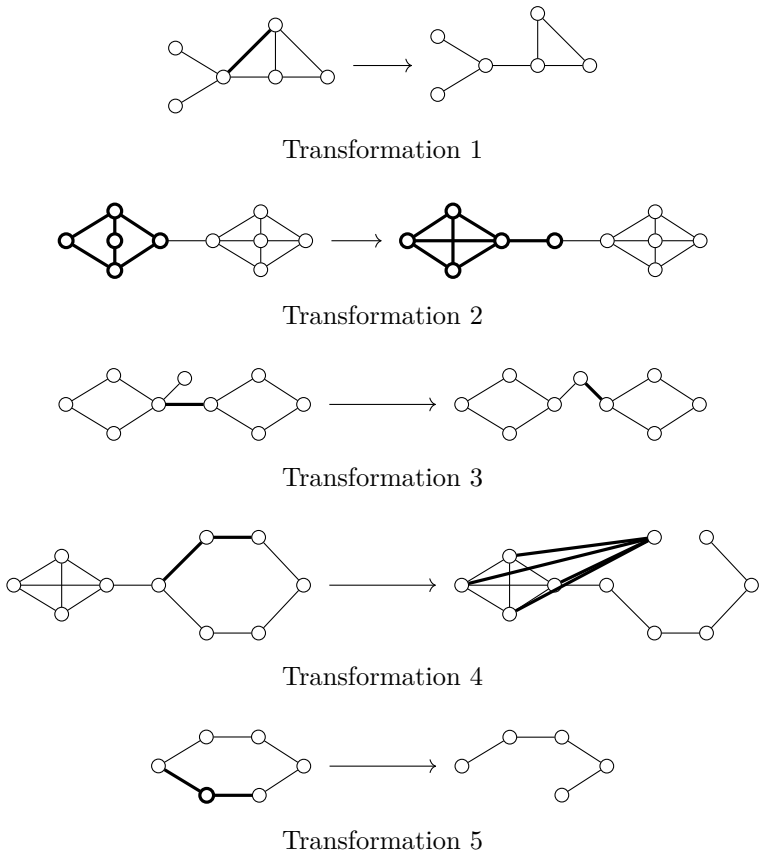


Figure 4.3: Examples of the transformations used in the paper from Hansen *et al.* [51].

a great help in finding the special cases of a transformation in order to refine them or maybe to find examples where a transformation does not work. For example, cases when it will produce a graph with a lower value for the studied invariant instead of a higher value.

4.2 The metagraph of transformations

Given a set of graphs $\mathcal{G}$ and a set of transformations $\mathcal{T}$, we can build a directed graph with vertex set $\mathcal{G}$ and where an arc from vertex A to vertex B

means that there is a transformation $\tau \in \mathcal{T}$ such that for the graphs A and B, $B \in \tau(A)$. We call this graph, the *metagraph of transformations* or metagraph for short.

Definition 8. Let $\mathcal{G}$ be a set of graphs and $\mathcal{T}$ be a set of graph transformations. Let M be the directed graph with vertex set $\mathcal{G}$ and arc set $\{(G, U) \in \mathcal{G} \times \mathcal{G} \mid \exists \tau \in \mathcal{T} \wedge \exists H \in \tau(G), H \simeq U\}$. This graph is the *metagraph of transformations* for the given graph set $\mathcal{G}$ and transformation set $\mathcal{T}$.

metagraph of transformations

Figure 4.4 shows such a metagraph with the set of graphs with 5 vertices and 5 edges as vertices. The set of transformations used is the set containing only the first transformation from the proof of Theorem 1. The irregularity of each graph is written next to it. Note that while this transformation is a rotation, not every rotation satisfies the constraints defined by the transformation. Thus, this is not a metagraph for the rotation but a subgraph of it.

By analyzing this metagraph, we can easily see that this only rotation is not enough to prove the theorem, but for the graphs on 5 vertices and 5 edges, there is no situation where the rotation does not strictly increase the irregularity.

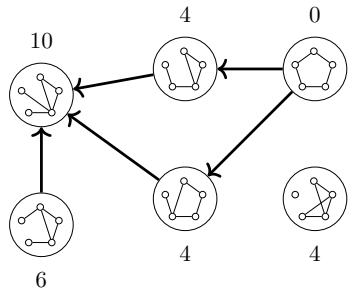


Figure 4.4: Metagraph representing the first transformation of proof of theorem 1 on graphs with 5 vertices and 5 edges.

With this representation, it becomes possible to study properties of graph transformations on a set of graphs by studying a directed graph. This idea has

already been used to study transformations properties, such as the diameter of a transformation, which is the diameter of the metagraph [20, 42].

In our context, we can use this metagraph to help build a proof by transformation. If the conjecture we are trying to prove is for an upper bound on invariant $\mathcal{I}$ over the set of graphs $\mathcal{G}$, writing a proof by transformation for this conjecture requires finding a set of transformations $\mathcal{T}$ such that for any non-extremal graph G in $\mathcal{G}$, there is a transformation τ in $\mathcal{T}$ and $\mathcal{I}(G) < \mathcal{I}(\tau(G))$. Since we have a set of graphs and a set of transformations, we can build a metagraph. Then, in order to be able to write a proof by transformation using $\mathcal{T}$, the metagraph needs to have the following two properties:

1. every arc (A, B) from the metagraph is such that $\mathcal{I}(A) < \mathcal{I}(B)$,

2. the extremal graphs are the only vertices of the metagraph without outgoing arcs, that is, without arcs leaving this vertex.

The first property means that there is no situation where applying a transformation will not increase the invariant. This would be problematic, as the goal is to show that every non-extremal graph can be transformed into a new graph with a higher value for the invariant. This property also implies that the graph is acyclic because of the strict inequality. For the sake of readability, we will refer to arcs in the metagraph respecting the first property as *improving arcs*.

improving arcs

The second property makes sure that the chosen set of transformations suffices to show the non-extremality of all the graphs in $\mathcal{G}$ by imposing that there is no non-extremal graph for which we cannot find a graph with a higher value of the invariant. Also, since every arc is supposed to be an improving arc from the first property, if one extremal graph has an outgoing arc, the conjecture is wrong because there is a graph in $\mathcal{G}$ with a higher invariant value than the extremal graph.

If a chosen set of transformations does not produce a metagraph with those properties, it is therefore useless to try and use only those transformations in a

proof. Looking at the metagraph can thus provide the researcher with counter-examples to their proof idea and save them time they would have spent trying to write it. If, on the other hand, no counter-example is produced, it can reinforce their ideas that the proof by transformation would work.

It would be valuable to have a tool allowing researchers to try their ideas beforehand and explore the metagraph more in depth. Such a tool is currently being developed and is presented in the next chapter.

Chapter 5

TransProof

As we explained in section 4.2, being able to explore the metagraph formed by a set of graphs $\mathcal{G}$ and a set of transformations $\mathcal{T}$ would be helpful to researchers in order to check if their idea of a proof by transformation is sound. This is why we are currently developing TransProof, a tool to compute this metagraph and provide ways to explore it[1].

The next section illustrates how, having computed a metagraph of transformations, one can analyze it to find potential flaws in a proof by transformation. The following sections explain the ideas used by TransProof to provide help to researchers. The second section presents different methods to remove redundancy in the metagraph in order to make it easier to store and study. Section 5.3 explores the different types of databases that can be used to store a metagraph, as well as to provide powerful support for queries. In section 5.4, we talk about different possibilities to provide a user-friendly way for researchers to define their own transformations without the need to know a programming language. Finally, the last section explores ideas that can be used to extract more information from the amount of data produced by TransProof.

[1]The source code for TransProof is available at `https://github.com/umons-dept-comp-sci/PhoegTransRust` and the source code for the custom graph library developed alongside it at `https://github.com/umons-dept-comp-sci/PhoegRustGraph`.

5.1 Introduction

The main use of TransProof is to compute a metagraph for a given set of graphs and transformations. This metagraph is then stored in a database which can be queried to extract information.

Those queries allow checking if there are no counter-examples to an idea of a proof by transformation (*i.e.,* arcs that are non-improving). But thanks to the query language offered by the database, more complex queries can be used. For example, it is possible to know if there is at least one improving arc for each non-extremal graph or if some non-extremal graph cannot be improved.

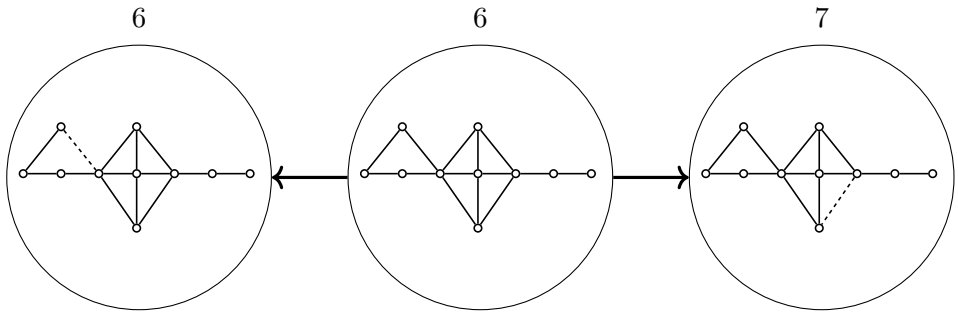


Figure 5.1: Removing an edge can increase the diameter or not.

For example, in Figure 5.1, the graph in the middle has a diameter of 6 and removing an edge can produce several non-isomorphic graphs. It can produce a graph with the same diameter (the left one) and a graph with a higher diameter (the right one). This is an example where exploring the metagraph can show that the selected transformations need to be refined or constrained by adding conditions to keep only those that increase the diameter. Indeed, here we know that removing an edge might not always work, but we know that, for the graph in the middle, there is an edge that will increase the diameter if removed and that removing an edge might be an interesting lead.

But if we consider the graph of Figure 5.2, any removal of a single edge will not change the diameter. However, removing the two edges $(2, 5)$ and $(3, 5)$ produces a path on 5 vertices with a higher diameter.

Thus, beyond verifying if a given transformation always works to improve an invariant, another possibility is to check whether a combination of several transformations will always improve the invariant when a single transformation does not suffice. For example, if removing an edge does sometimes decrease the irregularity, maybe removing two edges or removing an edge and then performing a rotation will work.

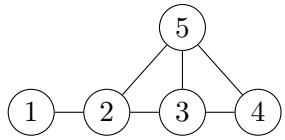


Figure 5.2: Removing any edge without disconnecting the graph will not increase the diameter.

Of course, it is not possible to build a metagraph with an infinite number of vertices and TransProof cannot guarantee that the studied transformations suffice to build a proof. But only considering graphs with small order (up to 10 vertices) can already reveal problems and provide information to refine the proof and, hopefully, correct it.

5.2 Removing symmetries

When computing transformations for a metagraph, the number of instances on a single graph can be exponentially large. For example, removing an edge on a complete graph of order n in a naive fashion will produce $n(n-1)$ results as each edge can be counted in both directions. Of course, if we number the vertices, we can simply impose that an edge (x, y) is removed only if $x < y$. But we still have $\frac{n(n-1)}{2}$ cases. For a small single graph, this might not be a problem. But if we consider this increase multiplied by the number of graphs on n vertices, the amount of data produced quickly becomes too large to handle even with modern computers. Only considering the connected graphs with 10 vertices would produce at least $9 \cdot 11\,716\,571 = 105\,449\,139$ or more

than a hundred million of results[2].

But, for example, in a complete graph, all the edges are the same with regard to the transformation. No matter which edge we remove, we obtain the same result. The same problem arises to a smaller extent with other graphs. In the graph shown in Figure 5.3, removing the edge $(3, 4)$ or the edge $(2, 5)$ would give identical results, as we are disconnecting a pendant vertex adjacent to a vertex of a clique of order 3 in both cases.

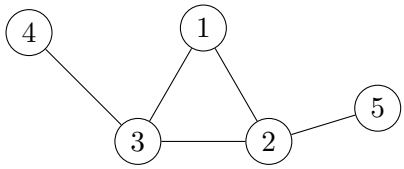


Figure 5.3: In this graph, vertices 4 and 5 are symmetrical.

However, the problem is more complex than just obtaining the same result. In fact, two different transformations can produce the same graph. In Figure 5.4, we can see two different rotations on the same graph, with the bold edge being removed and the dashed edge being added. These two rotations produce the same graph, but they are not similar. In the left case, we are only modifying vertices of degree ≥ 2 but in the right case, the pendant vertex is made adjacent to another vertex. A researcher interested in pendant vertices could consider only the right case, while a researcher working on an invariant that is not modified by pendant vertices might be interested in only the left case. It is not enough to simply discard instances that produce the same graph.

To obtain the symmetries of a given graph, we will need to consider all the automorphisms of a graph G. The set they form is called the *automorphism group* of G and is actually a set of permutations of the vertex set, denoted by $Aut(G)$. In this section, we will use the cycle notation to represent a permutation. For example, the permutation of the set $\{1, 2, 3, 4\}$ that permutes 1 with 3 and keeps 2 and 4 at the same place will be noted $(1\,3)(2)(4)$.

automorphism group

[2]Every connected graph on 10 vertices has at least 9 edges.

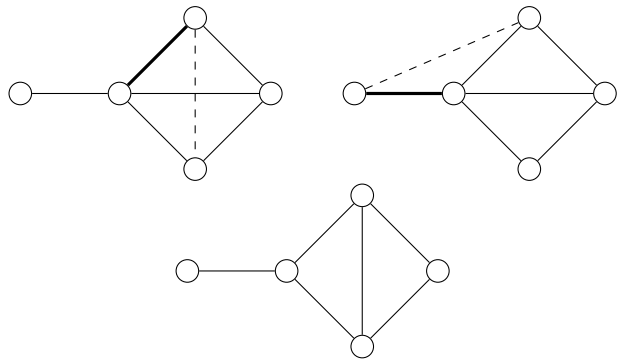


Figure 5.4: Both of the different transformations of the graph (top) produce the same graph (bottom).

It is possible to group vertices together based on whether there exists an automorphism in the automorphism group that permutes them. Given a vertex v in a graph $G = (V, E)$, the *orbit* of v is the set $\{w \in V \mid \exists p \in Aut(G)$ such that $p(w) = v\}$. Since permuting every vertex with itself is also an automorphism, the orbit of a vertex contains this vertex. Also, because the inverse of a permutation that is an automorphism is also an automorphism, if a vertex w is contained in the orbit of another vertex v, then v is contained in the orbit of w. Thus, if two vertices are in the orbit of one another, their orbits are the same subset of vertices. The set of the orbits of all the vertices of G forms a partition of the vertex set of G. We call this partition the *orbits* of G.

orbit of a vertex

orbits

In Figure 5.5, we see the only two automorphisms of the graph. The vertices 4 and 5 are permuted together, as well as the pair of vertices 3 and 2. The vertex 1 is never permuted. Thus, we have three orbits: one composed of vertices 4 and 5, one with 2 and 3 and a last one containing only 1. We will note the orbits using the following notation: $\{\{1\}, \{2, 3\}, \{4, 5\}\}$.

In this section, we present three methods that can be used to remove the symmetries when computing transformations. The first one, the *iterating method*, uses the orbits when iterating over the vertices of the graph to avoid generating symmetrical instances of a transformation. The second method,

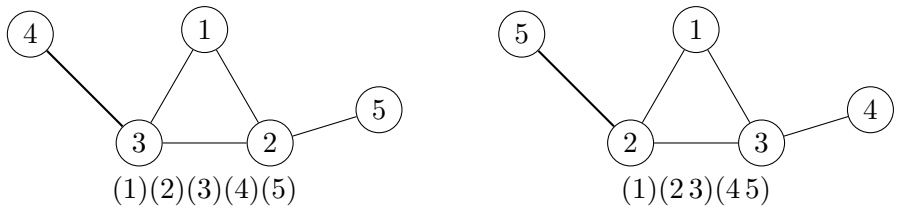


Figure 5.5: These two permutations are the only automorphisms of the graph G of Figure 5.3, which means that the orbits are $\{1\}, \{2, 3\}$ and $\{4, 5\}$.

the *hypergraph method* uses a different approach and instead produces a special type of graph called a hypergraph that will then be used to remove the symmetries. The *subgraph method* is an adaptation of a subgraph isomorphism algorithm to include the idea of the *iterating method*. The three approaches have different strengths and weaknesses. We thus give a comparison is given in Subsection 5.2.4.

5.2.1 The iterating method

Using the instances of a transformation without any filtering of symmetries, a naive approach would be to simply consider all the possible pairings of a vertex in the basis and a vertex in the graph to which we want to apply a transformation. More precisely, to find an instance of a transformation in a graph, we would normally try to map the first vertex of the basis of this transformation with every vertex of G. And for every mapping, we would try to map the second vertex of the basis to another vertex of G and so on.

For example, if we try to apply a transformation with P_3 as a basis, we will look for all triplets of vertices (u, v, w) in G that form a path. Algorithm 1 does so in a naive fashion. If we label the vertices of the basis as p_1, p_2 and p_3 with p_1 and p_3 being the two vertices with degree 1, this algorithm will iterate over every vertex u of G to try mapping u to p_1. For each mapping of p_1, it will iterate over every vertex v of G to map it to p_2. And for every mapping of p_1 and p_2, the algorithm will search for a vertex w that could be mapped to p_3.

Algorithm 1: Algorithm $NaivePath3(G)$

Input: A graph G

Output: The paths of order 2 in G

```
1 foreach vertex u in G do
2 |   foreach vertex v in G adjacent to u do
3 |   |   foreach vertex w in G adjacent to u and not adjacent to w do
4 |   |   |   output the path (u, v, w)
5 |   |   end
6 |   end
7 end
```

This algorithm will produce all the induced paths of order 3 in G and can then be used to compute the instances of the transformation. But obviously, there will be duplicates. If we permute the two extremities of a path of order 3, we still obtain a path of order 3, using the same set of vertices. This is an automorphism. Note that, because we are looking for instances of a transformation, the path $(1, 2, 3)$ and the path $(3, 2, 1)$ might not be equivalent with regard to the transformation. For example, the transformation could consist in adding a vertex pendant to the first vertex of the path.

Given a vertex x in the basis of a transformation, instead of trying to map it to every vertex of G, we can consider only one vertex per orbit of G. Indeed, if two vertices a and b of G are in the same orbit, there is an automorphism of G that permutes a and b. Thus, all the instances where x is mapped to b can be obtained from an instance where x is mapped to a simply by applying an automorphism of G. Thus, in Figure 5.5, we only need to consider vertices 1, 2 and 4.

But this is not enough and would not work. In our example, we would never consider the edge $(2, 5)$ or the edge $(3, 4)$ as they both contain a vertex that is not part of the considered vertices. This is because once we select some elements of the graph, not all automorphisms remain valid. For example, if we select vertex 2 as the first vertex of an edge, we can no longer allow

automorphisms that permute 2 with another vertex. Indeed, we would then be searching for an edge $(2, v)$ and thus, we must only consider vertices v that are adjacent to 2 and not to some vertex in the same orbit as 2. Actually, the orbits are different. In fact, in the example, once 2 is selected, no symmetries remain as there are no automorphisms that do not permute 2 with another vertex except for the automorphism that permutes every vertex with itself. Thus, the orbits become $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$. When computing the orbits of a graph, we will sometimes need to restrict the automorphisms we consider by ignoring those that permute a vertex v with another vertex. We will call this operation *fixing* the vertex v. And when computing the orbits of a graph G, we will then only consider automorphisms p such that for every vertex v that was fixed, $p(v) = v$ instead of using all the automorphisms of G.

fixing

The procedure to find instances of a transformation in a graph G is then adapted to use the orbits. We will try to map the first vertex u of the basis to one vertex per orbit. For every mapping of u, we will fix u and compute the resulting orbits. These orbits are then the orbits produced by the set of all the automorphisms that do not permute u with another vertex. Then, we will try to map the next vertex v of the basis to one vertex per orbit for the new orbits. After we exhausted the possibilities for v, we unfix u, thereby allowing all the automorphisms permuting u again, and try to map it to another vertex of G. Since we only consider one vertex per orbit of a graph G, we will often use the *orbits set* of G which is the set obtained by taking only the vertex with the lowest index in every orbit of G.

orbits set

Algorithm 2 is an adaptation of the previous one that is now using the orbits of G to find all the instances of a transformation having a path (u, v, w) of order 3 as its basis. It starts by computing the orbits set of G and iterates over it to find candidates for the vertex u. Because we are using the orbits set, we only consider one vertex per orbit. For every vertex of G that is mapped to u, it fixes this vertex and then computes the new orbits set. Thus, when computing the orbits set, we only consider automorphisms such that u is not permuted with any other vertex. The same principle is applied when searching

Algorithm 2: Algorithm $OrbitsPath3(G)$

Input: A graph G

Output: The paths of order 2 in G

```
1  orbits ← the orbits set of G
2  foreach vertex u in orbits do
3      fix the vertex u
4      orbits_u ← the new orbits set of G
5      foreach vertex v in orbits_u adjacent to u do
6          fix the vertex v
7          orbits_v ← the new orbits set of G
8          foreach vertex w in orbits_v adjacent to v and not to u do
9              output the path (u, v, w)
10         end
11         unfix the vertex v
12     end
13     unfix the vertex u
14 end
```

for candidates for the vertex w. When the possible candidates for vertex v have been exhausted, the algorithm moves on to the next candidate for u. This means that the previously fixed vertex can now be permuted again by automorphisms because it is no longer a candidate for u.

To illustrate the procedure, we try to find all non-symmetrical edges (a, b) of the graph in Figure 5.5. The detail is visible in Table 5.1. The first column is the first vertex a of the edge, the second column is the second vertex b and the last column is the orbits of the graph when fixing the selected vertices. Since our graph is undirected, we only consider edges where $a < b$. In the first line, we have not fixed any vertex and thus have the orbits $\{\{1\}, \{2, 3\}\{4, 5\}\}$. We will only consider the vertex with the lowest number in each orbit. Thus, the first vertex of G we consider is 1. Once we are done generating edges starting with 1, we will try using vertex 2 and vertex 4. Vertex 1 being fixed

does not change the orbits, as 1 could not be permuted with any other vertex of G before. The first candidate for b is 2 and since 2 is adjacent to 1, we found an edge $(1, 2)$. This is shown by the gray background for the line. The second candidate for b is 4 which is not adjacent to 1. We did not consider edges $(1, 3)$ or $(1, 5)$ since 3 is in the same orbit as 2 and 5 is in the same orbit as 4 when 1 is fixed.

Once we considered every non-symmetrical edge $(1, b)$, we unfix 1 and select another candidate for a which is 2 according to the orbits. This time, fixing 2 produces orbits where every vertex is alone, and we have to consider all the vertices b such that $2 < b$. The last candidate for a is 4 since 3 is in the same orbit as 2 when no vertex is fixed, and we only explored the vertices starting with 2. But there is no edge $(4, b)$ where $4 < b$. In the end, we generated the three edges $(1, 2)$, $(2, 3)$ and $(2, 5)$. All the other edges are symmetrical to one of those three.

This *iterating method* allows generating instances of a transformation on a graph while avoiding some useless symmetries. But, until now, we considered a transformation impacting only one edge, but transformations can be more complex. It is interesting to note that simply preventing selected vertices from being swapped in a permutation will not filter out all the possible symmetries. Consider a transformation which, given an edge (a, b) and a vertex c that is not adjacent to a and b, removes (a, b) and adds the two edges (a, c) and (c, b).

detour

This is called a *detour*.

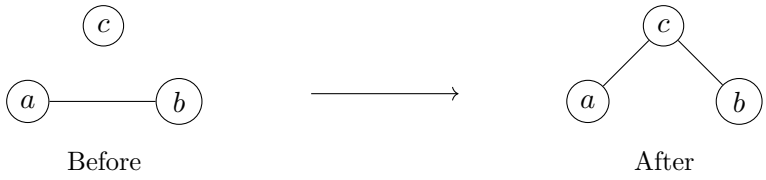


Figure 5.6: Illustration of a detour.

If we try to apply this transformation to the graph in Figure 5.7, we will first select a vertex a and then a vertex b. Suppose we selected vertices 1 and 2, we then need to select a vertex c that is not adjacent to 1 and 2. We can

Table 5.1: Procedure of iterating over the non-symmetrical edges of the graph in Figure 5.3.

first vertex	second vertex	orbits
		$\{\{1\},\{2,3\},\{4,5\}\}$
1		$\{\{1\},\{2,3\},\{4,5\}\}$
1	2	$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
1		$\{\{1\},\{2,3\},\{4,5\}\}$
1	4	$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
		$\{\{1\},\{2,3\},\{4,5\}\}$
2		$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
2	3	$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
2		$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
2	4	$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
2		$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
2	5	$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
2		$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
		$\{\{1\},\{2,3\},\{4,5\}\}$
4		$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
4	5	$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
4		$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
		$\{\{1\},\{2,3\},\{4,5\}\}$

choose either 5 or 6, and they are symmetrical. But since vertices 1 and 2 are already selected, they are fixed, and we ignore the automorphisms that permute them together. There is no non-trivial automorphism that does not permute 1 and 2 but permutes 5 and 6. Because of this, we will obtain the two instances composed of the edge $(1, 2)$ and either vertex 5 or 6 and those two detours are symmetrical. Fortunately, we will still filter some symmetries, such as not trying vertices 5 and 6 as candidates for a.

But here, vertices 5 and 6 are symmetrical with respect to the edge $(1, 2)$

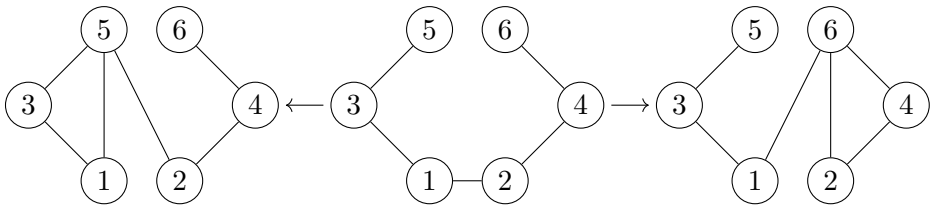


Figure 5.7: If 1 and 2 are fixed, vertices 5 and 6 are no longer symmetrical and we miss two symmetrical transformations.

and not to the vertices. And in this case, using either the edge $(1, 2)$ or the edge $(2, 1)$ for a detour will produce two symmetrical applications of the transformation. Thus, instead of fixing vertices 1 and 2, we could only fix the edge $(1, 2)$. More precisely, instead of considering only the automorphisms that do not permute 1 and 2 with any other vertex, we could relax this restriction and also consider automorphisms that permute vertices 1 and 2 together. This is both an advantage and a disadvantage of the *iterating method*. We can further improve the amount of filtered symmetries, but this improvement requires changes that are specific to the transformation.

Another situation where not all symmetries are filtered happens when we have transformations involving subgraphs. Let us imagine a transformation consisting in linking a vertex to all the vertices of a clique of size 3. Iterating over the orbits set, we could first look for a vertex of this clique, then a second one, then the third one and finally the vertex to link to the clique. But in the graph of Figure 5.8, looking for the first vertex would get us two candidates that are not symmetrical: 1 and 2. Only, those two vertices are part of the same clique and would both give us the clique $\{1, 2, 3\}$. And since there is no difference between the vertices of the clique in the transformation, we would have exactly the same result twice.

This is caused by the fact that we are not looking for three specific vertices that happen to form a clique, but we are looking for a clique without any difference between its vertices. In fact, some transformations such as the rotation are asymmetrical because there are vertices in their basis that cannot

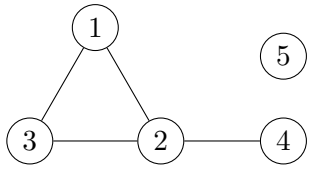


Figure 5.8: Vertices 1 and 2 are in different orbits but they are part of the same K_3.

simply be swapped or the application produced from an instance would be different. In the case of a basis that is simply a clique, all the vertices can be swapped and still produce the same instance. This would be a symmetrical transformation.

Definition 9. Let τ be a transformation with basis H. We say that τ is a *symmetrical transformation* if the orbit of each vertex of H is the whole vertex set of H, that is, if for every pair of vertices u and v of H, there is an automorphism p of H such that $p(u) = v$.

symmetrical transformation

Symmetrical transformations are not handled efficiently by a generic *iterating method*. A more customized method could impose an order on the vertices of that triangle. Therefore, the *iterating method* is not well suited to produce a generic algorithm working for any basis, but would rather serve as a method that can be used to write algorithms for specific transformations. While this is not a problem for someone used to working with symmetries, it would not be user-friendly.

In the next section, we will define a different method that can filter out the symmetries in symmetrical transformations while requiring far less customization in order to remove as many symmetries as possible.

5.2.2 The hypergraph method

The first example given for the *iterating method* was to iterate over the edges of a graph while avoiding the symmetries by using the orbits of the vertices. It is possible to compute the orbits of the edges of a graph G by

constructing a new graph L whose vertices are the edges of G. Such a graph L is called the *line graph* of G and is defined as follows.

line graph

Definition 10. Let $G = (V, E)$ be a graph, the *line graph* of G is the graph L such that each vertex of L represents an edge of G and given two vertices e and f of L, there is an edge (e, f) in L if and only if the edges of G represented by e and f share an extremity.

Figure 5.9 shows a graph G and its line graph. We can see that the edge $(3, 4)$ of G is represented by the vertex 3–4 and is adjacent to the vertex 2–3, representing the edge $(2, 3)$ of G, because the edges they represent are both incident to the vertex 3 of G.

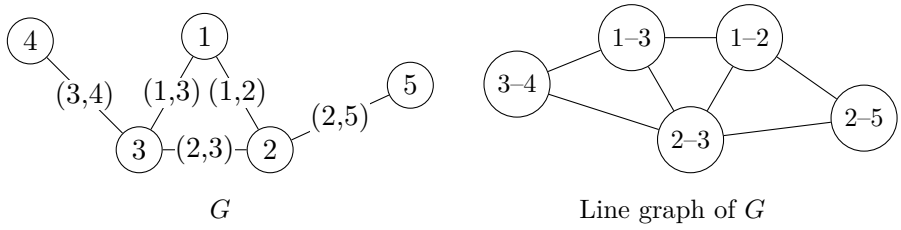


Figure 5.9: The graph of Figure 5.5 and its line graph.

Computing the orbits of the vertices of the line graph of G gives us the orbits of the edges of G. This way, we compute the orbits once and not every time we fix a vertex. In Figure 5.9, the orbits of the line graph are $\{\{1{-}2, 1{-}3\}, \{2{-}3\}, \{2{-}5, 3{-}4\}$. Thus, if we consider one vertex per orbit, we would select $(1, 2)$, $(2, 3)$ and $(2, 5)$ which correspond to the same edges we obtained previously with the *iterating method*.

Using the line graph allows us to compute the orbits only once. This is interesting as there is no know algorithm having a polynomial worst-case time complexity to compute the orbits at the time of writing. But, the number of edges can be larger than the number of vertices. And in this case, computing the orbits of a larger graph once can be slower than computing the orbits of

the original graph n times. Choosing one of these methods should then depend on both the order of the graph and its size.

When the basis of the transformation is more complex than a simple edge, however, the method of the line graph cannot be used as we are looking for a more complex structure than an edge. Thus, we have to adapt the idea of the line graph. Instead of building a graph L where each vertex is associated with an edge of the graph G, we use a graph where each vertex corresponds to an instance of the transformation. This means that we are actually back to the previous problem of generating all the instances of a transformation after generating them. Still, the previous consideration applies, as having a small number of instances of a transformation would produce a smaller graph on which to compute the orbits.

However, the adjacencies of the instances with more than two vertices are not as easy to represent as a graph. Indeed, in the line graph, we know that if there is a path between two vertices, this path is made of edges and vertices which are both represented in the line graph. But if, instead of trying to represent edges, we wish to represent triangles (cliques of order 3), a path between two triangles is made of edges and vertices. But if we want a graph where the vertices are the triangles of the original graph, how do we represent the edges in this path of the original graph? For example, how do we represent the configuration in Figure 5.10 in a graph where vertices can only represent triangles? We cannot simply state that two triangles are adjacent if there is a path between them because we would lose much information about the structure of the original graph such as the length of that path, the fact that two paths cross, the adjacencies of the vertices of the path, etc.

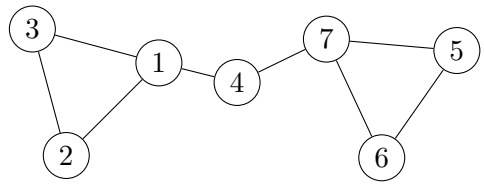


Figure 5.10: These triangles are not adjacent but not disconnected.

hypergraph
hyperedge

Fortunately, there is already a model based on graphs to represent the adjacencies of such subgraphs: hypergraphs. A *hypergraph* is a graph on which we add hyperedges. A *hyperedge* is simply a set of vertices of the graph. If we want to extend the concept of the line graph to triangles, for every triangle, we add a hyperedge containing the vertices of that triangle. This can be generalized to every transformation if we can find all its instances in order to build this hypergraph. We will refer to this method as the *hypergraph method*.

hypergraph method

hypergraph of instances

Definition 11. Given a transformation T and a graph G, we define the *hypergraph of instances* of T on G as the hypergraph having G as its underlying graph and where each hyperedge represents an instance of T on G.

graph representation

To compute the hyperedges, we could pre-compute every instance of the transformation in the graph and then compute the orbits of this hypergraph. Computing those orbits is equivalent to computing the orbits of a graph that is a copy of the underlying graph, where we added one vertex per instance of the transformation and made this vertex adjacent to each vertex of the instance. The orbits of those hyperedges are the orbits of the vertices we added. We will refer to this graph as the *graph representation* of the hypergraph. An illustration of this method can be seen in Figure 5.11. The large circles in the left graph are the hyperedges, each containing a triangle. The right graph is the equivalent graph used to compute the orbits. If we restrict ourselves to automorphisms that do not permute vertices a and b with any vertex of the original graph, their orbits will correspond to the orbits of the triangles.

The generic algorithm is given as Algorithm 3. We do not give details on the computation of the orbit set because there are already implemented algorithms to perform this operation while restricting the allowed automorphisms, such as in the Nauty software [74].

Note that a limitation of this method is that it does not differentiate between different instances having the same set of vertices. Indeed, it makes no difference between instances of an asymmetrical transformation using the same set of vertices. For example, if we consider two rotations $rot(a, b, c)$ and

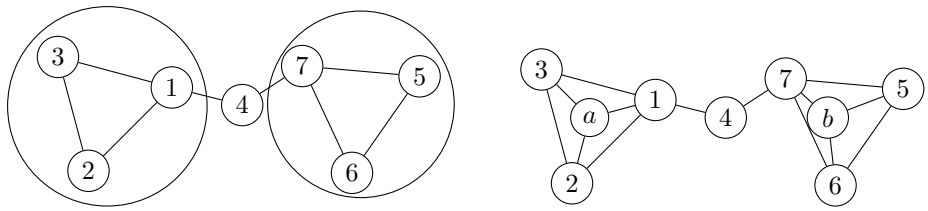


Figure 5.11: A hypergraph (left) and its representation as a graph to compute the orbits (right).

Algorithm 3: Algorithm $HyperMethod(G, B)$

Input: A graph G, the basis B of a transformation

Output: The non-symmetrical instances of the transformation

```
1  V ← V(G);
2  A ← an empty set;
3  foreach Instance I of the basis B do
4  |   Insert a vertex v in G;
5  |   Add v to the set A;
6  |   Make v adjacent to all the vertices of I in G;
7  end
8  Compute the orbit set of G when forbidding automorphisms that
     permute a vertex in A with a vertex in V;
9  foreach Vertex v in A do
10 |   output the subgraph induced by the neighbors of v in G;
11 end
```

$rot(b, c, a)$ that are not necessarily symmetrical, this *hypergraph method* would treat both rotations as the same hyperedge.

The symmetry of the transformation can be added to the graph representation of the hypergraph. Given a hyperedge, for each instance of the transformation that contains the vertices of this hyperedge, we add a copy of the induced subgraph formed by the vertices in the hyperedge and apply the transformation to this copy. Every copy v' of a vertex v is then made

adjacent to the original vertex v and to a vertex representing the instance. To differentiate this method from the original *hypergraph method*, we will refer to it as the *hypergraph method with copies*.

For example, if one wishes to remove one vertex from a path of order 3, there are three possibilities, and they are not all symmetrical. Indeed, removing an extremity or the central vertex is different. To account for this difference, we can adapt the *hypergraph method* as seen in Figure 5.12. For every application of this vertex removal, such as removing vertex c, we add a copy of the instance and apply the transformation. In the figure, we add vertices a_1 and b_1 for the application removing c. Every copy is then made adjacent to the original. We then add one more vertex per application that will be adjacent to the copies and the original vertices of the instance. For the application removing c, we add p_1 that will be adjacent to a_1, b_1, a, b and c. The orbits of those applications of the transformation are then the orbits of the vertices p_1, p_2 and p_3 if we forbid automorphisms that permute one of those vertices with the original vertices or the copies. In Figure 5.12, p_1 and p_2 would be in the same orbit but not p_3. Thus, removing a or b is symmetrical, but not removing c.

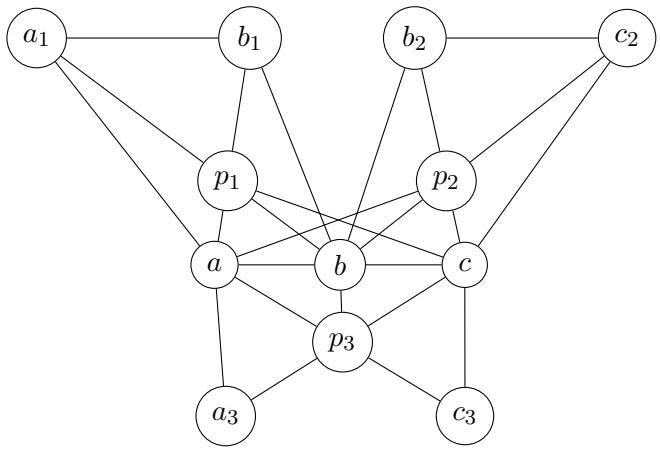


Figure 5.12: Using the *hypergraph method* with copies to remove one vertex from a P_3

The adapted algorithm is given as Algorithm 4. The only part that is modified is that instead of adding one vertex per instance, we now also add a complete copy of this instance to which we apply the transformation.

Algorithm 4: Algorithm $HyperMethodWithCopies(G, B, R)$

Input: A graph G, the basis B of a transformation and its result R

Output: The non-symmetrical instances of the transformation

```
1  V ← V(G);
2  A ← an empty set;
3  foreach Instance I of the basis B do
4  |  Insert a vertex v in G;
5  |  Add v to the set A;
6  |  Make v adjacent to all the vertices of I in G;
7  |  Add a copy I' of I in G with a vertex v' for every vertex v in I;
8  |  Apply the transformation by replacing I' by R;
9  |  Make every vertex v' of I' adjacent to its original vertex v in I;
10 end
11 Compute the orbit set of G when forbidding automorphisms that
     permute a vertex in A with a vertex in V;
12 foreach Vertex v in A do
13 |  output the subgraph induced by the neighbors of v in G;
14 end
```

This adaptation, however, further increases the size of the graph on which the orbits need to be computed. Also, both this adaptation and the basic hypergraph method require computing all the subgraphs isomorphic to the basis of the transformation, and this number can be quite high. In a complete graph of order n, there are $n(n-1)(n-2)$ triangles. We would then need to compute the orbits of the automorphism group of a graph of order $n + n(n-1)(n-2)$ (the original graph and one vertex per triangle). In some cases, and especially with large basis, the amount of possible instances of a transformation can still be quite small. Thus, the *hypergraph method* can be

useful. There is however the need to compute all those instances. We will see how it can be done in the next subsection, and we will introduce a third method that does not require a hypergraph.

5.2.3 The subgraph method

Both the *iterating method* and the *hypergraph method* require a way to find subgraphs that are isomorphic to a basis of a transformation. This problem is called the *subgraph isomorphism* problem and is known to be NP-hard [29]. However, there already exist fast algorithms to solve this problem [11, 88] (though with an exponential time complexity). One of them is the VF2 algorithm [27]. This algorithm uses a backtracking recursive method where it tries to expand a partial mapping between the graphs.

The pseudocode is shown in Algorithm 5. Let G and H be two graphs such that $|G| \geq |H|$. The goal of VF2 is to find subgraphs of G that are isomorphic to H. It starts with an empty mapping between the vertices of G and H. It will then try to map the first vertex h_1 of H to every vertex of G. For each candidate g in G, we only consider it if it is *valid*. In the VF2 algorithm, this means that the degree of g is at least equal to the degree of h_1. Then, we recursively try to map the second vertex h_2 of H to a vertex of G. But this time, the algorithm will not consider all the vertices of G. If h_1 is adjacent to h_2, it only considers the neighbors of g. If h_1 is not adjacent to h_2, it only considers of G that are not adjacent to g. The same idea is applied recursively until we either obtain a mapping that contains all the vertices of H or until we run out of possibilities.[3]

But using the orbits of H, we can adapt this algorithm to produce all the instances of a transformation without generating the symmetries. A first problem in our case is that the algorithm will generate all the permutations for each subgraph. For example, for every triangle (a, b, c), VF2 will also generate $(a, c, b), (b, a, c), (b, c, a), (c, a, b)$ and (c, b, a).

[3] In the algorithm, $|m|$ is the number of mappings in m. We are trying to map the vertex of H with the lowest number that is not already mapped: $h_0, h_1, h_2, \ldots$.

Algorithm 5: Algorithm $Match(G, H, m)$ (also called VF2)

Input: The supergraph G, the subgraph H and a partial mapping m from the vertices of H to the vertices of G.

Output: The subgraphs of G isomorphic to H

```
1  if m covers all the vertices of H then
2      output m(G)
3  else
4      Compute the pairs of vertices P that can be mapped onto vertex
        |m| of H
5      foreach (g, h) ∈ P do
6          if (g, h) is a valid pair then
7              Compute m', the mapping augmented by adding the pair
                (g, h)
8              call Match(G, H, m')
9          end
10     end
11 end
```

This problem can be solved by adapting the conditions defining which pairs are valid. Let h_i (g_i) be the ith vertex of H (G). Because of the order in which the vertices of H and G are explored, a mapping containing the pair (h_i, g_k) will be generated before a mapping containing a pair (h_j, g_k) with $j > i$. Also, if h_i and h_j are in the same orbit, for every mapping m containing the pair (h_j, g_k), there is a mapping containing (h_i, g_k) and automorphic to m. Thus, once we explored all mappings where h_i is mapped to g_k, we can ignore every mapping such that g_k is mapped to a vertex h_j ($j > i$) in the same orbit as h_i. This idea is, in fact, similar to the one used when iterating over the edges and imposing that for an edge (a, b), $a < b$.

It is important to note that, when exploring the mappings for some vertex h_i, we have a partial mapping for all the vertices h_l with $l < i$. This means that vertices h_l are fixed, and the allowed automorphisms in H are restricted.

We cannot ignore matchings with the pair (h_l, g_k) even if we found all the matchings with the pair (h_i, g_k) and h_l is in the same orbit as h_i. Because once h_l is mapped, it is fixed and thus, cannot be permuted with h_i until it is removed from the mapping. Also, once a vertex h_l is remapped, we have a different partial mapping and the forbidden mappings for every vertex h_i $(i > l)$ should be removed as the partial subgraph obtained by mapping all the vertices up to h_i is different.

Those changes can be implemented by using a matrix of dimensions $|G| \times |H|$. In this matrix, a value of 0 at coordinates (i, j) means that we can map h_i with g_j. A value bigger than 0 means that it is forbidden. Then, every time we are done exploring the possible subgraphs containing the pair (h_i, g_j), we set the value at every coordinates (k, j) $(k \geq i)$ to the current depth of the recursion and every value in the matrix that is bigger than the depth to 0 before exploring the possible subgraphs containing the next valid pair for h_i. Those values can be obtained in linear time if we store the pairs in a linked list in the order they are added.

But the adaptation of the VF2 algorithm does nothing to prevent generating automorphic subgraphs of G. We are actually filtering out the symmetries of H but not those of G. Fortunately, we can further adapt the VF2 algorithm. Instead of trying to map some vertex h of H to every vertex of G, we can try to only map it to one vertex per orbit of G. This is the same idea as before, and we will also need to recompute those orbits every time a vertex of G is mapped.

If one has an already implemented version of the VF2 algorithm, those two changes can be added easily by changing the way pairs of vertices are generated and by adapting the conditions to check if a pair is valid as seen in Algorithm 6.

Note that, if we are searching for subgraphs where no vertex holds a specific role, we can forbid automorphisms that permute mapped vertices with unmapped ones when computing the orbits. For example, if we have a path $\{a, b, c, d\}$ and fix the vertices b and c, the vertices a and d are not in the same

Algorithm 6: Algorithm $MatchAuto(G, H, m, F, L, i)$

Input:

- G, the supergraph, H, the subgraph
- a partial mapping m from the vertices of H to the vertices of G,
- a matrix F of dimensions $|G| \times |H|$,
- a list L and the depth of the current recursion i.

Output: The subgraphs of G isomorphic to H

```
1  if m covers all the vertices of H then
2      output m(G)
3  else
4      Compute the possible pairings P from a vertex of G onto vertex |m| of H
5      foreach (g_k, h_i) ∈ P do
6          if (g_k, h_i) is a valid pair and F[k][i] = 0 then
7              Compute m', the mapping augmented by adding the pair (g, h)
8              call MatchAuto(G, H, m', F, L, i + 1)
9              foreach Vertex h_j (i < j) in H that is in the same orbit as h_i do
10                 F[k][j] ← i
11                 Add (k, j) at the end of L
12             end
13         end
14     end
15     (k, j) ← the last element of L
16     while F[k][j] ≥ i do
17         F[k][j] ← 0
18         Remove the last element of L
19         (k, j) ← the new last element of L
20     end
21 end
```

orbit because a is adjacent to b while c is adjacent to d. If we allow automorphisms permuting vertices b and c together, then there is an automorphism where a and d are permuted together, and they are in the same orbit since b and c do not have distinct roles. This allows filtering more symmetries for

symmetrical transformations.

This *subgraph method* is then able to filter out more symmetries than the *iterating method* in the case of symmetrical transformations. But again, this requires manual configuration from the user. In the next section, we will compare all three methods to find out which one would be the better suited for which situation.

5.2.4 A comparison of the methods

Thanks to these three methods, we can generate all instances of a transformation on a given graph while ignoring many of the symmetrical instances that are just redundant. However, none of these methods is clearly better than the others. We will compare them on different aspects.

5.2.4.1 Ease of use

As we stated at the beginning of this section, we would like to avoid the need for in-depth knowledge for using TransProof. Thus, the method used should, ideally, require little configuration besides providing the transformation.

The *iterating method* can be used without specific configuration if we consider that no fixed vertex can be swapped with another one. However, this restricts its efficiency when it comes to filtering out symmetries. The same problem can arise for the *subgraph method* since it also uses the orbits of the graph when searching for vertices in the supergraph to map to a vertex of the subgraph.

The *hypergraph method* requires no specific information and can be used directly without any loss of efficiency. However, this is only true if using the version with copies allowing to differentiate between instances on the same subset of vertices. Without this change, only transformations where two instances on the same subset of vertices would produce the same result would be correctly handled.

5.2.4.2 Amount of symmetries filtered

As we saw, the *iterating method* will miss some symmetries and so will the *subgraph method* since it uses the same iteration pattern. But while some symmetries can remain, a lot are still filtered out. To illustrate, there is a total of 1 170 edges among all the graphs of order 6 but the *iterating method* produces only 572 non-symmetrical ones. Applying the same method for the rotations gives a number of 4 368 total instances while only 1 632 of those instances remain when using the *iterating method*.

Of course, because it generates all instances, the *subgraph method* will remove all the symmetries.

In order to better visualize the performances of those methods, we used all three methods to find the instances of a transformation with P_3 as a basis on all the graphs of a given order. We did not consider applying the transformation to those instances, since it should not change the number of instances. The results are plotted in Figure 5.13. We also plotted the total number of instances without filtering any symmetries except the permutations of the vertices of a path.

Because the number of non-isomorphic graphs on n vertices increases exponentially with n, the number of instances produced also increases exponentially. For the sake of clarity, the values are given in Table 5.2. Without surprise, the *hypergraph method* is the one filtering the most symmetries. It is directly followed by the *iterating method* and the difference decreases with the order of the graphs. The *subgraph method* is the least efficient.

However, a P_3 is relatively small and not all its vertices are in the same orbit. This is why we also ran the same tests for a K_5. This graph has a large automorphism group and will produce many symmetries. A transformation using this basis would be a symmetrical transformation. The results are shown in Figure 5.14 and given in Table 5.3. This time, the number of instances produced is much closer for all the methods. The inefficiency of the *iterating method* in the case of a symmetrical transformation is not visible here because the version used imposed that the indices of the vertices were strictly increasing

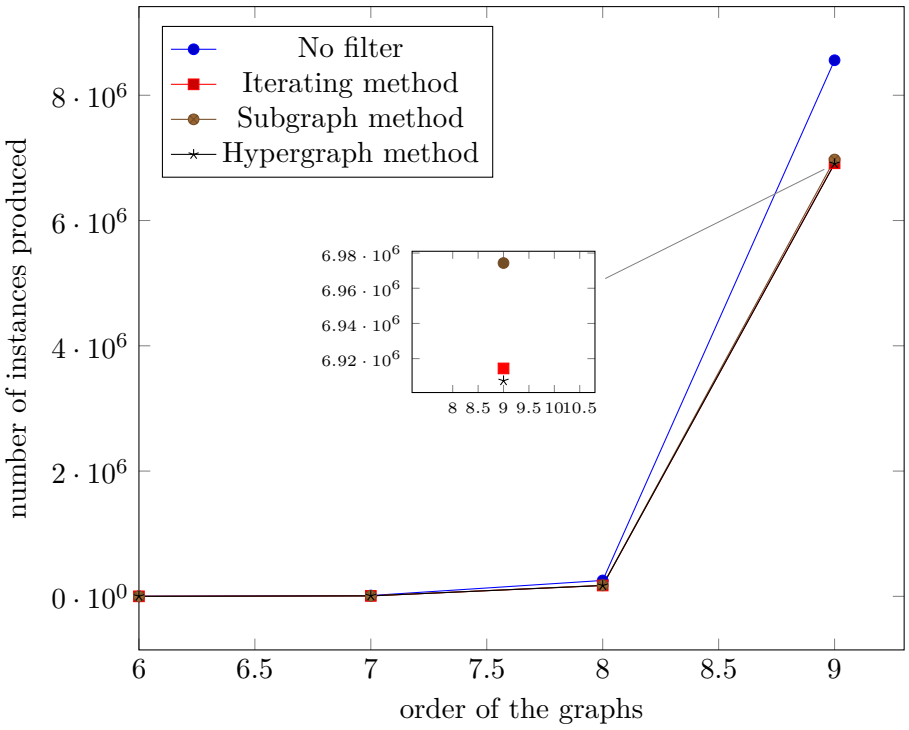


Figure 5.13: Number of instances of P_3 in all the graphs of order n produced by each method.

Table 5.2: Table of the number of instances of P_3 produced by each method on all the graphs of order n.

n	3	4	5	6	7	8	9
No filter	1	14	113	1,092	13,144	$2.54 \cdot 10^5$	$8.56 \cdot 10^6$
Iterating method	1	6	48	488	7,304	$1.73 \cdot 10^5$	$6.91 \cdot 10^6$
Subgraph method	1	5	46	488	7,461	$1.76 \cdot 10^5$	$6.97 \cdot 10^6$
Hypergraph method	1	6	48	484	7,272	$1.73 \cdot 10^5$	$6.91 \cdot 10^6$

in an instance. For example, the instance composed of the vertices $(1, 2, 3, 4, 5)$ was considered, but not $(2, 1, 3, 4, 5)$.

If one only considers the number of filtered symmetries, the *hypergraph method* would be the best choice. But as stated in Subsection 5.2.2, it is

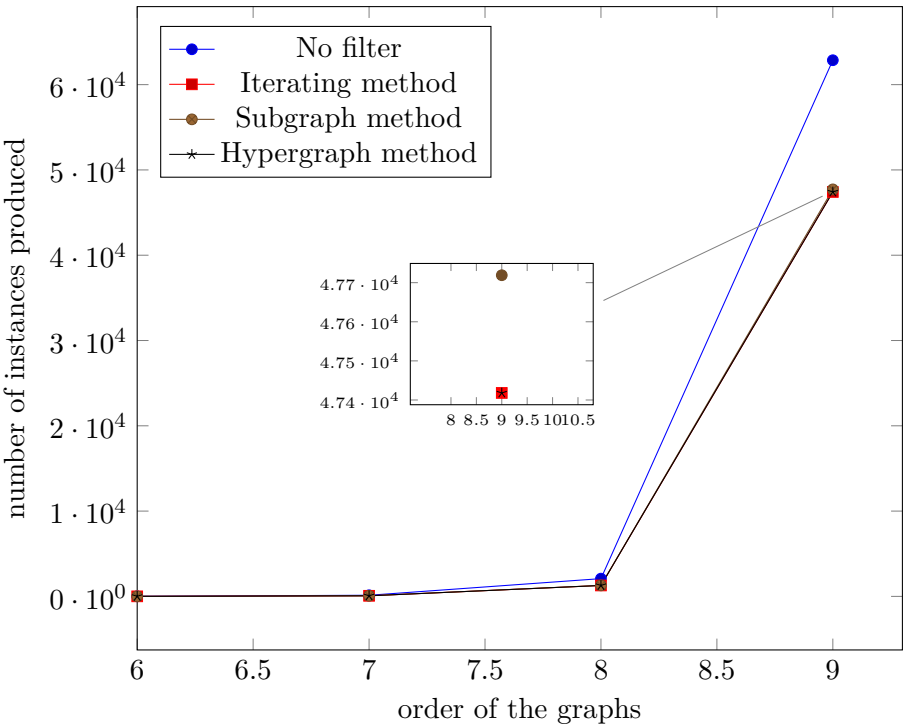


Figure 5.14: Number of instances of K_5 in all the graphs of order n produced by each method.

Table 5.3: Table of the number of instances produced of K_5 by each method on all the graphs of order n.

n	5	6	7	8	9
No filter	1	12	132	2,082	62,862
Iterating method	1	6	68	1,280	47,418
Subgraph method	1	6	68	1,288	47,719
Hypergraph method	1	6	68	1,280	47,418

limited to symmetrical transformations. The *hypergraph method* with copies solves this problem, but we need to compare its running time to that of the other methods to see if it is interesting. This is what we do next.

5.2.4.3 Expected performance

Although there is, at the time of writing, no polynomial algorithm to compute the orbits of a graph, we would like our methods to be as fast as possible in the worst-case scenarios.

Evaluating the theoretical performances of the three methods is quite complex because of the huge number of different configurations, and is out of the scope of this work. But it is known that the canonical labeling algorithm used by Nauty has an exponential time complexity [76].

The *iterating method* will have to compute the orbits of the graph once for the first vertex, then once for every orbit for this first vertex, and so on. In the worst-case scenario, if the basis has k vertices and each vertex is in its own orbit, we could compute the orbits $(k-1)!$ times, which will quickly become large. The same applies for the *subgraph method*.

The *hypergraph method* will only compute those orbits once but after having computed all the instances of the transformation which is a NP-hard problem too as it requires solving the subgraph isomorphism problem. Then, if there are l instances in a graph of order n, we will compute the orbits of a graph of order $n+l$. And, in the case of the *hypergraph method* with copies, if the instances are of order k, the resulting graph on which to compute the orbits is of order $n+k!\,l$.

Using the *hypergraph method* to produce the instances thus seems inefficient. But if l is small, one could use this method to produce the sets of vertices of those instances and then use those smaller sets to produce the instances using, for example, the *iterating method*.

To confirm those hypotheses, we measured the time taken by each of these methods to filter out the symmetrical instances of a transformation with a basis that is P_3. Those times were obtained on a machine using an AMD Ryzen 9 5900X processor with 32 GB of RAM and with the Artix Linux operating system. We plot these results in Figure 5.15. There is also a plot for the time taken to generate all the instances without any filtering. The last dataset corresponds to the *hypergraph method* with copies. Of course, adding copies will

not affect the number of symmetries produced, but this allows comparing the *hypergraph method* with methods that can handle asymmetrical transformations. The instances for the *hypergraph method* were obtained thanks to the VF2 algorithm with the modification to avoid permutations of vertices of instances and for the *hypergraph method* with copies, the original VF2 algorithm was used. Again, for the sake of clarity, the data is available in Table 5.4.

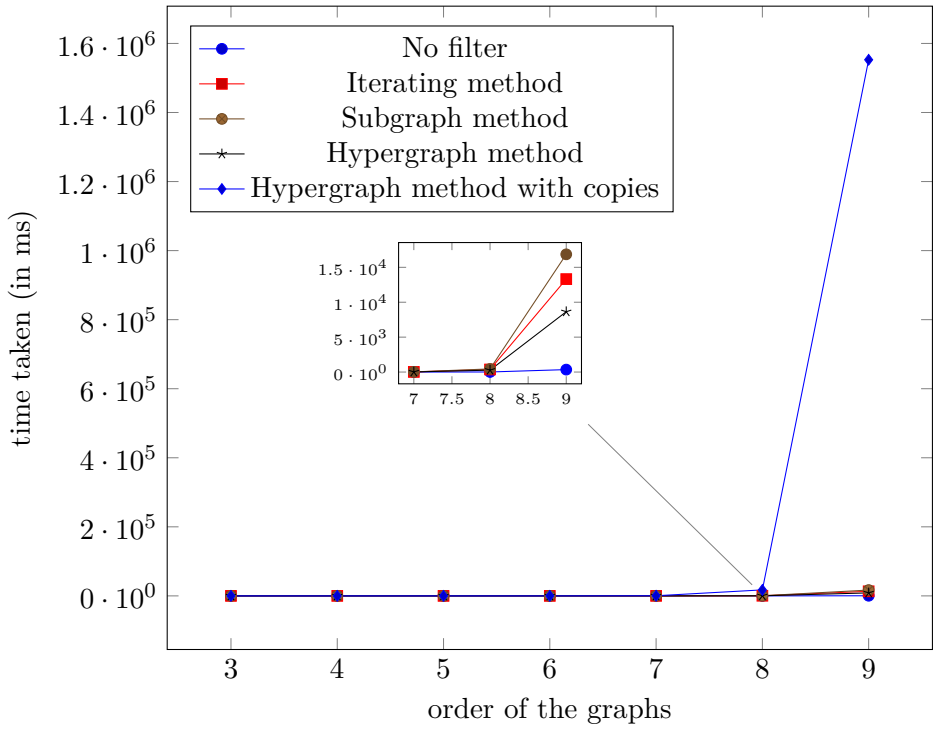


Figure 5.15: Time taken by each method to compute all the instances of P_3 in all the graphs of order n.

The first thing we see is that adding copies of instances in the case of the *hypergraph method* is really inefficient. However, the version without copies is quite efficient. The *iterating method* is the fastest method supporting asymmetrical transformations, and it is directly followed by the *subgraph method*.

Again, we also ran the same tests using a K_5 as the basis. The results are plotted in Figure 5.16 and given in Table 5.5. This time, the *hypergraph method*

Table 5.4: Table of times (in ms) taken by each method to produce the instances of P_3 on all the graphs of order n.

n	3	4	5	6	7	8	9
No filter	0	0	0	0	0	10	349
Iterating method	0	0	0	1	18	370	13,313
Subgraph method	0	0	0	1	21	449	16,861
Hypergraph method	0	0	0	1	15	272	8,640
Hypergraph method with copies	0	0	0	16	386	17,353	$1.55 \cdot 10^6$

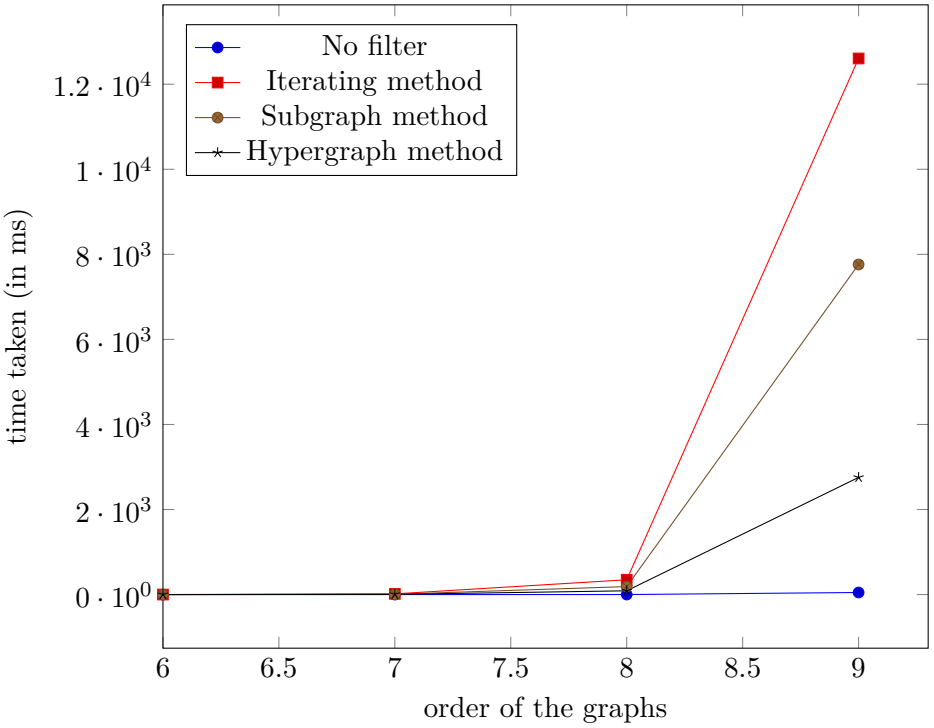


Figure 5.16: Time taken by each method to compute all the instances of K_5 in all the graphs of order n.

with copies was not used because of the factorial increase in the computational time when the order of the basis increases. The normal *hypergraph method* still remains the fastest, but the *subgraph method* is now faster than the *iterating*

method. The *subgraph method* indeed makes use of the symmetry of the graph to which the transformation needs to be applied, but also of the symmetry of the basis it is looking for. It can then consider fewer instances than a naïve *iterating method*.

Table 5.5: Table of times (in ms) taken by each method to produce the instances of K_5 on all the graphs of order n.

n	5	6	7	8	9
No filter	0	0	0	1	49
Iterating method	0	1	17	348	12,605
Subgraph method	0	0	8	188	7,761
Hypergraph method	0	0	5	89	2,755

Comparing the speeds, if one wants support of complex transformations, the *iterating method* would seem to be the most interesting one. However, this method has the disadvantage that it requires more work to be used. Indeed, the order in which the vertices of the basis are explored and the way the orbits are used can change the number of instances produced, as we saw earlier.

5.2.5 What is used in TransProof

As we saw before, all the methods reduce the number of instances produced with different efficiencies. But this work also increases the computation time by a non-negligible amount. While being the most efficient in terms of symmetry filtering, the *hypergraph method* with copies is also the slowest.

For now, TransProof uses the *iterating method* for its already implemented transformations. This choice was made because the basis of these transformations is small (at most 4 vertices), they all are asymmetrical and since they were directly implemented in TransProof, it was not a problem to require more customization. To make defining such transformations easier, a specific language was defined using a Rust [69] macro that generates Rust code when compiling TransProof. The macro, however, is limited as it does not work for

a transformation using an arbitrary number of vertices.

The *iterating method* is not the most efficient in general, but it is the fastest for small basis provided the user knows how to use it. Therefore, it is well suited for small general transformations. The adapted VF2 algorithm is implemented and available, but not yet used in practice. We will see in Section 5.4 that this algorithm could be used for custom transformations.

But even though we can filter out symmetries, a general consideration is that, for all these methods, the number of results can remain quite large. The next section will explore some considerations when storing the data produced in order to provide support for complex queries while limiting the size of the data and the answering time for those queries.

5.3 Storing the metagraph

Once the transformations have been computed, TransProof stores them in a database to be able to use the powerful query languages in order to explore the metagraph. This leads to different technical questions. The first subsection presents a format that can be used to store transformations in a compact fashion while keeping all the necessary information to later explore them. Subsection 5.3.2 compares two types of databases and the advantages to using them.

5.3.1 A format for the transformations

When storing an arc of the metagraph, we need to store three elements: the original graph G, an application G' of the transformation and the changes from G to G'. The first two can be stored easily using, for example, a binary format such as graph6 [72]. But, because we do not differentiate between isomorphic graphs, we need to store G and G' using their canonical labelling which is a canonical ordering of the vertices.

This canonical ordering makes it difficult to know which vertex v' of G' corresponds to a vertex v in G. For example, if we remove all edges incident

with the first vertex of G, this vertex might be the third vertex of G' in its canonical ordering. Therefore, in addition to the graphs G and G', we need to store the information about which vertex of G corresponds to which vertex of G'.

In TransProof, this is done using a labelled graph. Let $G = (V, E)$ be the original graph and $G' = (V', E')$ be an application of a transformation on G before applying the canonical ordering, that is, the ordering of the vertices of G' is the same as that of G. The *graph of changes* between G and G' is the complete graph H with vertex set $V \cup V'$. For every edge and vertex of H, we add a label composed of two Boolean values b_m and b_p. The Boolean b_m is true if the edge or vertex has been added or removed when transforming G into G' and false otherwise. The Boolean b_p is true if the edge or vertex is present in G' and false otherwise.

graph of changes

For example, in Figure 5.17, the graph G on the left is transformed into the middle graph G' by removing the vertex a and adding the edge (b, c). The graph of changes on the right is the complete graph with vertex set $\{a, b, c\}$. For every vertex and edge of the graph of changes, we added the two Boolean values b_m and b_p in this order (T for true and F for false). We can see that the vertex a was removed when transforming G into G'. It was thus modified and is not present in G'. Its label is then T, F. The vertex b, on the other hand, was not modified by the transformation and is still a vertex of G'. Its label is then F, T. The edge (a, b) was not modified, but is not an edge of neither G nor G'. Its label is F, F.

Those labels give us a lot of information. Of course, since b_p indicates if an element is part of G', obtaining G' is easy and because we compute the graph of changes using the same vertex order as G, we do not have to consider the problem of the canonical ordering. Obtaining the graph G is simply keeping the non-modified elements of G' (where b_m is false) and the elements not in G' (b_p is true) that were modified (b_m is true). If we wish to know which vertices or edges were added or removed, we simply keep only the elements where b_m is true.

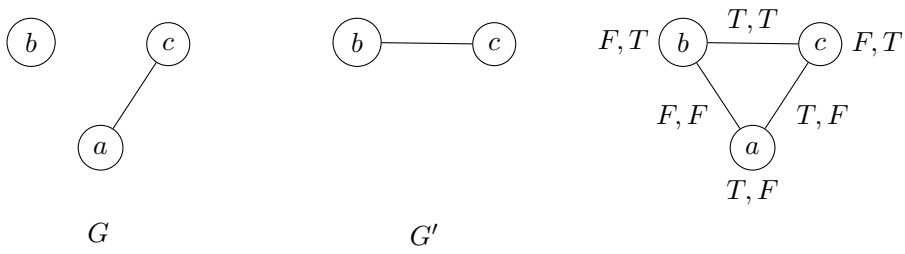


Figure 5.17: The graph of changes between the original graph G and the application G'.

This format is equivalent to using two Booleans b_G and b'_G to know if an element was present in G and in G' respectively. In both formats, using those Booleans gives us a labelled graph on which graph algorithms can be used to study those transformations. For example, it is possible to compute the canonical ordering of a graph of changes.

In practice, those two Booleans are stored as a binary number on 2 bits in order to have a compact format. This, however, does not reduce the number of arcs in the metagraph, and they still need to be stored. The next subsection will discuss about different types of databases for this purpose.

5.3.2 A database to store the metagraph

We study here different existing database models to store the metagraph. Ideally, the model used should be able to handle large amounts of data as well as offer a good support for queries.

The most common model is the relational database, which stores data in tables. Each table has a fixed number of named columns, and the data correspond to the rows of this table. A simple structure for a metagraph would be to have one table per transformation with three columns: the graph to which the transformation is applied, the resulting graph and the changes between the two. Those elements can be compacted using a binary format such as graph6.

Figure 5.18 shows how graphs and invariants are stored in PHOEG's

database in the two tables on the left. The left table stores the graph6 format of the graphs using their canonical ordering. The middle table stores the value of the eccentric connectivity index of these graphs, which will be explained in Chapter 6. The table on the right is how an application of a transformation is stored in a table in TransProof. The first column stores the graph of changes of the application using a special binary format based on graph6. The two other columns use the graph6 format of the original graph and of the graph obtained after applying the transformation.

Graphs
signature
A_
A?
B?
BG
Bw
BW
C‘
C^
C~
C?
C@

ECI	
signature	val
A_	2
Bw	6
BW	6
C^	14
C~	12
CF	9
CN	13
Cr	16
CR	14
D‘[	25
D‘{	20

remove-edge		
signature	orig	dest
AmQ=	A_	A?
AOSQ	BG	B?
AOZQ	BW	BG
A2VQ	Bw	BW
BEQQkA==	C@	C?
BEQSUA==	CB	C@
BEQZUA==	CF	CB
BGQQUA==	C‘	C@
BEYRUA==	CR	CB
BEURkA==	CR	C‘
BESRUA==	CJ	CB

Figure 5.18: Tables in a relational database for TransProof and PHOEG.

With this table, it becomes possible to write SQL queries about the metagraph. A second table storing the invariants values for each graph is enough to know if every arc is an improving arc, or simply if the conjectured extremal graphs are indeed extremal within the data.

For example, Figure 5.19 is an SQL query to compute all the arcs of the metagraph, built from edge-removal transformation, where removing an edge will increase the eccentric connectivity index.

```
SELECT t.orig, eci1.val, t.dest, eci2.val
FROM eci eci1
JOIN remove_edge t ON (eci1.signature = t.orig)
JOIN eci eci2 ON (eci2.signature = t.dest)
where eci1.val < eci2.val;
```

Figure 5.19: An SQL query to obtain the cases where removing an edge increases the eccentric connectivity index.

But we can write more complex queries such as the effect of chaining two transformations instead of just one or finding a path in the metagraph between two graphs.

This latest example is possible using recursive joins, that is, joining a table with itself recursively until some condition is achieved.

Instead of using tables, an alternative is to use graph databases. A graph database uses a graph as underlying structure and allows the user to run queries directly on the graph itself. Such model is then much closer to the principle of the metagraph. Furthermore, it makes queries requiring graph exploration such as finding paths between two vertices faster than tables, as the edges are already known.

A downside to graph databases is that, while they outperform SQL when querying about paths for specific vertices, most of them are not written with broader queries in mind, as Cheng *et al.* showed in their benchmarks [25]. So, a query about a large number of elements such as knowing if all arcs in the metagraph strictly increase an invariant will be slower than a relational database.

Given the advantages and disadvantages of the two database models, a combination of both would be best performance-wise. For example, we could have two databases containing the same data, with one using a relational model and the other using a graph model. Storage-wise, however, we would double the size of the data.

Since the main difficulty in writing a proof by transformation is to find problematic cases among many graphs, broad queries are more frequent and queries about specific graphs or even paths are rare. Thus, a relational database is, at the time of writing, better suited to TransProof. It uses the same relational database as PHOEG in order to easily access the invariants values. But the field of graph databases is evolving quickly and is likely to become a better alternative once it is mature enough.

At the time of writing, the database contains several simple transformations such as removing, adding or rotating an edge for all graphs up to order 10. This limitation is imposed by the size of the data, which remain large despite removing symmetrical transformations. TransProof can handle bigger graphs if necessary.

5.4 Efficient custom transformations

To help research, several common transformations are implemented in TransProof and they are pre-computed for all the graphs up to order 10. This makes trying to build a proof by transformation using those faster. But as we saw in Chapter 4, more complex transformations are sometimes needed. Therefore, researchers need to be able to provide their own transformation and, ideally, computing a metagraph should remain fast with custom transformations.

5.4.1 Simple transformations

One simple way to provide support for custom transformations is to allow combination of simple transformations. Any transformation can be described in terms of adding or removing edges or vertices. Thus, it is possible to define a set of basic transformations that will serve as a basis to produce more complex ones.

The choice of this basis of transformations requires being able to generate all transformations from a subset of the simple ones. As an example, one

can consider increasing the number of connected components of a complete graph. This requires removing a number of edges depending on the graph the transformation is applied to.

This means that we need a way to specify how a variable number of edges will be added or removed. We should thus add different transformations based on the ways one can connect or disconnect a subset of vertices of a graph.

With this basis of transformations, we need only to compute simple transformations and store them inside the database. This data can then be exploited by queries to the database for more complex transformations, and we do not need to recompute those.

However, it is possible to generate a large number of duplicated results. Indeed, simply removing k edges produces $k!$ possible orders in which to remove them. Furthermore, the queries are limited to what has already been computed and stored in the database.

5.4.2 Complex transformations

A different approach is to use hypergraphs to define transformations and the *subgraph method* to compute the instances. A transformation could be defined by the user as the basis (using hyperedges for specific subgraphs) and the result. Using the subgraph method and the adapted VF2 algorithm, it becomes possible to generate the instances of such transformations while filtering symmetries. Also, this hypergraph can be defined in a visual way, providing user-friendliness.

This hypergraph provides more flexibility than the simple method explained in the previous subsection and does not require pre-computed data. But this flexibility comes with a cost. Indeed, storing all the custom transformations in a database can quickly fill the available storing space. Therefore, transformations defined this way should not be stored permanently.

5.4.3 A hybrid method

As we can see, both versions have their advantages and downsides. But they also intervene at different points in the process of writing a proof by transformation. At first, when little is known about the impact of changes on some invariant, simple transformations are preferred and many graphs have to be considered. Those simple transformations can be quite generic and pre-computed.

When the subject becomes better understood, the number of graphs for which extremality cannot be refuted becomes smaller and transformations used for those become more complex. At this point, since there are fewer graphs, computing the metagraph will usually be faster and pre-computed values are not necessary, but the need arise to have more flexibility in defining transformations. For example, in their paper, Hansen *et al.* [51] use transformations with little requirements in the beginning of their proof and the transformations become more complex and more constrained as the proof unfolds.

In TransProof, one can use the already computed and implemented transformations to prune the data and then use custom transformations to try solving the remaining problems. At the time of writing, the hypergraph model to define transformations is being implemented, but it is not yet functional. It is possible, however, to use custom transformations by writing them in TransProof thanks to the specific graph library that was developed.

5.5 Conclusion

We presented ideas and methods that can be used to compute the applications of a transformation on a large amount of small graphs. These ideas are currently being used in TransProof or in the process of being added. Currently, TransProof uses the *iterating method* to compute those applications and stores them in a relational database. It contains the applications of several simple transformations such as edge-removal, rotation and vertex-removal for all graphs up to 10 vertices. But TransProof can compute other transfor-

mations and store them temporarily in its database.

The main difficulty is not so much the computation time for one graph, but rather the large number of graphs, even if restricted to a small order, and the exponential number of applications for each graph. Despite this, querying the metagraph can prove quite helpful, as we will see in the next two chapters which present results on two graph invariants.

Chapter 6

On the eccentric connectivity index

During the development of TransProof, in order to assess its usefulness and study the needed functionalities, we worked on different problems in extremal graph theory. The first one is about the eccentric connectivity index of a graph. Two papers were published about bounds on this invariant [32, 55] but only one uses proofs by transformation. This paper [32] is presented in this chapter with minor changes to better fit this document and to better show how TransProof was used.

6.1 Introduction

A chemical graph is a representation of the structural formula of a chemical compound in terms of graph theory where atoms are represented by vertices and chemical bonds by edges. Arthur Cayley [24] was probably the first to publish results that consider chemical graphs. In an attempt to analyze the chemical properties of alkanes, Wiener [91] introduced the *path number index*, nowadays called *Wiener index*, which is defined as the sum of the lengths of the shortest paths between all pairs of vertices. Mathematical properties and chemical applications of this distance-based index have been widely researched.

Numerous other topological indices are used to help describe and understand the structure of molecules [67, 86] by means of studies on quantitative structure-property relationship (QSPR) and quantitative structure-activity relationship (QSAR). Among these indices, the *eccentric connectivity index* can be defined as follows. Let $G = (V, E)$ be a simple connected undirected graph.

eccentric connectivity index

The *eccentric connectivity index* $\xi^c(G)$ of G is defined by

$$\xi^c(G) = \sum_{v \in V} deg_G(v) e_G(v).$$

This index was introduced by Sharma *et al.* [84] and successfully used for mathematical models of biological activities of diverse nature [41,49,62,70,83]. It was also studied for specific classes of graphs [38,54,94] and several extremal results were published [77, 78, 95]. Among them, Hauweele *et al.* [55] have characterized those graphs which have the largest eccentric connectivity index among all connected graphs of a given order n. These results are summarized in Table 6.1, where

- K_n is the complete graph of order n;
- P_n is the path of order n;
- W_n is the wheel of order n, i.e., the graph obtained by joining a vertex to all vertices of a cycle of order $n-1$;
- M_n is the graph obtained from K_n by removing a maximum matching and, if n is odd, an additional edge adjacent to the unique vertex that still has degree $n-1$;
- $\mathsf{E}_{n,D}$ is the graph constructed from a path $u_0 - u_1 - \ldots - u_D$ by joining each vertex of a clique K_{n-D-1} to u_0, u_1 and u_2.

In addition to the above-mentioned graphs, we will also consider the following ones:

- C_n is the chordless cycle of order n;
- $\mathsf{S}_{n,x}$ is the graph of order n obtained by linking all vertices of a stable set of $n-x$ vertices with all vertices of a clique K_x. The graph $\mathsf{S}_{n,1}$ is called a *star*.

Table 6.1: Largest eccentric connectivity index for a fixed order n

n	optimal graphs
1	K_1
2	K_2
3	K_3 and P_3
4	M_4
5	M_5 and W_5
6	M_6
7	M_7
8	M_8 and $\mathsf{E}_{8,4}$
≥ 9	$\mathsf{E}_{n,\lceil \frac{n+1}{3} \rceil + 1}$

Also, for $n \geq 4$ and $p \leq n-3$, let $\mathsf{H}_{n,p}$ be the graph of order n obtained by adding a dominating vertex (i.e., a vertex linked to all other vertices) to the graph of order $n-1$ having p vertices of degree 0, and

- $n-1-p$ vertices of degree 1 if $n-p$ is odd;
- $n-2-p$ vertices of degree 1 and one vertex of degree 2 if $n-p$ is even.

For illustration, $\mathsf{H}_{8,3}$ and $\mathsf{H}_{9,3}$ are drawn on Figure 6.1. Note that $\mathsf{H}_{4,0} \simeq \mathsf{S}_{4,2}$. Moreover, $\mathsf{H}_{4,0}$ has two dominating vertices while $\mathsf{H}_{4,1}$ and $\mathsf{H}_{n,p}$ have exactly one dominating vertex for all $n \geq 5$ and $p \leq n-3$.

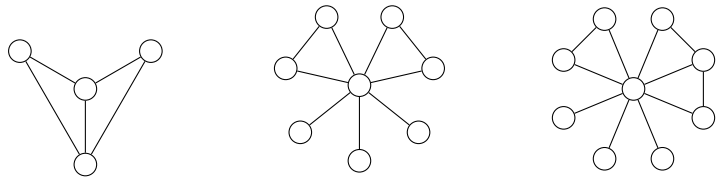


Figure 6.1: Three examples of $\mathsf{H}_{n,p}$.

In this chapter, we first give an alternative proof to a result of Zhou and Du [96] showing that the stars are the only graphs with the smallest eccentric

connectivity index among all connected graphs of a given order $n \geq 4$. These graphs have $n-1$ pendant vertices (i.e., vertices of degree 1). We then consider all pairs (n, p) of integers with $p \leq n-1$ and characterize the graphs with the smallest eccentric connectivity index among all connected graphs of order n with p pendant vertices. This characterization is proven using a proof by transformation. A similar study appears in [52] where bounds on the Randić index are given for graphs with fixed order and fixed number of pendant vertices. Note that pendant vertices play a special role in chemical compounds known as *alkanes*, which are exclusively composed of carbon and hydrogen atoms. Each carbon atom has four chemical bonds and each hydrogen atom has one chemical bond. Hence, the number of pendant vertices in the chemical graph that represents an alkane (where every vertex corresponds to an atom, and every edge to a chemical bond) is equal to the number of hydrogen atoms in the considered chemical compound.

6.2 Minimizing ξ^c for graphs with fixed order

K_1 and K_2 are the only connected graphs with 1 and 2 vertices, respectively, while K_3 and P_3 are the only connected graphs with 3 vertices. Since $\xi^c(\mathsf{K}_3) = \xi^c(\mathsf{P}_3) = 6$, all connected graphs with the same order have the same eccentric connectivity index when $n \leq 3$. From now on, we therefore only consider connected graphs with fixed order $n \geq 4$. A proof of the following theorem was already given by Zhou and Du in [96]. Ours is slightly different.

Theorem 2. *Let G be a connected graph of order $n \geq 4$. Then $\xi^c(G) \geq 3(n-1)$, with equality if and only if $G \simeq \mathsf{S}_{n,1}$.*

Proof. Let x be the number of dominating vertices (i.e., vertices of degree $n-1$) in G. We distinguish three cases.

- If $x = 1$, then let u be the dominating vertex in G. Clearly, $e_G(u) = 1$ and $deg_G(u) = n-1$. All vertices $v \neq u$ have eccentricity $e_G(v) = 2$, while their degree is at least 1 (since G is connected). Hence, $\xi^c(G) \geq$

$(n-1)+2(n-1)=3(n-1)$, with equality if and only if all $v \neq u$ have degree 1, i.e., $G \simeq \mathsf{S}_{n,1}$.

- If $x > 1$, then all dominating vertices u have $deg_G(u)e_G(u) = n-1$, while all non-dominating vertices v have $deg_G(v) \geq x \geq 2$ and $e_G(v) \geq 2$, which implies $deg_G(v)e_G(v) \geq 4$. If $n = 4$, all the dominating vertices have degree 3 and eccentricity 1 and we therefore have $\xi^c(G) \geq 3n > 3(n-1)$ which is reached when $x = 4$. If $n > 4$, we have $\xi^c(G) \geq 2(n-1)+4(n-2) = 6n-10 > 3(n-1)$.
- If $x = 0$, then every pendant vertex v has $e_G(v) \geq 3$ since its only neighbor is a non-dominating vertex. Since the eccentricity of the non-pendant vertices is at least two, we have $deg_G(v)e_G(v) \geq 3$ for all vertices v in G, which implies $\xi^c(G) \geq 3n > 3(n-1)$.

□

Stars have $n-1$ pendant vertices. As will be shown in the next subsection, obtaining a similar result is more challenging when the total number of pendant vertices is fixed to a value strictly smaller than $n-2$.

6.3 Minimizing ξ^c for graphs with fixed order and fixed number of pendant vertices

Let G be a connected graph of order $n \geq 4$ with p pendant vertices. Clearly, $p \leq n-1$, and $G \simeq \mathsf{S}_{n,1}$ if $p = n-1$. For $p = n-2$, let u and v be the two non-pendant vertices. Note that u is adjacent to v since G is connected. Clearly, G is obtained by linking $x \leq n-3$ vertices of a stable set S of $n-2$ vertices to u, and the $n-2-x$ other vertices of S to v. The $n-2$ pendant vertices w have $deg_G(w) = 1$ and $e_G(w) = 3$, while $e_G(u) = e_G(v) = 2$ and $deg_G(u) + deg_G(v) = n$. Hence, $\xi^c(G) = 3(n-2)+2n = 5n-6$ for all graphs of order n with $n-2$ pendant vertices.

The above observations show that all graphs of order n with a fixed number p of pendant vertices have the same eccentric connectivity index when $p \geq$

$n-2$. As will be shown, this is not the case when $n \geq 4$ and $p \leq n-3$. We will prove that $\mathsf{H}_{n,p}$ is almost always the unique graph minimizing the eccentric connectivity index. We first give results that can be proven mathematically when the graphs contain at least one dominating vertex. The general result when there is no dominating vertex is then proven by transformation. Note that

$$\xi^c(\mathsf{H}_{n,p}) = \begin{cases} n-1+2p+4(n-p-1) = 5n-2p-5 & \text{if } n-p \text{ is odd} \\ n-1+2p+4(n-p-2)+6 = 5n-2p-3 & \text{if } n-p \text{ is even.} \end{cases}$$

Theorem 3. *Let G be a graph of order $n \geq 4$ with $p \leq n-3$ pendant vertices and one dominating vertex. Then $\xi^c(G) \geq \xi^c(\mathsf{H}_{n,p})$, with equality if and only if $G \simeq \mathsf{H}_{n,p}$.*

Proof. The dominating vertex u in G has $deg_G(u)e_G(u) = n-1$, the pendant vertices v have $deg_G(v)e_G(v) = 2$, and the other vertices w have $e_G(w) = 2$ and $deg_G(w) \geq 2$. Hence, $\xi^c(G)$ is minimized if all non-pendant and non-dominating vertices have degree 2, except for one vertex with degree 3 if $n-p-1$ is odd. In other words, $\xi^c(G)$ is minimized if and only if $G \simeq \mathsf{H}_{n,p}$. □

Theorem 4. *Let G be a connected graph of order $n \geq 4$, with at least two dominating vertices.*

- *If $n = 4$ then $\xi^c(G) \geq 12$, with equality if and only if $G \simeq \mathsf{K}_4$.*
- *If $n = 5$ then $\xi^c(G) \geq 20$, with equality if and only if $G \simeq \mathsf{S}_{5,2}$ or $G \simeq \mathsf{K}_5$.*
- *If $n \geq 6$ then $\xi^c(G) \geq 6n-10$, with equality if and only if $G \simeq \mathsf{S}_{n,2}$.*

Proof. Let x be the number of dominating vertices in G. Then $deg_G(u)e_G(u) = n-1$ for all dominating vertices u, while $e_G(v) = 2$ and $deg_G(v) \geq x$ for all other vertices v. Hence, $\xi^c(G) \geq x(n-1) + 2x(n-x) = -2x^2 + x(3n-1)$.

- If $n = 4$ then $\xi^c(G) \geq f(x) = -2x^2 + 11x$. Since $2 \leq x \leq 4$, $f(2) = 14, f(3) = 15$, and $f(4) = 12$, we conclude that $\xi^c(G) \geq 12$, with equality if and only if $x = 4$, which is the case when $G \simeq \mathsf{K}_4$.
- If $n = 5$ then $\xi^c(G) \geq f(x) = -2x^2 + 14x$. Since $2 \leq x \leq 5$, $f(2) = f(5) = 20$ and $f(3) = f(4) = 24$, we conclude that $\xi^c(G) \geq 20$, with equality if and only if $x = 2$ or 5, which is the case when $G \simeq \mathsf{S}_{5,2}$ or $G \simeq \mathsf{K}_5$.
- If $n \geq 6$ then $-2x^2 + x(3n-1)$ is minimized for $x = 2$, which is the case when $G \simeq \mathsf{S}_{n,2}$.

□

Theorem 5. *Let G be a connected graph of order $n \geq 4$, with $p \leq n-3$ pendant vertices and no dominating vertex. Then $\xi^c(G) > \xi^c(\mathsf{H}_{n,p})$ unless $n = 5$, $p = 0$ and $G \simeq \mathsf{C}_5$, in which case $\xi^c(G) = \xi^c(\mathsf{H}_{n,0}) = 20$.*

Proof. Let U be the subset of vertices u in G such that $deg_G(u) = e_G(u) = 2$. If U is empty, then all non-pendant vertices v in G have $deg_G(v) \geq 2$ and $e_G(v) \geq 2$ (since G has no dominating vertex), and at least one of these two inequalities is strict, which implies $deg_G(u)e_G(u) \geq 6$. Also, every pendant vertex w has $e_G(w) \geq 3$ since their only neighbor is not dominant. Hence, $\xi^c(G) \geq 6(n-p)+3p = 6n-3p$. Since $p \leq n-3$, we have $\xi^c(G) \geq 5n-2p+3 > \xi^c(\mathsf{H}_{n,p})$.

So, assume $U \neq \emptyset$. Let u be a vertex in U, and let v, w be its two neighbors. Also, let $A = N(v) \setminus (N(w) \cup \{w\})$, $B = (N(v) \cup N(w)) \setminus \{u\}$, and $C = N(w) \setminus (N(v) \cup \{v\})$. Since $e_G(u) = 2$, all vertices of G belong to $A \cup B \cup C \cup \{u, v, w\}$. We finally define B' as the subset of B that contains all vertices b of B with $deg_G(b) = 2$ (i.e., their only neighbors are v and w). This decomposition is shown in Figure 6.2. The vertices A, B, C and B' are sets of vertices. Not all the edges are shown here. Vertices v and w could be adjacent, and there could be edges between vertices of A and vertices of C or between vertices of $B \setminus B'$ and vertices of A or C.

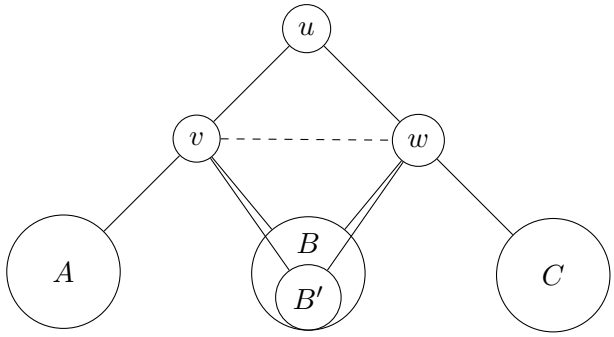


Figure 6.2: A decomposition of G into sets of vertices.

To show that such a graph cannot be extremal, we will use four transformations defined below.

Transformation 1:
If v is adjacent to w, $A \neq \emptyset$ otherwise w is a dominating vertex, and $C \neq \emptyset$ otherwise v is dominating. In this case, it is easy to obtain a new graph with the same number of pendant vertices but with a dominant vertex. Let G' be the graph obtained from G by replacing every edge linking v to a vertex $a \in A$ with an edge linking w to a, and by removing all edges linking v to a vertex of $B \setminus B'$. The resulting graph G' is shown in Figure 6.3. Clearly, G' is also a connected graph of order n with p pendant vertices, and w is the only dominating vertex in G'. It follows from Theorem 3 that $\xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$. Also,

- $deg_G(u) = deg_{G'}(u)$ and $e_G(u) = e_{G'}(u)$;
- $deg_G(x) = deg_{G'}(x)$ and $e_G(x) \geq e_{G'}(x)$ for all $x \in A \cup C$;
- $deg_G(x) = deg_{G'}(x)$ and $e_G(x) = e_{G'}(x)$ for all $x \in B'$;
- $deg_G(x) > deg_{G'}(x)$ and $e_G(x) = e_{G'}(x)$ for all $x \in B \setminus B'$.

Hence,

$$\sum_{x \in A \cup B \cup C \cup \{u\}} deg_G(x) e_G(x) \geq \sum_{x \in A \cup B \cup C \cup \{u\}} deg_{G'}(x) e_{G'}(x).$$

Moreover,

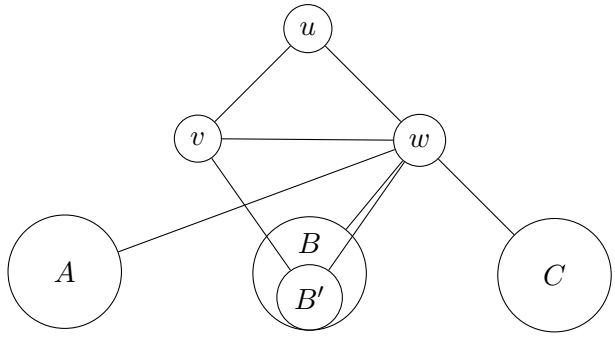


Figure 6.3: The graph obtained from transformation 1.

- $deg_G(v)e_G(v) + deg_G(w)e_G(w) = 2(|A| + |B| + 2) + 2(|C| + |B| + 2) = 2|A| + 4|B| + 2|C| + 8$;
- $deg_{G'}(v)e_{G'}(v) + deg_{G'}(w)e_{G'}(w) = 2(|B'| + 2) + |A| + |B| + |C| + 2$.

We therefore have

$$\begin{aligned}
\xi^c(G)-\xi^c(G') &= \sum_{x\in A\cup B\cup C\cup\{u\}} deg_G(x)e_G(x) \\
&\quad + deg_G(v)e_G(v)+deg_G(w)e_G(w) \\
&\quad - \sum_{x\in A\cup B\cup C\cup\{u\}} deg_{G'}(x)e_{G'}(x) \\
&\quad - (deg_{G'}(v)e_{G'}(v)+deg_{G'}(w)e_{G'}(w)) \\
&\geq (2|A|+4|B|+2|C|+8)-(2(|B'|+2)+|A|+|B|+|C|+2) \\
&= |A|+|C|+3(|B'|+|B\setminus B'|)-2|B'|+2 \\
&= |A|+|C|+|B'|+3|B\setminus B'|+2 > 0
\end{aligned}$$

This implies $\xi^c(G) > \xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$.

Transformation 2:
If v is not adjacent to w, and both $A\cup(B\setminus B')$ and $C\cup(B\setminus B')$ are nonempty, we will instead apply transformation 1 after having added the edge (u, v). Let G' be the graph obtained from G by adding an edge linking v to w, by replacing every edge linking v to a vertex $a \in A$ with an edge linking w to a,

and by removing all edges linking v to a vertex of $B \setminus B'$. Clearly, G' is also a connected graph of order n with p pendant vertices. As in the previous case, we have

$$\sum_{x \in A \cup B \cup C \cup \{u\}} deg_G(x) e_G(x) \geq \sum_{x \in A \cup B \cup C \cup \{u\}} deg_{G'}(x) e_{G'}(x).$$

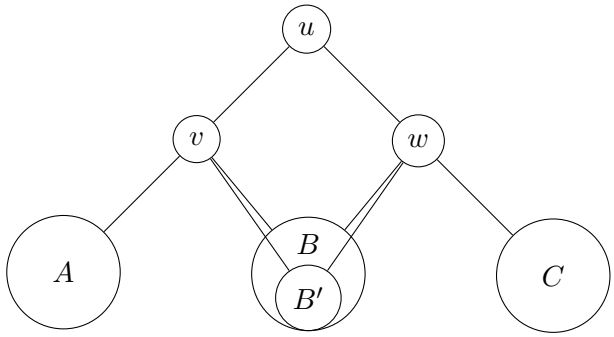


Figure 6.4: The graph before applying transformation 2.

Moreover, $e_G(v) \geq 2$ and $e_G(w) \geq 2$, while $e_{G'}(v) \leq 2$ and $e_{G'}(w) = 1$, which implies

- $deg_G(v)e_G(v) + deg_G(w)e_G(w) \geq 2(|A| + |B| + 1) + 2(|C| + |B| + 1) = 2|A| + 4|B| + 2|C| + 4$;
- $deg_{G'}(v)e_{G'}(v) + deg_{G'}(w)e_{G'}(w) \leq 2(|B'| + 2) + |A| + |B| + |C| + 2$.

We therefore have

$$\begin{aligned} \xi^c(G) - \xi^c(G') &\geq (2|A| + 4|B| + 2|C| + 4) - (2(|B'| + 2) + |A| + |B| + |C| + 2) \\ &= |A| + |C| + |B'| + 3|B \setminus B'| - 2. \end{aligned}$$

If $B \setminus B' \neq \emptyset$, w is the only dominating vertex in G', and $\xi^c(G) - \xi^c(G') > 0$. It then follows from Theorem 3 that $\xi^c(G) > \xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$. So assume $B \setminus B' = \emptyset$. Since $A \cup (B \setminus B') \neq \emptyset$, and $C \cup (B \setminus B') \neq \emptyset$, we have $A \neq \emptyset$ and $C \neq \emptyset$. Hence, once again, w is the only dominating vertex in G', and we know from Theorem 3 that $\xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$.

- If $|B'| \geq 1$, $|A| \geq 2$ or $|C| \geq 2$, then $\xi^c(G) > \xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$.
- If $|B'| = 0$ and $|A| = |C| = 1$, there are two possible cases:
  - if the vertex in A is not adjacent to the vertex in C, then $n = 5$, $p = 2$, $G \simeq \mathsf{P}_5$ and $G' \simeq \mathsf{H}_{5,2}$. Hence, $\xi^c(G) = 24 > 16 = \xi^c(\mathsf{H}_{n,p})$;
  - if the vertex in A is adjacent to the vertex in C, then $n = 5$, $p = 0$, $G \simeq \mathsf{C}_5$ and $G' \simeq \mathsf{H}_{5,2}$. Hence, $\xi^c(G) = \xi^c(\mathsf{H}_{n,p}) = 20$;

Transformation 3:
If v is not adjacent to w, and at least one of $A \cup (B \setminus B')$ and $C \cup (B \setminus B')$ is empty, TransProof allows us to see that if, B' is empty, the number of pendant vertices can be modified by transformation 2. For example, if $A \cup (B \setminus B') = \emptyset$, then v is a pendant vertex and transformation 2 makes v non-pendant as shown in Figure 6.5. Because our theorem is about graphs with a fixed number of pendant vertices, this is a problem.

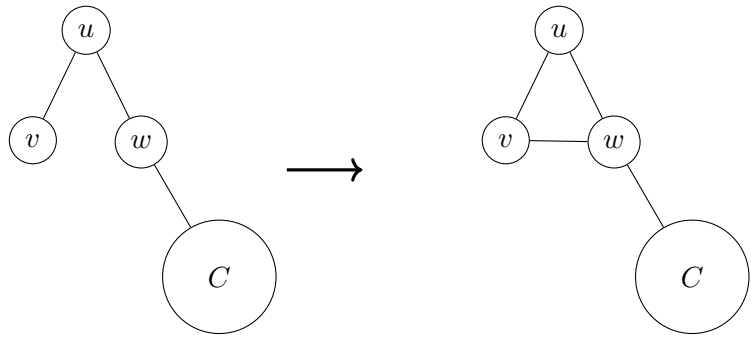


Figure 6.5: Applying transformation 2 when A and B are empty.

However, assuming that $A \cup (B \setminus B') = \emptyset$, since $n \geq 4$, we have that $C \neq \emptyset$ and also, there is a non-pendant vertex $r \in C$ because $p \leq n - 3$. Let G' be the graph obtained from G by removing the edge linking u and v and by linking v to w and to r. The result can be seen in Figure 6.6. Note that G' is a connected graph of order n with p pendant vertices : while v was pendant

in G, but not u, the situation is the opposite in G'. Note also that Theorem 3 implies $\xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$ since w is the only dominating vertex in G'. We then have:

- $deg_G(u)=2$, $deg_{G'}(u)=1$ and $e_G(u)=e_{G'}(u)=2$, giving $deg_G(u)e_G(u) - deg_{G'}(u)e_{G'}(u) = 2$;
- $deg_G(v)=1$, $deg_{G'}(v)=2$ $e_G(v)=3$ and $e_{G'}(v)=2$, giving $deg_G(v)e_G(v) - deg_{G'}(v)e_{G'}(v) = -1$;
- $deg_G(w) = n-2$, $deg_{G'}(w) = n-1$ $e_G(w) = 2$ and $e_{G'}(w) = 1$, giving $deg_G(w)e_G(w) - deg_{G'}(w)e_{G'}(w) = n-3$;
- $deg_{G'}(r)= deg_G(r)+1$, $e_G(r)=3$ and $e_{G'}(w)=2$, giving $deg_G(r)e_G(r) - deg_{G'}(r)e_{G'}(r) = deg_G(r)-2$;
- $deg_{G'}(c)=deg_G(c)$ and $e_G(c) > e_{G'}(c)$ for all $c \in (C \setminus \{r\})$. Since r has a neighbor in C of degree at least 2, we have $\sum_{c \in C \setminus \{r\}}(deg_G(c)e_G(c) - deg_{G'}(c)e_{G'}(c) \geq 2)$.

Hence, $\xi^c(G) - \xi^c(G') \geq 2 - 1 + \underbrace{n-3}_{>0} + \underbrace{deg_G(r)-2}_{\geq 0} + 2 > 0$, which implies $\xi^c(G) > \xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$.

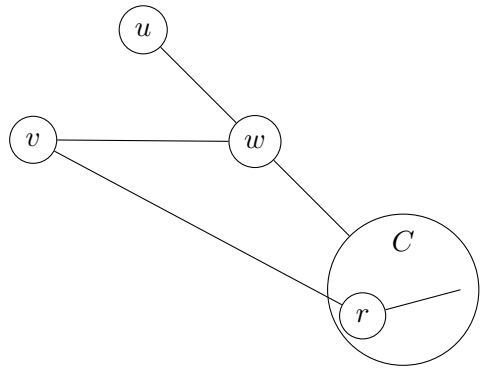


Figure 6.6: Result of transformation 3.

Transformation 4:

If v is not adjacent to w, and at least one of $A \cup (B \setminus B')$ and $C \cup (B \setminus B')$ is empty, but B' is not empty, applying transformation 2 can produce graphs with a smaller value for ξ^c. But there is another problem raised by TransProof: transformations 1 and 2 are not sufficient to reach $\mathsf{H}_{n,p}$. Indeed, if we apply transformation 2 to the graph of Figure 6.7, we do obtain a graph with a lower value for ξ^c with two dominating vertices. But transformations 1, 2 and 3 do not apply anymore, and we do not know yet if $\mathsf{H}_{n,p}$ is the extremal graph when we have two dominant vertices. Thus, we define a different transformation.

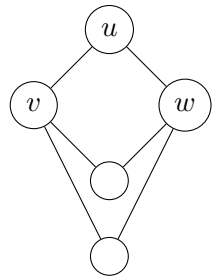


Figure 6.7: A graph from which transformations 1, 2 and 3 cannot produce $\mathsf{H}_{n,p}$

Let $b_1, \ldots, b_{|B'|}$ be the vertices in B'. Remember that the unique neighbors of these vertices are v and w. Let G' be the graph obtained from G as follows. We first add an edge linking v to w. Then, for every odd $i < |B'|$, we add an edge linking b_i to b_{i+1} and remove the edges linking v to b_i and to b_{i+1}. We then have

- $deg_G(x) = deg_{G'}(x)$ and $e_G(x) = e_{G'}(x)$ for all $x \in B' \cup C \cup \{u\}$;
- $deg_G(v) = |B'| + 1$, $deg_{G'}(v) \leq 3$, $e_G(v) \geq 2$, and $e_{G'}(v) \leq 2$;
- $deg_G(w) = |B'| + |C| + 1$, $deg_{G'}(w) = |B'| + |C| + 2$, $e_G(w) = 2$, and $e_{G'}(w) = 1$.

Hence,

$$\begin{aligned}\xi^c(G) - \xi^c(G') &= deg_G(v)e_G(v) + deg_G(w)e_G(w) \\ &\quad - deg_{G'}(v)e_{G'}(v) + deg_{G'}(w)e_{G'}(w)\end{aligned}$$

$$\begin{aligned}&\geq 2(|B'|+1)+2(|B'|+|C|+1)-6-(|B'|+|C|+2)\\&=3|B'|+|C|-4.\end{aligned}$$

IF $|B'| \geq 2$ or $|C| \geq 2$, then $\xi^c(G) - \xi^c(G') > 0$, and since w is then the only dominating vertex in G', we know from Theorem 3 that $\xi^c(G) > \xi^c(G') \geq \xi^c(\mathsf{H}_{n,p})$. So, assume $|B'| = 1$ and $|C| \leq 1$:

- if $|C| = 0$ then $n = 4$, $p = 0$, $G \simeq \mathsf{C}_4$ and $G' \simeq \mathsf{H}_{4,0}$ which implies $\xi^c(G) = 16 > 14 = \xi^c(\mathsf{H}_{n,p})$;
- if $|C| = 1$ then $n = 5$, $p = 1$, $\xi^c(G) = 23$ and $G' \simeq \mathsf{H}_{5,1}$ which implies $\xi^c(G) > 20 = \xi^c(\mathsf{H}_{n,p})$.

□

We can now combine these results as follows. Assume G is a connected graph of order n with p pendant vertices. If $p \geq 1$, then G has at most one dominating vertex, and it follows from Theorems 3 and 5 that $\mathsf{H}_{n,p}$ is the only graph with maximum eccentric connectivity index. If $p = 0$ and $n = 4$, then G cannot contain exactly one dominating vertex, and Theorems 4 and 5 show that K_4 is the only graph with maximum eccentric connectivity index. If $p = 0$ and $n = 5$, Theorems 3, 4 and 5 show that $\mathsf{H}_{5,0}$, $\mathsf{S}_{5,2}$, K_5 and C_5 are the only candidates to minimize the eccentric connectivity index, and since $\xi^c(\mathsf{H}_{5,0}) = \xi^c(\mathsf{S}_{5,2}) = \xi^c(\mathsf{K}_5) = \xi^c(\mathsf{C}_5) = 20$, the four graphs are the optimal ones. If $p = 0$ and $n \geq 6$ then we know from Theorems 3, 4 and 5 that $\mathsf{S}_{n,2}$ and $\mathsf{H}_{n,0}$ are the only candidates to minimize the eccentric connectivity index. Since $\xi^c(\mathsf{S}_{6,2}) = 26 < 27 = \xi^c(\mathsf{H}_{6,0})$, $\xi^c(\mathsf{S}_{7,2}) = 32 > 30 = \xi^c(\mathsf{H}_{7,0})$ and $\xi^c(\mathsf{S}_{n,2}) = 6n - 10 > 5n - 3 \geq \xi^c(\mathsf{H}_{n,0})$ for $n \geq 8$, we deduce that $\mathsf{S}_{6,2}$ is the only graph with maximum eccentric connectivity index when $n = 6$ and $p = 0$, while $\mathsf{H}_{n,0}$ is the only optimal graph when $n \geq 7$ and $p = 0$. This is summarized in the following Corollary.

Corollary 6. *Let G be a connected graph of order $n \geq 4$ with $p \leq n - 3$ pendant vertices.*

- *If $p \geq 1$ then $\xi^c(G) \geq \xi^c(\mathsf{H}_{n,p})$, with equality if and only if $G \simeq \mathsf{H}_{n,p}$;*

- *If $p = 0$ then*
  - *if $n = 4$ then $\xi^c(G) \geq 12$, with equality if and only if $G \simeq \mathsf{K}_4$;*
  - *if $n = 5$ then $\xi^c(G) \geq 20$, with equality if and only if $G \simeq \mathsf{H}_{5,0}$, $\mathsf{S}_{5,2}$, K_5 or C_5;*
  - *if $n = 6$ then $\xi^c(G) \geq 26$, with equality if and only if $G \simeq \mathsf{S}_{6,2}$;*
  - *if $n \geq 7$ then $\xi^c(G) \geq \xi^c(\mathsf{H}_{n,0})$, with equality if and only if $G \simeq \mathsf{H}_{n,0}$.*

6.4 Conclusion

We have characterized the graphs with the smallest eccentric connectivity index among those of fixed order n and fixed or non-fixed number of pendant vertices. The proof of this result was helped by TransProof which found graphs that were not covered by the transformations, allowing us to correct the proof. A characterization of graphs with a fixed order n and a fixed size m minimizing ξ^c(G) was given in [96]. It reads as follows.

Theorem 7 ([96])**.** *Let G be a connected graph of order n with m edges, where $n - 1 \leq m < \binom{n}{2}$. Also, let*

$$k = \left\lfloor \frac{2n - 1 - \sqrt{(2n-1)^2 - 8m}}{2} \right\rfloor.$$

Then $\xi^c(G) \geq 4m - k(n-1)$, with equality if and only if G has k dominating vertices and $n - k$ vertices of eccentricity 2.

It is, however, an open question to characterize the graphs with the largest eccentric connectivity index among those of fixed order n and fixed size m. The following conjecture appears in our paper [55], which was not presented in this thesis as it does not use transformations. We define $\mathsf{E}_{n,D,k}$ as the graph of order n constructed from a path $u_0 - u_1 - \ldots - u_D$ by joining each vertex of a clique K_{n-D-1} to u_0 and u_1, and k vertices of the clique to u_2.

Conjecture 8. *Let G be a connected graph of order n with m edges, where $n-1 \leq m \leq \binom{n-1}{2}$. Also, let*

$$D = \left\lfloor \frac{2n+1-\sqrt{17+8(m-n)}}{2} \right\rfloor \text{ and } k = m - \binom{n-D+1}{2} - D + 1.$$

Then $\xi^c(G) \leq \xi^c(\mathsf{E}_{n,D,k})$, with equality if and only if $G \simeq \mathsf{E}_{n,D,k}$ if $D > 3$ or $D = 3$, $k = n-4$ and G is the graph constructed from a path $u_0 - u_1 - u_2 - u_3$, by joining $1 \leq i \leq n-3$ vertices of a clique K_{n-4} to u_0, u_1, u_2 and the $n-4-i$ other vertices of K_{n-4} to u_1, u_2, u_3.

Chapter 7

On the average number of colors in the non-equivalent colorings of a graph

Following our work on the eccentric connectivity index of a graph (see Chapter 6), we studied another invariant: the average number of colors in the non-equivalent colorings of a graph. This time, we produced proofs by transformation for some cases, but also conjectures about the effect of some transformations on the invariant. Indeed, we could check with TransProof that removing a vertex from a graph G with $n \leq 10$ vertices will always produce a graph G' with a lower value of this invariant. But we could not prove it for higher values of n. Two papers were written and submitted for publication [59,60]. This chapter is based on those articles, with some changes to highlight transformations and uses of TransProof.

7.1 Introduction

The total number $\mathcal{B}(G)$ of non-equivalent colorings (i.e., with different partitions into color classes) of a graph G is the number of partitions of the vertex set of G whose blocks are stable sets (i.e., sets of pairwise non-adjacent

vertices). This invariant has been studied by several authors in the last few years [1, 39, 40, 45, 58, 68] under the name of (graphical) Bell number[1].

Recently, Hertz et *a*l. [56] have defined a new graph invariant $\mathcal{A}(G)$ which is equal to the average number of colors in the non-equivalent colorings of a graph G. It can be seen as a generalization of a concept linked to Bell numbers. More precisely, the Bell numbers $(\mathsf{B}_n)_{n\geq 0}$ count the number of different ways to partition a set that has exactly n elements. The 2-Bell numbers $(\mathsf{T}_n)_{n\geq 0}$ count the total number of blocks in all partitions of a set of n elements. Odlyzko and Richmond [80] have studied the average number A_n of blocks in a partition of a set of n elements, which can be defined as $\mathsf{A}_n = \frac{\mathsf{T}_n}{\mathsf{B}_n}$. The graph invariant $\mathcal{A}(G)$ that we study in this chapter generalizes A_n. Indeed, when constraints (represented by edges in G) impose that certain pairs of elements (represented by vertices) cannot belong to the same block of a partition, $\mathcal{A}(G)$ is the average number of blocks in the partitions that respect all constraints. Hence, for a graph of order n, $\mathcal{A}(G) = \mathsf{A}_n$ if G is the empty graph of order n.

As shown in [56], $\mathcal{A}(G)$ can help discover nontrivial inequalities for the Bell numbers. For example, the authors show that $\mathcal{A}(\mathsf{P}_n) = \frac{\mathsf{B}_n}{\mathsf{B}_{n-1}}$ and $\mathcal{A}(\mathsf{P}_n) < \mathcal{A}(\mathsf{P}_{n+1})$ for $n \geq 1$, where P_n is the path on n vertices. This immediately implies $\mathsf{B}_n^2 < \mathsf{B}_{n-1}\mathsf{B}_{n+1}$, which means that the sequence $(\mathsf{B}_n)_{n\geq 0}$ is strictly log-convex. This result has also been proved recently by Alzer [5] using numerical arguments.

In the next section, we fix some notations, while Section 7.3 is devoted to basic properties of $\mathcal{A}(G)$. In Section 7.4, we give values of $\mathcal{A}(G)$ for some particular graphs G that we will deal with later. The following section introduces necessary notations for this chapter. We then establish some properties before providing values for some specific classes of graphs. Section 7.5 studies lower bounds of $\mathcal{A}(G)$ and Section 7.6 studies upper bounds. In Section 7.7, we give an improved algorithm to compute the value of $\mathcal{A}(G)$. The results are summarized in Section 7.8.

[1] A formal definition is given on page 96.

7.2 Notation

For basic notions of graph theory that are not defined here, we refer the reader to Diestel [33]. We write $\mathsf{K}_{a,b}$ for the *complete bipartite graph* where a and b are the cardinalities of the two sets of vertices of the bipartition. For a subset S of vertices in a graph G, we write $G[S]$ for the subgraph of G induced by S.

complete bipartite graph

Let $N(v)$ be the set of neighbors of a vertex v in G. A vertex v is *isolated* if $|N(v)| = 0$ and is *dominant* if $|N(v)| = n - 1$ (where n is the order of G). We write $\Delta(G)$ for the *maximum degree* of G. Given a vertex v of a graph G, we say that v is *simplicial* if the induced subgraph $G[N(v)]$ of G is a clique. A graph is *triangulated* or *chordal* if each of its induced subgraphs contains a simplicial vertex or equivalently if it contains no induced cycle of order ≥ 4.

maximum degree
simplicial vertex
triangulated graph

Let G be a graph of order n and let u and v be two vertices in G. We use the following notations:

- $G_{|(u,v)}$ is the graph (of order $n-1$) obtained from G by *identifying* (merging) the vertices u and v and, if $(u,v) \in E(G)$, by removing the edge (u,v);
- if $(u,v) \in E(G)$, $G - (u,v)$ is the graph obtained by removing the edge (u,v) from G;
- if $(u,v) \notin E(G)$, $G + (u,v)$ is the graph obtained by adding the edge (u,v) in G;
- $G - v$ is the graph obtained from G by removing v and all its incident edges.

identifying

Given two graphs G_1 and G_2 (with disjoint sets of vertices), we write $G_1 \cup G_2$ for the *disjoint union* of G_1 and G_2, while the graph obtained from $G_1 \cup G_2$ by adding all possible edges between the vertices of G_1 and those of G_2 is the *join* $G_1 + G_2$ of G_1 and G_2. Also, $G \cup p\mathsf{K}_1$ is the graph obtained from G by adding p isolated vertices.

disjoint union
join

A *coloring* of a graph G is an assignment of colors to the vertices of G such that adjacent vertices have different colors. The *chromatic number* $\chi(G)$

of G is the minimum number of colors in a coloring of G. Two colorings are *equivalent* if they induce the same partition of the vertex set into color classes. So for instance, all permutations of the colors in a fixed coloring are equivalent. Let $S(G,k)$ be the number of non-equivalent colorings of a graph G that use *exactly* k colors. Then, the total number $\mathcal{B}(G)$ of non-equivalent colorings of a graph G is defined by

equivalent colorings

$$\mathcal{B}(G) = \sum_{k=1}^{n} S(G,k) = \sum_{k=\chi(G)}^{n} S(G,k),$$

and the total number $\mathcal{T}(G)$ of color classes in the non-equivalent colorings of a graph G is defined by

$$\mathcal{T}(G) = \sum_{k=1}^{n} kS(G,k) = \sum_{k=\chi(G)}^{n} kS(G,k).$$

The average number $\mathcal{A}(G)$ of colors in the non-equivalent colorings of a graph G can therefore be defined as

$$\mathcal{A}(G) = \frac{\mathcal{T}(G)}{\mathcal{B}(G)}.$$

For illustration, as shown in Figure 7.1, there are one non-equivalent coloring of P_4 with 2 colors, three with 3 colors, and one with 4 colors, which gives $\mathcal{B}(\mathsf{P}_4) = 5$, $\mathcal{T}(\mathsf{P}_4) = 15$ and $\mathcal{A}(\mathsf{P}_4) = \frac{15}{5} = 3$.

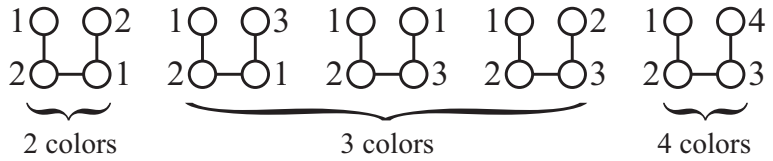


Figure 7.1: The non-equivalent colorings of P_4.

7.3 Properties of $S(G,k)$ and $\mathcal{A}(G)$

deletion-contraction rule

As for several other invariants in graph coloring, the *deletion-contraction* rule (also often called the *Fundamental Reduction Theorem* [37]) can be used

to compute $\mathcal{B}(G)$ and $\mathcal{T}(G)$. More precisely, let u and v be any pair of distinct vertices of G. As shown in [40,68], we have

$$S(G,k) = S(G-(u,v),k) - S(G_{|(u,v)},k) \quad \forall (u,v) \in E(G), \tag{7.1}$$

$$S(G,k) = S(G+(u,v),k) + S(G_{|(u,v)},k) \quad \forall (u,v) \notin E(G). \tag{7.2}$$

It follows that

$$\left.\begin{array}{l} \mathcal{B}(G) = \mathcal{B}(G-(u,v)) - \mathcal{B}(G_{|(u,v)}) \\ \mathcal{T}(G) = \mathcal{T}(G-(u,v)) - \mathcal{T}(G_{|(u,v)}) \end{array}\right\} \quad \forall (u,v) \in E(G), \tag{7.3}$$

$$\left.\begin{array}{l} \mathcal{B}(G) = \mathcal{B}(G+(u,v)) + \mathcal{B}(G_{|(u,v)}) \\ \mathcal{T}(G) = \mathcal{T}(G+(u,v)) + \mathcal{T}(G_{|(u,v)}) \end{array}\right\} \quad \forall (u,v) \notin E(G). \tag{7.4}$$

This rule is very useful to study the impact of removing or adding an edge to a graph. We next give several results on transformations and their impact on the value of $\mathcal{A}(G)$ for a graph G.

But first, we give a useful Theorem to prove those properties.

Theorem 9. *Given any two graphs G_1 and G_2, we have*

$$\mathcal{A}(G_1+G_2) = \mathcal{A}(G_1) + \mathcal{A}(G_2).$$

Proof. As observed in [1], given any coloring of G_1+G_2, none of the vertices of G_1 can share a color with a vertex of G_2, which immediately gives $\mathcal{B}(G_1+G_2) = \mathcal{B}(G_1)\mathcal{B}(G_2)$. For $\mathcal{T}(G_1+G_2)$, assuming that G_1 and G_2 are of order n_1 and n_2, respectively, we get

$$\begin{aligned}
\mathcal{T}(G_1+G_2) &= \sum_{k=1}^{n_1}\sum_{k'=1}^{n_2}(k+k')S(G_1,k)S(G_2,k') \\
&= \sum_{k=1}^{n_1} S(G_1,k)\sum_{k'=1}^{n_2}(k+k')S(G_2,k') \\
&= \sum_{k=1}^{n_1} S(G_1,k)\left(k\sum_{k'=1}^{n_2}S(G_2,k') + \sum_{k'=1}^{n_2}k'S(G_2,k')\right) \\
&= \sum_{k=1}^{n_1} kS(G_1,k)\sum_{k'=1}^{n_2}S(G_2,k') + \sum_{k=1}^{n_1}S(G_1,k)\sum_{k'=1}^{n_2}k'S(G_2,k')
\end{aligned}$$

$$= \mathcal{T}(G_1)\mathcal{B}(G_2) + \mathcal{B}(G_1)\mathcal{T}(G_2).$$

Hence,

$$\begin{aligned}\mathcal{A}(G_1 + G_2) &= \frac{\mathcal{T}(G_1 + G_2)}{\mathcal{B}(G_1 + G_2)} = \frac{\mathcal{T}(G_1)\mathcal{B}(G_2) + \mathcal{B}(G_1)\mathcal{T}(G_2)}{\mathcal{B}(G_1)\mathcal{B}(G_2)} \\ &= \frac{\mathcal{T}(G_1)}{\mathcal{B}(G_1)} + \frac{\mathcal{T}(G_2)}{\mathcal{B}(G_2)} = \mathcal{A}(G_1) + \mathcal{A}(G_2).\end{aligned}$$

□

According to TransProof, removing a vertex from a graph G of order $n \leq 10$ will always produce a graph G' such that $\mathcal{A}(G) > \mathcal{A}(G')$. However, we could not prove this property for an arbitrary value of n. The following properties show that removing a vertex under some condition will always decrease $\mathcal{A}(G)$. The following Corollary is also proved in [56].

Corollary 10. *If v is a dominant vertex of a graph G, then,*

$$\mathcal{A}(G) = \mathcal{A}(G - v) + 1.$$

Proof. If v is a dominant vertex of a graph G, then $G \simeq (G - v) + \mathsf{K}_1$, and since $\mathcal{A}(\mathsf{K}_1) = 1$, Theorem 9 gives $\mathcal{A}(G) = \mathcal{A}(G - v) + 1$. □

In the following, given a subset W of vertices in a graph G, we denote by $S_{W,i}(G, k)$ the number of non-equivalent colorings of G that use exactly k colors, and where exactly i of them appear on W. Hence, $S(G, k) = \sum_{i=0}^{|W|} S_{W,i}(G, k)$.

Lemma 11. *Let v be a vertex in a graph G of order n and let $N(v)$ be its set of neighbors in G. Then*

- $\mathcal{B}(G) = \mathcal{B}(G - v) + \sum_{k=1}^{n-1} \sum_{i=0}^{|N(v)|} (k - i) S_{N(v),i}(G - v, k)$, *and*
- $\mathcal{T}(G) = \mathcal{T}(G - v) + \sum_{k=1}^{n-1} \sum_{i=0}^{|N(v)|} (k(k - i) + 1) S_{N(v),i}(G - v, k)$.

Proof. Since $S_{N(v),i}(G,k) = S_{N(v),i}(G-v,k-1) + (k-i)S_{N(v),i}(G-v,k)$, we have

$$\begin{aligned}
\mathcal{B}(G) &= \sum_{k=1}^{n} S(G,k) = \sum_{k=1}^{n}\sum_{i=0}^{|N(v)|} S_{N(v),i}(G,k) \\
&= \sum_{k=1}^{n}\sum_{i=0}^{|N(v)|} S_{N(v),i}(G-v,k-1) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k-i)S_{N(v),i}(G-v,k) \\
&= \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} S_{N(v),i}(G-v,k) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k-i)S_{N(v),i}(G-v,k) \\
&= \mathcal{B}(G-v) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k-i)S_{N(v),i}(G-v,k)
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{T}(G) &= \sum_{k=1}^{n} kS(G,k) = \sum_{k=1}^{n}\sum_{i=0}^{|N(v)|} kS_{N(v),i}(G,k) \\
&= \sum_{k=1}^{n}\sum_{i=0}^{|N(v)|} kS_{N(v),i}(G-v,k-1) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} k(k-i)S_{N(v),i}(G-v,k) \\
&= \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k+1)S_{N(v),i}(G-v,k) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} k(k-i)S_{N(v),i}(G-v,k) \\
&= \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} kS_{N(v),i}(G-v,k) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k(k-i)+1)S_{N(v),i}(G-v,k) \\
&= \mathcal{T}(G-v) + \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k(k-i)+1)S_{N(v),i}(G-v,k).
\end{aligned}$$

□

Theorem 12. *Let v be a vertex in a graph G. If $\chi(G[N(v)]) \geq |N(v)|-3$ then $\mathcal{A}(G) > \mathcal{A}(G-v)$.*

Proof. Let n be the order of G. We know from Lemma 11 that

$$\mathcal{A}(G) - \mathcal{A}(G-v) = \frac{\mathcal{T}(G-v)+a}{\mathcal{B}(G-v)+b} - \frac{\mathcal{T}(G-v)}{\mathcal{B}(G-v)} = \frac{a\mathcal{B}(G-v) - b\mathcal{T}(G-v)}{\mathcal{B}(G)\mathcal{B}(G-v)}$$

where $a = \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k(k-i)+1)S_{N(v),i}(G-v,k)$ and $b = \sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} (k-i)S_{N(v),i}(G-v,k)$.

It suffices to show that $a\mathcal{B}(G-v)-b\mathcal{T}(G-v) > 0$. Let $\mathcal{P}$ be the set of pairs (k,i) such that $S_{N(v),i}(G-v,k)>0$. Since $\chi(G[N(v)]) \geq |N(v)|-3$, we have $k \geq i \geq |N(v)|-3$ for all $(k,i) \in \mathcal{P}$. For two pairs (k,i) and (k',i') in $\mathcal{P}$, we write $(k,i)>(k',i')$ if $k>k'$ or $k=k'$ and $i>i'$. Also, let $f(k,k',i,i') = S_{N(v),i}(G-v,k)S_{N(v),i'}(G-v,k')$. We have

$$
\begin{aligned}
&a\mathcal{B}(G-v) - b\mathcal{T}(G-v)\\
=&\, a\sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} S_{N(v),i}(G-v,k) - b\sum_{k=1}^{n-1}\sum_{i=0}^{|N(v)|} kS_{N(v),i}(G-v,k)\\
=&\sum_{(k,i)\in\mathcal{P}} S_{N(v),i}(G-v,k)^2\Big((k(k-i)+1)-(k-i)k\Big)\\
&+\sum_{(k,i)>(k',i')} f(k,k',i,i')\Big((k(k-i)+1)+(k'(k'-i')+1)-(k-i)k'-(k'-i')k\Big)\\
=&\sum_{(k,i)\in\mathcal{P}} S_{N(v),i}(G-v,k)^2\\
&+\sum_{(k,i)>(k',i')} f(k,k',i,i')\Big((k-k')^2-(k-k')(i-i')+2\Big).
\end{aligned}
$$

Note that since $S_{N(v),|N(v)|}(G-v,n-1) = 1$, $\mathcal{P} \neq \emptyset$. Hence, we have $\sum_{(k,i)\in\mathcal{P}} S_{N(v),i}(G-v,k)^2>0$, and it is sufficient to prove that $(k-k')^2-(k-k')(i-i')+2 \geq 0$ for every two pairs (k,i) and (k',i') in $\mathcal{P}$ with $(k,i)>(k',i')$. For two such pairs (k,i) and (k',i'), we have $i-i' \leq |N(v)|-(|N(v)|-3) = 3$. Hence,

- if $k=k'$, then $(k-k')^2-(k-k')(i-i')+2 = 2 > 0$;
- if $k=k'+1$, then $(k-k')^2-(k-k')(i-i')+2 = 3-(i-i') \geq 0$;
- if $k=k'+2$, then $(k-k')^2-(k-k')(i-i')+2 = 6-2(i-i') \geq 0$;
- if $k \geq k'+3$, then $(k-k')^2-(k-k')(i-i')+2 \geq 2$. □

Corollary 13. *If v is a vertex of degree at most 4 in a graph G, then $\mathcal{A}(G) > \mathcal{A}(G-v)$.*

Proof. Since $|N(v)| \leq 4$, we have:

- if $N(v)=\emptyset$, then $\chi(G[N(v)]) = 0 > -3 = |N(v)|-3$;

- if $N(v) \neq \emptyset$, then $\chi(G[N(v)]) \geq 1 \geq |N(v)| - 3$.

In both cases, we conclude from Theorem 12 that $\mathcal{A}(G) > \mathcal{A}(G - v)$.

□

Corollary 14. *Let v be a simplicial vertex in a graph G. Then $\mathcal{A}(G) > \mathcal{A}(G - v)$.*

Proof. Since v is simplicial in G, we have $\chi(G[N(v)]) = |N(v)| > |N(v)| - 3$. Hence, we conclude from Theorem 12 that $\mathcal{A}(G) > \mathcal{A}(G - v)$.

□

Given that removing a vertex seems to always result in a decrease in the average number of colors, one might think that removing an edge would have the same effect. Indeed, removing an edge allows more colorings using fewer colors. But this is not true. Using TransProof, we found that removing any edge from the graph of Figure 7.2 produces a graph with a larger average number of colors. And it is not the only example.

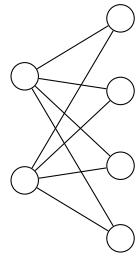


Figure 7.2: Any edge removal results in an increase of $\mathcal{A}(G)$.

Thus, instead of trying to prove this incorrect property, we restricted ourselves to simplicial vertices.

Theorem 15. *Let v be a simplicial vertex of degree at least one in a graph G of order n, and let w be one of its neighbors in G. Then $\mathcal{A}(G) > \mathcal{A}(G - (v, w))$.*

Proof. Let $H = (G - v) \cup \mathsf{K}_1$. In other words, H is obtained from $G - v$ by

adding an isolated vertex. It follows from Lemma 11 that

$$\begin{aligned}\mathcal{B}(H) &= \mathcal{B}(G-v) + \sum_{k=1}^{n-1}\sum_{i=0}^{0}(k-i)S_{\emptyset,i}(G-v,k)\\ &= \mathcal{B}(G-v) + \sum_{k=1}^{n-1} kS(G-v,k)\end{aligned}$$

and

$$\begin{aligned}\mathcal{T}(H) &= \mathcal{T}(G-v) + \sum_{k=1}^{n-1}\sum_{i=0}^{0}(k(k-i)+1)S_{\emptyset,i}(G-v,k)\\ &= \mathcal{T}(G-v) + \sum_{k=1}^{n-1}(k^2+1)S(G-v,k).\end{aligned}$$

Also, since v is a simplicial vertex, $S_{N(v),i}(G-v,k)=0$ for $i \neq |N(v)|$, and we have that $S(G-v,k)=S_{N(v),|N(v)|}(G-v,k)$. It follows from Lemma 11 that

$$\begin{aligned}\mathcal{B}(G) &= \mathcal{B}(G-v) + \sum_{k=1}^{n-1}(k-|N(v)|)S(G-v,k)\\ &= \Big(\mathcal{B}(G-v) + \sum_{k=1}^{n-1} kS(G-v,k)\Big) - |N(v)|\sum_{k=1}^{n-1} S(G-v,k)\\ &= \mathcal{B}(H) - |N(v)|\mathcal{B}(G-v)\end{aligned}$$

and

$$\begin{aligned}\mathcal{T}(G) &= \mathcal{T}(G-v) + \sum_{k=1}^{n-1}\Big(k(k-|N(v)|)+1\Big)S(G-v,k)\\ &= \Big(\mathcal{T}(G-v) + \sum_{k=1}^{n-1}(k^2+1)S(G-v,k)\Big) - |N(v)|\sum_{k=1}^{n-1} kS(G-v,k)\\ &= \mathcal{T}(H) - |N(v)|\mathcal{T}(G-v).\end{aligned}$$

Similarly, since v is simplicial (of degree $|N(v)|-1$) in $G-(v,w)$, we have

$$\mathcal{B}(G-(v,w)) = \mathcal{B}(H) - (|N(v)|-1)\mathcal{B}(G-v)$$

and

$$\mathcal{T}(G-(v,w)) = \mathcal{T}(H) - (|N(v)|-1)\mathcal{T}(G-v).$$

Hence,

$$\begin{aligned}\mathcal{A}(G)-\mathcal{A}(G-(v,w)) &= \frac{\mathcal{T}(G)}{\mathcal{B}(G)}-\frac{\mathcal{T}(G-(v,w))}{\mathcal{B}(G-(v,w))}\\ &= \frac{\mathcal{T}(H)-|N(v)|\mathcal{T}(G-v)}{\mathcal{B}(H)-|N(v)|\mathcal{B}(G-v)}-\frac{\mathcal{T}(H)-(|N(v)|-1)\mathcal{T}(G-v)}{\mathcal{B}(H)-(|N(v)|-1)\mathcal{B}(G-v)}\\ &= \frac{\mathcal{T}(H)\mathcal{B}(G-v)-\mathcal{B}(H)\mathcal{T}(G-v)}{\mathcal{B}(G)\mathcal{B}(G-(v,w))}.\end{aligned}$$

Since v is isolated in H, it is simplicial and we know from Corollary 14 that

$$\begin{aligned}\mathcal{A}(H)>\mathcal{A}(G-v) &\iff \frac{\mathcal{T}(H)}{\mathcal{B}(H)}>\frac{\mathcal{T}(G-v)}{\mathcal{B}(G-v)}\\ &\iff \mathcal{T}(H)\mathcal{B}(G-v)-\mathcal{B}(H)\mathcal{T}(G-v)>0\end{aligned}$$

which implies $\mathcal{A}(G)-\mathcal{A}(G-(v,w))>0$.

□

The following properties will also be useful to prove our results.

Lemma 16. *Let G and H be two graphs, and suppose H has order n. Then*

- $\mathcal{B}(G\cup H)=\sum_{k=1}^{n} S(H,k)\mathcal{B}(G\cup \mathsf{K}_k)$, *and*
- $\mathcal{T}(G\cup H)=\sum_{k=1}^{n} S(H,k)\mathcal{T}(G\cup \mathsf{K}_k)$.

Proof. We first prove that $\mathcal{B}(G\cup H)=\sum_{k=1}^{n} S(H,k)\mathcal{B}(G\cup \mathsf{K}_k)$ for all graphs H of order n. For $n=1$, we have $H=\mathsf{K}_1$, and since $S(\mathsf{K}_1,1)=1$, we have $\mathcal{B}(G\cup \mathsf{K}_1)=\sum_{k=1}^{1} S(\mathsf{K}_1,k)\mathcal{B}(G\cup \mathsf{K}_1)$. For larger values of n we proceed by double induction on the order n and the size m of H. So assume H has order n and size m.

- If $m=\frac{n(n-1)}{2}$, then $H=\mathsf{K}_n$. Since $S(\mathsf{K}_n,i)=0$ for $i=1,\ldots,n-1$ and $S(\mathsf{K}_n,n)=1$, we have $\mathcal{B}(G\cup \mathsf{K}_n)=\sum_{k=1}^{n} S(\mathsf{K}_n,k)\mathcal{B}(G\cup \mathsf{K}_n)$.
- If $m<\frac{n(n-1)}{2}$, then H contains two non-adjacent vertices u and v, and we know from Equations (7.4) that $\mathcal{B}(G\cup H)=\mathcal{B}(G\cup(H+(u,v)))+$

$\mathcal{B}(G \cup H_{|(u,v)})$. Since $H + (u,v)$ has order n and size $m+1$ and $H_{|(u,v)}$ has order $n-1$, we know by induction that

$$\begin{aligned}\mathcal{B}(G \cup H) &= \sum_{k=1}^{n} S(H+(u,v),k)\mathcal{B}(G \cup \mathsf{K}_k) + \sum_{k=1}^{n-1} S(H_{|(u,v)},k)\mathcal{B}(G \cup \mathsf{K}_k) \\ &= \sum_{k=1}^{n} \big(S(H+(u,v),k) + S(H_{|(u,v)},k)\big)\mathcal{B}(G \cup \mathsf{K}_k) \\ &= \sum_{k=1}^{n} S(H,k)\mathcal{B}(G \cup \mathsf{K}_k).\end{aligned}$$

The proof for $\mathcal{T}(G \cup H)$ is similar.

□

Theorem 17. *Let H_1, H_2 be any two graphs. If, for all $k>k'$, we have that $S(H_1,k)S(H_2,k') \geq S(H_2,k)S(H_1,k')$, the inequality being strict for at least one pair (k,k'), then $\mathcal{A}(G \cup H_1) > \mathcal{A}(G \cup H_2)$ for all graphs G.*

Proof. Let $f(k,k')=S(H_1,k)S(H_2,k')-S(H_2,k)S(H_1,k')$ and assume that H_1 and H_2 are of order n_1 and n_2, respectively. Note that $n_1 \geq n_2$ else we would have $n_2 > n_1$ and $f(n_2,n_1)=S(H_1,n_2)S(H_2,n_1)-S(H_2,n_2)S(H_1,n_1) = -1 < 0$. We know from Lemma 16 that

$$\begin{aligned}\mathcal{A}(G \cup H_1) - \mathcal{A}(G \cup H_2) &= \frac{\sum_{k=1}^{n_1} S(H_1,k)\mathcal{T}(G \cup \mathsf{K}_k)}{\sum_{k=1}^{n_1} S(H_1,k)\mathcal{B}(G \cup \mathsf{K}_k)} - \frac{\sum_{k=1}^{n_2} S(H_2,k)\mathcal{T}(G \cup \mathsf{K}_k)}{\sum_{k=1}^{n_2} S(H_2,k)\mathcal{B}(G \cup \mathsf{K}_k)} \\ &= \frac{\sum_{k=1}^{n_1}\sum_{k'=1}^{n_1} f(k,k')\mathcal{T}(G \cup \mathsf{K}_k)\mathcal{B}(G \cup \mathsf{K}_{k'})}{\mathcal{B}(G \cup H_1)\mathcal{B}(G \cup H_2)}.\end{aligned}$$

Since $f(k,k) = 0$ for all k and $f(k,k') = -f(k',k)$ for all $k \neq k'$, we deduce

$$\begin{aligned}\mathcal{A}(G \cup H_1) - \mathcal{A}(G \cup H_2) = &\frac{\sum_{k'=1}^{n_1-1}\sum_{k=k'+1}^{n_1} f(k,k')\mathcal{T}(G \cup \mathsf{K}_k)\mathcal{B}(G \cup \mathsf{K}_{k'})}{\mathcal{B}(G \cup H_1)\mathcal{B}(G \cup H_2)} \\ &- \frac{\sum_{k'=1}^{n_1-1}\sum_{k=k'+1}^{n_1} f(k,k')\mathcal{T}(G \cup \mathsf{K}_{k'})\mathcal{B}(G \cup \mathsf{K}_k)}{\mathcal{B}(G \cup H_1)\mathcal{B}(G \cup H_2)}\end{aligned}$$

$$= \frac{\sum_{k'=1}^{n_1-1} \sum_{k=k'+1}^{n_1} f(k,k') \Big(\mathcal{T}(G \cup \mathsf{K}_k) \mathcal{B}(G \cup \mathsf{K}_{k'}) - \mathcal{T}(G \cup \mathsf{K}_{k'}) \mathcal{B}(G \cup \mathsf{K}_k) \Big)}{\mathcal{B}(G \cup H_1) \mathcal{B}(G \cup H_2)}.$$

Note that if $k > k'$, then $G \cup \mathsf{K}_k$ is obtained from $G \cup \mathsf{K}_{k'}$ by repeatedly adding a simplicial vertex. Hence, we know from Corollary 14 that

$$\begin{aligned} \mathcal{A}(G \cup \mathsf{K}_k) > \mathcal{A}(G \cup \mathsf{K}_{k'}) \iff & \quad \frac{\mathcal{T}(G \cup \mathsf{K}_k)}{\mathcal{B}(G \cup \mathsf{K}_k)} > \frac{\mathcal{T}(G \cup \mathsf{K}_{k'})}{\mathcal{B}(G \cup \mathsf{K}_{k'})} \\ \iff & \quad \mathcal{T}(G \cup \mathsf{K}_k) \mathcal{B}(G \cup \mathsf{K}_{k'}) \\ & \quad - \mathcal{T}(G \cup \mathsf{K}_{k'}) \mathcal{B}(G \cup \mathsf{K}_k) > 0. \end{aligned}$$

Since $f(k,k') = S(H_1,k)S(H_2,k') - S(H_1,k')S(H_2,k)$ is positive for all pairs $k > k'$, and strictly positive for at least one such pair, we have $\mathcal{A}(G \cup H_1) - \mathcal{A}(G \cup H_2) > 0$.

□

As a final property, we mention one which is proved in [56] and which will be helpful in proving results in the following sections.

Theorem 18 ([56])**.** *Let G, H and $F_1, \cdots, F_r$ be $r+2$ graphs, and let $\alpha_1, \cdots, \alpha_r$ be r positive numbers such that*

- $\mathcal{B}(G) = \mathcal{B}(H) + \sum_{i=1}^{r} \alpha_i \mathcal{B}(F_i)$,
- $\mathcal{T}(G) = \mathcal{T}(H) + \sum_{i=1}^{r} \alpha_i \mathcal{T}(F_i)$, *and*
- $\mathcal{A}(F_i) < \mathcal{A}(H)$ *for all* $i = 1, \cdots, r$.

Then $\mathcal{A}(G) < \mathcal{A}(H)$.

7.4 Some values for $\mathcal{A}(G)$

The value $\mathcal{A}(G)$ is known for some graphs G. We mention here some of them which are proven in [56] and determine some others.

Proposition 19. *[56]*

- $\mathcal{A}\left(\overline{\mathsf{K}}_n\right) = \mathcal{A}\left(n\mathsf{K}_1\right) = \dfrac{\mathsf{B}_{n+1} - \mathsf{B}_n}{\mathsf{B}_n}$ *for all* $n \geq 1$*;*
- $\mathcal{A}\left(T \cup p\mathsf{K}_1\right) = \dfrac{\sum_{i=0}^{p} \binom{p}{i} \mathsf{B}_{n+i}}{\sum_{i=0}^{p} \binom{p}{i} \mathsf{B}_{n+i-1}}$ *for all trees* T *of order* $n \geq 1$ *and all* $p \geq 0$*;*
- $\mathcal{A}\left(\mathsf{C}_n \cup p\mathsf{K}_1\right) = \dfrac{\sum_{j=1}^{n-1}(-1)^{j+1} \sum_{i=0}^{p} \binom{p}{i} \mathsf{B}_{n+i-j+1}}{\sum_{j=1}^{n-1}(-1)^{j+1} \sum_{i=0}^{p} \binom{p}{i} \mathsf{B}_{n+i-j}}$ *for all* $n \geq 3$ *and* $p \geq 0$.

Since $S(\mathsf{K}_n, k) = 1$ for $k = n$, and $S(\mathsf{K}_n, k) = 0$ for $k < n$, we have $\mathcal{A}\left(\mathsf{K}_n\right) = n$. We prove here a stronger property, which we use in the next section. Let $\left\{{a \atop b}\right\}$ be the Stirling number of the second kind, with parameters a and b (i.e., the number of partitions of a set of a elements into b blocks).

Proposition 20.

$$\mathcal{A}\left(\mathsf{K}_n \cup p\mathsf{K}_1\right) = \frac{\sum_{k=n}^{n+p} k \sum_{j=0}^{n} \binom{k-j}{n-j}\binom{n}{j}(n-j)!\left\{{p \atop k-j}\right\}}{\sum_{k=n}^{n+p} \sum_{j=0}^{n} \binom{k-j}{n-j}\binom{n}{j}(n-j)!\left\{{p \atop k-j}\right\}} \quad \textit{for all } n \geq 1 \textit{ and all } p \geq 0.$$

Proof. It is proved in [58] that given two graphs G_1 and G_2, we have

$$S(G_1 \cup G_2, k) = \sum_{i=1}^{k} \sum_{j=0}^{i} \binom{i}{j}\binom{k-j}{i-j}(i-j)!S(G_1, i)S(G_2, k-j).$$

For $G_1 \simeq \mathsf{K}_n$ and $G_2 \simeq p\mathsf{K}_1$, we have $S(G_1, i) = 1$ if $i = n$, and $S(G_1, i) = 0$ otherwise. Also, $S(G_2, k-j) = \left\{{p \atop k-j}\right\}$. Hence,

$$S(\mathsf{K}_n \cup p\mathsf{K}_1, k) = \sum_{j=0}^{n} \binom{k-j}{n-j}\binom{n}{j}(n-j)!\left\{{p \atop k-j}\right\}.$$

The result then follows from the fact that

$$\mathcal{A}\left(\mathsf{K}_n \cup p\mathsf{K}_1\right) = \frac{\sum_{k=n}^{n+p} kS(\mathsf{K}_n \cup p\mathsf{K}_1, k)}{\sum_{k=n}^{n+p} S(\mathsf{K}_n \cup p\mathsf{K}_1, k)}.$$

□

We now determine $\mathcal{A}(G)$ for G equal to the complement of a path and the complement of a cycle. In what follows, we write F_n and L_n for the nth Fibonacci number and the nth Lucas number, respectively.

Proposition 21. $\mathcal{A}\left(\overline{\mathsf{P}}_n\right) = \dfrac{(n+1)F_{n+2} + (2n-1)F_{n+1}}{5F_{n+1}}$ *for all* $n \geq 1$.

Proof. The result is true for $n \leq 2$. Indeed,

- For $n = 1$, we have $\overline{\mathsf{P}}_1 = K_1$ which implies $\mathcal{A}\left(\overline{\mathsf{P}}_1\right) = 1 = \frac{2F_3+F_2}{5F_2}$;
- For $n = 2$, we have $\overline{\mathsf{P}}_2 = \overline{K}_2$ which implies $\mathcal{A}\left(\overline{\mathsf{P}}_2\right) = \frac{\mathsf{B}_3-\mathsf{B}_2}{\mathsf{B}_2} = \frac{3}{2} = \frac{3F_4+3F_3}{5F_3}$.

For larger values of n, we proceed by induction. It is shown in [40] that $\mathcal{B}(\overline{\mathsf{P}}_n)$=$F_{n+1}$. Also, it follows from Equations (7.4) that $\mathcal{T}(\overline{\mathsf{P}}_n) = \mathcal{T}(\overline{\mathsf{P}}_{n-1} + \mathsf{K}_1) + \mathcal{T}(\overline{\mathsf{P}}_{n-2} + \mathsf{K}_1)$. Moreover, as shown in the proof of Theorem 9, we have

$$\mathcal{T}(G + \mathsf{K}_1) = \mathcal{T}(G)\mathcal{B}(\mathsf{K}_1) + \mathcal{B}(G)\mathcal{T}(\mathsf{K}_1) = \mathcal{T}(G) + \mathcal{B}(G).$$

Hence,

$$\begin{aligned}
\mathcal{A}\left(\overline{\mathsf{P}}_n\right) &= \frac{\mathcal{T}(\overline{\mathsf{P}}_{n-1}) + \mathcal{B}(\overline{\mathsf{P}}_{n-1}) + \mathcal{T}(\overline{\mathsf{P}}_{n-2}) + \mathcal{B}(\overline{\mathsf{P}}_{n-2})}{F_{n+1}} \\
&= \frac{nF_{n+1} + (2n-3)F_n}{5F_{n+1}} + \frac{F_n}{F_{n+1}} + \frac{(n-1)F_n + (2n-5)F_{n-1}}{5F_{n+1}} + \frac{F_{n-1}}{F_{n+1}} \\
&= \frac{nF_{n+1} + (3n+1)F_n + 2nF_{n-1}}{5F_{n+1}} = \frac{3nF_{n+1} + (n+1)F_n}{5F_{n+1}} \\
&= \frac{(n+1)F_{n+2} + (2n-1)F_{n+1}}{5F_{n+1}}.
\end{aligned}$$

□

Proposition 22. $\mathcal{A}\left(\overline{\mathsf{C}}_n\right) = \dfrac{nF_{n+1}}{L_n}$ *for all* $n \geq 4$.

Proof. It follows from Equations (7.4) that

$$\mathcal{T}(\overline{\mathsf{C}}_n) = \mathcal{T}(\overline{\mathsf{P}}_n) + \mathcal{T}(\overline{\mathsf{P}}_{n-2} + \mathsf{K}_1).$$

Moreover, it is shown in [40] that $\mathcal{B}(\overline{\mathsf{C}}_n) = L_n$. Since $\mathcal{T}(\overline{\mathsf{P}}_{n-2} + \mathsf{K}_1) = \mathcal{T}(\overline{\mathsf{P}}_{n-2}) + \mathcal{B}(\overline{\mathsf{P}}_{n-2})$, Proposition 21 implies

$$\begin{aligned}
\mathcal{A}\left(\overline{\mathsf{C}}_n\right) &= \frac{\mathcal{T}(\overline{\mathsf{P}}_n) + \mathcal{T}(\overline{\mathsf{P}}_{n-2}) + \mathcal{B}(\overline{\mathsf{P}}_{n-2})}{L_n} \\
&= \frac{(n+1)F_{n+2} + (2n-1)F_{n+1}}{5L_n} + \frac{(n-1)F_n + (2n-5)F_{n-1}}{5L_n} + \frac{F_{n-1}}{L_n} \\
&= \frac{(n+1)F_{n+2} + (2n-1)F_{n+1} + (n-1)F_n + 2nF_{n-1}}{5L_n} \\
&= \frac{3nF_{n+1} + 2nF_n + 2nF_{n-1}}{5L_n} = \frac{5nF_{n+1}}{5L_n} = \frac{nF_{n+1}}{L_n}.
\end{aligned}$$

□

7.5 Lower bounds and properties for the average number of colors in the non-equivalent colorings of a graph

Coloring a graph with many colors is usually quite simple and uninteresting. But using as few colors as possible is an NP-hard problem [46]. This is why we wish to study the possible lower bounds for $\mathcal{A}(G)$, as a function of n, which is reached by at least one graph of order n. We think that the best possible lower bound is $\frac{\mathsf{B}_{n+1}-\mathsf{B}_n}{\mathsf{B}_n}$ and is reached by the empty graph of order n. But it is just a conjecture we are trying to prove.

In this section, we give three conjectures for potential lower bounds on $\mathcal{A}(G)$. We then establish their validity for triangulated graphs and for graphs G with maximum degree $\Delta(G) \leq 2$.

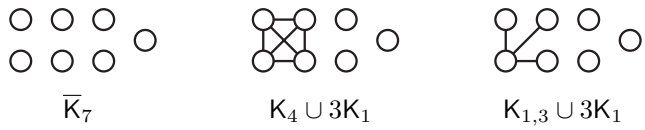


Figure 7.3: The three graphs that define the lower bounds $L_1(7)$, $L_2(7,4)$ and $L_3(7,4)$.

7.5.1 Conjectures

The lower bounds we are interested in depend on two parameters n and r with $1 \leq r \leq n$. They are equal to $\mathcal{A}(G)$ for some specific graphs G. More precisely, with the help of Propositions 19 and 20, we define

- $L_1(n) = \mathcal{A}\left(\overline{\mathsf{K}}_n\right) = \dfrac{\mathsf{B}_{n+1} - \mathsf{B}_n}{\mathsf{B}_n}$,
- $L_2(n,r) = \mathcal{A}\left(\mathsf{K}_r \cup (n-r)\,\mathsf{K}_1\right) = \dfrac{\displaystyle\sum_{k=r}^{n} k \sum_{i=0}^{r} \binom{k-i}{r-i}\binom{r}{i}(r-i)!\left\{\begin{matrix} n-r \\ k-i \end{matrix}\right\}}{\displaystyle\sum_{k=r}^{n} \sum_{i=0}^{r} \binom{k-i}{r-i}\binom{r}{i}(r-i)!\left\{\begin{matrix} n-r \\ k-i \end{matrix}\right\}}$,
- $L_3(n,r) = \mathcal{A}\left(\mathsf{K}_{1,r-1} \cup (n-r)\,\mathsf{K}_1\right) = \dfrac{\displaystyle\sum_{i=0}^{n-r} \binom{n-r}{i} \mathsf{B}_{r+i}}{\displaystyle\sum_{i=0}^{n-r} \binom{n-r}{i} \mathsf{B}_{r+i-1}}$.

For illustration, we show in Figure 7.3 the three graphs that give the bounds for $n = 7$ and $r = 4$.

Given a graph G of order n, we are interested in the following inequalities, one of them being a conjecture, the other ones being proved here below:

$$\begin{aligned} L_1(n) &\leq \min\Big\{L_2(n,\chi(G)), L_3(n,\Delta(G)+1)\Big\} \\ &\leq \max\Big\{L_2(n,\chi(G)), L_3(n,\Delta(G)+1)\Big\} \leq \mathcal{A}(G). \end{aligned}$$

The first inequality follows from Theorem 15 since $\overline{\mathsf{K}}_n$ is obtained from $\mathsf{K}_r \cup (n-r)\mathsf{K}_1$ and from $\mathsf{K}_{1,r-1} \cup (n-r)\mathsf{K}_1$ by repeatedly removing edges incident to simplicial vertices. The second inequality is trivial. The last inequality is an open problem stated in the two following conjectures.

Conjecture 23. *Let G be a graph of order n. Then,*

$$\mathcal{A}(G) \geq L_2(n, \chi(G))$$

with equality if and only if $G \simeq \mathsf{K}_{\chi(G)} \cup (n-\chi(G))\mathsf{K}_1$.

Conjecture 24. *Let G be a graph of order n. Then*

$$\mathcal{A}(G) \geq L_3(n, \Delta(G)+1)$$

with equality if and only if $G \simeq \mathsf{K}_{1,\Delta(G)} \cup (n-\Delta(G)-1)\mathsf{K}_1$.

Since $L_1(n) \leq \min\{L_2(n, \chi(G)), L_3(n, \Delta(G)+1)\}$, it suffices to show that one of these conjectures is true to prove that the empty graph $\overline{\mathsf{K}}_n$ has the minimum value for $\mathcal{A}(G)$ among all graphs G of order n. This leads to the following weaker conjecture.

Conjecture 25. *Let G be a graph of order n, then,*

$$\mathcal{A}(G) \geq L_1(n)$$

with equality if and only if $G \simeq \overline{\mathsf{K}}_n$.

Theorem 26. *Conjectures 23 and 24 (and therefore 25) are true for triangulated graphs.*

Proof. Let us first observe that removing an edge incident to a simplicial vertex in a triangulated graph gives another triangulated graph. So let G be a triangulated graph. Since G is perfect, it contains a clique K of order $|K| = \chi(G)$. It is well known that triangulated graphs that are not a clique contain at least two non-adjacent simplicial vertices [35]. Hence, G can be reduced to $\mathsf{K}_{\chi(G)} \cup (n-\chi(G))\mathsf{K}_1$ by repeatedly removing edges incident to simplicial vertices. We know from Theorem 15 that each of these edge removals strictly decreases $\mathcal{A}(G)$. We thus have $\mathcal{A}(G) \geq \mathcal{A}\left(\mathsf{K}_{\chi(G)}\right) \cup (n-\chi(G))\mathsf{K}_1)$, with equality if and only if $G \simeq \mathsf{K}_{\chi(G)} \cup (n-\chi(G))\mathsf{K}_1$. Conjecture 23 is therefore true for triangulated graphs.

Let us now deal with Conjecture 24. Let v be a vertex of degree $\Delta(G)$ in G. We consider the partition $(N_1(v), N_2(v))$ of the neighborhood $N(v)$ of v, where $N_1(v)$ contains all vertices of $N(v)$ of degree 1 in G (i.e., v is the only neighbor of every vertex of $N_1(v)$). Also, we consider the partition $(\overline{N}_1(v), \overline{N}_2(v))$ of the set $\overline{N}(v)$ of vertices of G that are not adjacent to v, where $\overline{N}_1(v)$ contains all vertices of $\overline{N}(v)$ of degree 0 in G. If $N_2(v) \cup \overline{N}_2(v) \neq \emptyset$ then $G[N_2(v) \cup \overline{N}_2(v) \cup \{v\}]$ contains a simplicial vertex $w \neq v$ (since it is also a triangulated graph). Clearly, w is simplicial in the whole graph that includes $N_1(v)$ and $\overline{N}_1(v)$.

- If $w \in N_2(v)$, we can remove all edges incident to w, except the one that links w with v. We thus get a new triangulated graph in which at least one vertex has been transferred from $N_2(v)$ to $N_1(v)$, vertices of $\overline{N}_2(v)$ may have transferred to $\overline{N}_1(v)$, but no vertex has undergone the reverse transfers.
- If $w \in \overline{N}_2(v)$, we can remove all edges incident to w. We thus get a new triangulated graph in which at least one vertex has been transferred from $\overline{N}_2(v)$ to $\overline{N}_1(v)$, vertices of $N_2(v)$ may have transferred to $N_1(v)$, but no vertex has undergone the reverse transfers.

Note that in both cases, no vertex has been transferred from $\overline{N}(v)$ to $N(v)$ or vice versa. Hence, by repeatedly applying the above-mentioned edge removals, we get $N_2(v) = \overline{N}_2(v) = \emptyset$, which means that the resulting graph is $\mathsf{K}_{1,\Delta(G)} \cup (n - \Delta(G) - 1)\mathsf{K}_1$. Again, we know from Theorem 15 that each of the edge removals performed strictly decreases $\mathcal{A}(G)$, which proves that $\mathcal{A}(G) \geq \mathcal{A}\left(\mathsf{K}_{1,\Delta(G)}\right) \cup (n - \Delta(G) - 1)\mathsf{K}_1)$, with equality if and only if $G \simeq \mathsf{K}_{1,\Delta(G)} \cup (n - \Delta(G) - 1)\mathsf{K}_1$. □

The three conjectures come from the discovery systems GraPHedron [75] and PHOEG [31]. Note that despite the apparent simplicity of Conjecture 25, its validity cannot be proven by simple intuitive means such as sequential edge removal. Indeed, there are graphs, for example $\mathsf{K}_{2,4}$, for which the removal of any edge strictly increases $\mathcal{A}(G)$. Also, we cannot proceed by induction

on the number of connected components of G. Indeed, there are pairs of graphs G_1, G_2 such that $\mathcal{A}(G_1) < \mathcal{A}(G_2)$ while $\mathcal{A}(G_1 \cup \mathsf{K}_1) > \mathcal{A}(G_2 \cup \mathsf{K}_1)$. For example, for $G_1 = \mathsf{K}_{2,3}$ and $G_2 = \mathsf{K}_3 \cup 2\mathsf{K}_1$, we have

$$\mathcal{A}(G_1) = 3.5 < 3.529 = \mathcal{A}(G_2) \text{ and } \mathcal{A}(G_1 \cup \mathsf{K}_1) = 3.867 > 3.831 = \mathcal{A}(G_2 \cup \mathsf{K}_1).$$

Note that proving that Conjecture 24 is true for all graphs G of order n and maximum degree $\Delta(G) = n-1$ is as difficult as proving Conjecture 25. Indeed, let v be a vertex of degree $n-1$ in a graph G of order n. Since v is a dominant vertex of G, we know from Corollary 10 that $\mathcal{A}(G) = \mathcal{A}(G-v)+1$. Hence, minimizing $\mathcal{A}(G)$ is equivalent to minimizing $\mathcal{A}(G-v)$, with no maximum degree constraint on $G-v$. We show in the next section that Conjectures 23 and 24 (and therefore 25) are true for graphs of maximum degree at most 2.

7.5.2 Proof of the conjectures for graphs G with $\Delta(G) \leq 2$

We start this section with a simple proof of the validity of Conjectures 23 and 24 when $\Delta(G) = 1$.

Theorem 27. *Let G be a graph of order n and maximum degree $\Delta(G) = 1$. Then,*

$$L_2(n, \chi(G)) = L_3(n, \Delta(G)+1) \leq \mathcal{A}(G),$$

with equality if and only if $G \simeq \mathsf{K}_2 \cup (n-2)\mathsf{K}_1$.

Proof. Since $\Delta(G) = 1$, we have $\chi(G) = 2$, which implies

$$\begin{aligned} & L_2(n, \chi(G)) = L_2(n, 2) = \mathcal{A}(\mathsf{K}_2 \cup (n-2)\,\mathsf{K}_1) = \mathcal{A}(\mathsf{K}_{1,1} \cup (n-2)\,\mathsf{K}_1) \\ = \quad & L_3(n, 2) = L_3(n, \Delta(G)+1). \end{aligned}$$

Note also that $\Delta(G) = 1$ implies $G \simeq p\mathsf{K}_2 \cup (n-2p)\mathsf{K}_1$ for $p \geq 1$. Hence, all vertices in G are simplicial. We can thus sequentially remove all edges of G, except one, and it follows from Theorem 15 that $\mathcal{A}(G) \geq \mathcal{A}(\mathsf{K}_2 \cup (n-2)\,\mathsf{K}_1)$, with equality if and only $G \simeq \mathsf{K}_2 \cup (n-2)\mathsf{K}_1$. □

The proofs that Conjectures 23 and 24 are true when $\Delta(G) = 2$ are more complex. We first prove some intermediate results in the form of lemmas.

Lemma 28. *$\mathcal{A}(G \cup \mathsf{C}_n) > \mathcal{A}(G \cup \mathsf{P}_n)$ for all $n \geq 3$ and all graphs G.*

Proof. Let $H \simeq \mathsf{P}_2$ if $n = 3$ and $H \simeq \mathsf{C}_{n-1}$ if $n > 3$. We know from Equations (7.3) that $\mathcal{B}(G \cup \mathsf{C}_n) = \mathcal{B}(G \cup \mathsf{P}_n) - \mathcal{B}(G \cup H)$ and $\mathcal{T}(G \cup \mathsf{C}_n) = \mathcal{T}(G \cup \mathsf{P}_n) - \mathcal{T}(G \cup H)$. Since P_n is a proper spanning subgraph of C_n, we have $\mathcal{B}(G \cup \mathsf{C}_n) < \mathcal{B}(G \cup \mathsf{P}_n)$. Altogether, this gives

$$\begin{aligned}
\mathcal{A}(G \cup \mathsf{C}_n) - \mathcal{A}(G \cup H) &= \frac{\mathcal{T}(G \cup \mathsf{C}_n)}{\mathcal{B}(G \cup \mathsf{C}_n)} - \frac{\mathcal{T}(G \cup H)}{\mathcal{B}(G \cup H)} \\
&= \frac{\mathcal{T}(G \cup \mathsf{P}_n) - \mathcal{T}(G \cup H)}{\mathcal{B}(G \cup \mathsf{P}_n) - \mathcal{B}(G \cup H)} - \frac{\mathcal{T}(G \cup H)}{\mathcal{B}(G \cup H)} \\
&= \frac{\mathcal{T}(G \cup \mathsf{P}_n)\mathcal{B}(G \cup H) - \mathcal{T}(G \cup H)\mathcal{B}(G \cup \mathsf{P}_n)}{\mathcal{B}(G \cup \mathsf{C}_n)\mathcal{B}(G \cup H)} \\
&> \frac{\mathcal{T}(G \cup \mathsf{P}_n)\mathcal{B}(G \cup H) - \mathcal{T}(G \cup H)\mathcal{B}(G \cup \mathsf{P}_n)}{\mathcal{B}(G \cup \mathsf{P}_n)\mathcal{B}(G \cup H)} \\
&= \frac{\mathcal{T}(G \cup \mathsf{P}_n)}{\mathcal{B}(G \cup \mathsf{P}_n)} - \frac{\mathcal{T}(G \cup H)}{\mathcal{B}(G \cup H)} \\
&= \mathcal{A}(G \cup \mathsf{P}_n) - \mathcal{A}(G \cup H)
\end{aligned}$$

$\iff \mathcal{A}(G \cup \mathsf{C}_n) > \mathcal{A}(G \cup \mathsf{P}_n)$.

□

For $n \geq 3$, let Q_n be the graph obtained from P_n by adding an edge between an extremity v of P_n and the vertex at distance 2 from v on P_n.

Lemma 29. *If $n \geq 3$, $0 \leq x \leq p$ and $1 \leq k \leq n$, then*

$$S(\mathsf{Q}_n \cup p\mathsf{K}_1, k) = \sum_{i=0}^{x} \binom{x}{i} S(\mathsf{Q}_{n+i} \cup (p-x)\mathsf{K}_1, k).$$

Proof. The result is clearly true for $p = 0$. For larger values of p, we proceed by induction. Since the result is clearly true for $x = 0$, we assume $x \geq 1$. Equations (7.2) imply

$$\begin{aligned}
S(\mathsf{Q}_n \cup p\mathsf{K}_1, k) = \;& S(\mathsf{Q}_{n+1} \cup (p-1)\mathsf{K}_1, k) + S(\mathsf{Q}_n \cup (p-1)\mathsf{K}_1, k) \\
= \;& \sum_{i=0}^{x-1} \binom{x-1}{i} S(\mathsf{Q}_{n+i+1} \cup (p-x)\mathsf{K}_1, k)
\end{aligned}$$

$$
\begin{aligned}
& + \sum_{i=0}^{x-1} \binom{x-1}{i} S(\mathsf{Q}_{n+i} \cup (p-x)\mathsf{K}_1, k) \\
= \quad & \sum_{i=1}^{x} \binom{x-1}{i-1} S(\mathsf{Q}_{n+i} \cup (p-x)\mathsf{K}_1, k) \\
& + \sum_{i=0}^{x-1} \binom{x-1}{i} S(\mathsf{Q}_{n+i} \cup (p-x)\mathsf{K}_1, k) \\
= \quad & S(\mathsf{Q}_{n+x} \cup (p-x)\mathsf{K}_1, k) + S(\mathsf{Q}_n \cup (p-x)\mathsf{K}_1, k) \\
& + \sum_{i=1}^{x-1} \left(\binom{x-1}{i-1} + \binom{x-1}{i} \right) S(\mathsf{Q}_{n+i} \cup (p-x)\mathsf{K}_1, k) \\
= \quad & \sum_{i=0}^{x} \binom{x}{i} S(\mathsf{Q}_{n+i} \cup (p-x)\mathsf{K}_1, k).
\end{aligned}
$$

□

Lemma 30. *If $n \geq 3$ is an odd number and $1 \leq k \leq n$, then*

$$S(\mathsf{C}_n \cup p\mathsf{K}_1, k) = \sum_{i=0}^{(n-3)/2} S(\mathsf{Q}_{2i+3} \cup p\mathsf{K}_1, k).$$

Proof. The result is clearly true for $n = 3$ since $\mathsf{C}_3 \simeq \mathsf{Q}_3$. For larger values of n, we proceed by induction. It follows from Equations (7.1) and (7.2) that

$$
\begin{aligned}
S(\mathsf{C}_n \cup p\mathsf{K}_1, k) = \quad & S(\mathsf{P}_n \cup p\mathsf{K}_1) - S(\mathsf{C}_{n-1} \cup p\mathsf{K}_1, k) \\
= \quad & \Big(S(\mathsf{Q}_n \cup p\mathsf{K}_1, k) + S(\mathsf{P}_{n-1} \cup p\mathsf{K}_1, k) \Big) \\
& - \Big(S(\mathsf{P}_{n-1} \cup p\mathsf{K}_1, k) - S(\mathsf{C}_{n-2} \cup p\mathsf{K}_1, k) \Big) \\
= \quad & S(\mathsf{Q}_n \cup p\mathsf{K}_1, k) + S(\mathsf{C}_{n-2} \cup p\mathsf{K}_1, k) \\
= \quad & S(\mathsf{Q}_n \cup p\mathsf{K}_1, k) + \sum_{i=0}^{(n-5)/2} S(\mathsf{Q}_{2i+3} \cup p\mathsf{K}_1, k) \\
= \quad & \sum_{i=0}^{(n-3)/2} S(\mathsf{Q}_{2i+3} \cup p\mathsf{K}_1, k).
\end{aligned}
$$

□

Lemma 31. *If n and x are two numbers such that $5 \leq x \leq n$ and x is odd, then*

$$S(\mathsf{C}_3 \cup (n-3)\mathsf{K}_1, k) = S(\mathsf{C}_x \cup (n-x)\mathsf{K}_1, k) + \sum_{i=0}^{x-5} \alpha_i S(\mathsf{Q}_{i+4} \cup (n-x)\mathsf{K}_1, k)$$

where

$$\alpha_i = \begin{cases} \binom{x-3}{i+1} - 1 & \text{if } i \text{ is even} \\ \binom{x-3}{i+1} & \text{if } i \text{ is odd.} \end{cases}$$

Proof. Since $\mathsf{Q}_3 \simeq \mathsf{C}_3$ we know from Lemma 29 that

$$\begin{aligned} S(\mathsf{Q}_3 \cup (n-3)\mathsf{K}_1, k) &= \sum_{i=0}^{x-3} \binom{x-3}{i} S(\mathsf{Q}_{i+3} \cup (n-x)\mathsf{K}_1, k) \\ &= \sum_{i=0}^{(x-3)/2} \binom{x-3}{2i} S(\mathsf{Q}_{2i+3} \cup (n-x)\mathsf{K}_1, k) \\ &\quad + \sum_{i=0}^{(x-5)/2} \binom{x-3}{2i+1} S(\mathsf{Q}_{2i+4} \cup (n-x)\mathsf{K}_1, k). \end{aligned}$$

It then follows from Lemma 30 that

$$\begin{aligned} S(\mathsf{Q}_3 \cup (n-3)\mathsf{K}_1, k) &= S(\mathsf{C}_x \cup (n-x)\mathsf{K}_1, k) \\ &\quad + \sum_{i=1}^{(x-5)/2} \left(\binom{x-3}{2i} - 1 \right) S(\mathsf{Q}_{2i+3} \cup (n-x)\mathsf{K}_1, k) \\ &\quad + \sum_{i=0}^{(x-5)/2} \binom{x-3}{2i+1} S(\mathsf{Q}_{2i+4} \cup (n-x)\mathsf{K}_1, k) \\ &= S(\mathsf{C}_x \cup (n-x)\mathsf{K}_1, k) + \sum_{i=0}^{x-5} \alpha_i S(\mathsf{Q}_{i+4} \cup (n-x)\mathsf{K}_1, k). \end{aligned}$$

□

Lemma 32. *If $n \geq 5$ and $3 \leq i < n$ then*

$$\mathcal{A}(\mathsf{Q}_i \cup p\mathsf{K}_1) < \mathcal{A}(\mathsf{C}_n \cup p\mathsf{K}_1)$$

Proof. Since $\mathsf{Q}_{n-1} \cup p\mathsf{K}_1$ is obtained from $\mathsf{Q}_i \cup p\mathsf{K}_1$ by iteratively adding vertices of degree 1, we know from Corollary 13 that $\mathcal{A}(\mathsf{Q}_i \cup p\mathsf{K}_1) \leq \mathcal{A}(\mathsf{Q}_{n-1} \cup p\mathsf{K}_1)$. Moreover, it is proved in [56] that $\mathcal{A}(\mathsf{Q}_{n-1} \cup p\mathsf{K}_1) < \mathcal{A}(\mathsf{P}_n \cup p\mathsf{K}_1)$ for all $n \geq 5$ and $p \geq 0$. It then follows from Lemma 28 that

$$\mathcal{A}(\mathsf{Q}_i \cup p\mathsf{K}_1) \leq \mathcal{A}(\mathsf{Q}_{n-1} \cup p\mathsf{K}_1) < \mathcal{A}(\mathsf{P}_n \cup p\mathsf{K}_1) < \mathcal{A}(\mathsf{C}_n \cup p\mathsf{K}_1).$$

□

Corollary 33. *If $n \geq 5$, x is odd and $5 \leq x \leq n$, then*

$$\mathcal{A}(\mathsf{C}_3 \cup (n-3)\,\mathsf{K}_1) < \mathcal{A}(\mathsf{C}_x \cup (n-x)\,\mathsf{K}_1).$$

Proof. Lemma 31 implies

- $\mathcal{B}(\mathsf{C}_3 \cup (n-3)\mathsf{K}_1) = \mathcal{B}(\mathsf{C}_x \cup (n-x)\mathsf{K}_1) + \sum_{i=0}^{x-5} \alpha_i \mathcal{B}(\mathsf{Q}_{i+4} \cup (n-x)\mathsf{K}_1)$, and
- $\mathcal{T}(\mathsf{C}_3 \cup (n-3)\mathsf{K}_1) = \mathcal{T}(\mathsf{C}_x \cup (n-x)\mathsf{K}_1) + \sum_{i=0}^{x-5} \alpha_i \mathcal{T}(\mathsf{Q}_{i+4} \cup (n-x)\mathsf{K}_1)$,

where

- $\alpha_i = \binom{x-3}{i+1} - 1 \geq 0$ if i is even, and
- $\alpha_i = \binom{x-3}{i+1} > 0$ if i is odd.

Also, we know from Lemma 32 that $\mathcal{A}(\mathsf{Q}_{i+4} \cup p\mathsf{K}_1) < \mathcal{A}(\mathsf{C}_x \cup (n-x)\,\mathsf{K}_1)$ for $i = 0, \ldots, x-5$. Hence, it follows from Theorem 18 that $\mathcal{A}(\mathsf{C}_3 \cup (n-3)\,\mathsf{K}_1) < \mathcal{A}(\mathsf{C}_x \cup (n-x)\,\mathsf{K}_1)$.

□

We are now ready to prove the validity of the two Conjectures 23 and 24 when $\Delta(G) = 2$.

Theorem 34. *Let G be a graph of order n with $\Delta(G) = 2$. Then,*

$$\mathcal{A}(G) \geq L_2(n, \chi(G))$$

with equality if and only if $G \simeq \mathsf{K}_{\chi(G)} \cup (n-\chi(G))\mathsf{K}_1$.

Proof. Since $\Delta(G) = 2$, G is the disjoint union of paths and cycles. If G does not contain any odd cycle, then $\chi(G) = 2$. It then follows from Theorem 15 and Lemma 28 that the edges of G can be removed sequentially, with a strict decrease of $\mathcal{A}(G)$ at each step, until we get $\mathsf{K}_2 \cup (n-2)\mathsf{K}_1$.

If $\chi(G)$=3, then at least one connected component of G is an odd cycle C_x with $x \leq n$. Again, we know from Theorem 15 and Lemma 28 that the edges of G can be removed sequentially, with a strict decrease of $\mathcal{A}(G)$ at each step, until we get $\mathsf{C}_x \cup (n-x)\mathsf{K}_1$. It then follows from Corollary 33 that $\mathcal{A}(G) \geq \mathcal{A}(\mathsf{C}_x \cup (n-x)\,\mathsf{K}_1) \geq \mathcal{A}(\mathsf{C}_3 \cup (n-3)\,\mathsf{K}_1)$, with equalities if and only if $G \simeq \mathsf{C}_3 \cup (n-3)\mathsf{K}_1 \simeq \mathsf{K}_3 \cup (n-3)\mathsf{K}_1$.

□

Theorem 35. *Let G be a graph of order n with $\Delta(G) = 2$. Then,*

$$\mathcal{A}(G) \geq L_3(n, \Delta(G) + 1)$$

with equality if and only if $G \simeq \mathsf{K}_{1,2} \cup (n-3)\mathsf{K}_1$.

Proof. Since $\Delta(G) = 2$, G is the disjoint union of paths and cycles. Also, G contains at least one vertex u of degree 2. Let v and w be two neighbors of u in G. It follows from Theorem 15 and Lemma 28 that the edges of G can be removed sequentially, with a strict decrease of $\mathcal{A}(G)$ at each step, until the edge set of the remaining graph H is $\{(u, v), (u, w)\}$. But H is then isomorphic to $\mathsf{K}_{1,2} \cup (n-3)\mathsf{K}_1$. □

7.6 Upper bounds on the average number of colors in the non-equivalent colorings of a graph

Lower bounds on $\mathcal{A}(G)$ are studied in the previous section, and we saw that there is no known lower bound on $\mathcal{A}(G)$ which is a function of n and such that there exists at least one graph of order n which reaches it. As we will show, the situation is not the same for the upper bound. Indeed, we show that there is an upper bound on $\mathcal{A}(G)$ which is a function of n and such that

there exists at least one graph of order n which reaches it. We also give a sharper upper bound for graphs with maximum degree $\Delta(G) \in \{1, 2, n-2\}$.

The next subsection is devoted to properties of $\mathcal{A}(G)$ and basic ingredients that we will use in Section 7.6.2 for proving the validity of the upper bounds on $\mathcal{A}(G)$.

7.6.1 Useful properties for the upper bound

Some graphs G of order $n \leq 9$ will play a special role in the next subsection. The values $S(G, k)$ of these graphs, with $2 \leq k \leq n$, are given in Table 7.1. These values lead to the following lemma.

Lemma 36. *The following strict inequalities are valid for all graphs G:*

$$
\begin{aligned}
&(a)\ \mathcal{A}(G \cup \mathsf{C}_6) < \mathcal{A}(G \cup 2\mathsf{C}_3)\\
&(b)\ \mathcal{A}(G \cup \mathsf{C}_7) < \mathcal{A}(G \cup \mathsf{C}_3 \cup \mathsf{C}_4)\\
&(c)\ \mathcal{A}(G \cup \mathsf{C}_8) < \mathcal{A}(G \cup \mathsf{C}_3 \cup \mathsf{C}_5)\\
&(d)\ \mathcal{A}(G \cup \mathsf{C}_3 \cup \mathsf{K}_2) < \mathcal{A}(G \cup \mathsf{C}_5)\\
&(e)\ \mathcal{A}(G \cup \mathsf{C}_4 \cup \mathsf{K}_2) < \mathcal{A}(G \cup 2\mathsf{C}_3)\\
&(f)\ \mathcal{A}(G \cup \mathsf{C}_5 \cup \mathsf{K}_2) < \mathcal{A}(G \cup \mathsf{C}_3 \cup \mathsf{C}_4)\\
&(g)\ \mathcal{A}(G \cup \mathsf{C}_4 \cup \mathsf{K}_1) < \mathcal{A}(G \cup \mathsf{C}_5)\\
&(h)\ \mathcal{A}(G \cup \mathsf{C}_5 \cup \mathsf{K}_1) < \mathcal{A}(G \cup 2\mathsf{C}_3)\\
&(i)\ \mathcal{A}(G \cup 2\mathsf{C}_4) < \mathcal{A}(G \cup \mathsf{C}_3 \cup \mathsf{C}_5)\\
&(j)\ \mathcal{A}(G \cup \mathsf{C}_4 \cup \mathsf{C}_5) < \mathcal{A}(G \cup 3\mathsf{C}_3)\\
&(k)\ \mathcal{A}(G \cup 2\mathsf{C}_5) < \mathcal{A}(G \cup 2\mathsf{C}_3 \cup \mathsf{C}_4).
\end{aligned}
$$

Proof. All these inequalities can be obtained from Theorem 17 by using the values given in Table 7.1. For example, to check that (a) holds, the 4^{th} and 7^{th} lines of Table 7.1 allow checking that $S(2\mathsf{C}_3, k)S(\mathsf{C}_6, k') - S(\mathsf{C}_6, k)S(2\mathsf{C}_3, k') \geq 0$ for all $k > k'$ and at least one of these values is strictly positive. □

Table 7.1: Values of $S(G,k)$ for some graphs G of order n and $2 \leq k \leq n$

k	2	3	4	5	6	7	8	9	10
$S(\mathsf{C}_3 \cup \mathsf{K}_2, k)$	0	6	6	1					
$S(\mathsf{C}_4 \cup \mathsf{K}_1, k)$	2	7	6	1					
$S(\mathsf{C}_5, k)$	0	5	5	1					
$S(2\mathsf{C}_3, k)$	0	6	18	9	1				
$S(\mathsf{C}_4 \cup \mathsf{K}_2, k)$	2	16	25	10	1				
$S(\mathsf{C}_5 \cup \mathsf{K}_1, k)$	0	15	25	10	1				
$S(\mathsf{C}_6, k)$	1	10	20	9	1				
$S(\mathsf{C}_3 \cup \mathsf{C}_4, k)$	0	18	66	55	14	1			
$S(\mathsf{C}_5 \cup \mathsf{K}_2, k)$	0	30	90	65	15	1			
$S(\mathsf{C}_7, k)$	0	21	70	56	14	1			
$S(\mathsf{C}_3 \cup \mathsf{C}_5, k)$	0	30	210	285	125	20	1		
$S(2\mathsf{C}_4, k)$	2	52	241	296	126	20	1		
$S(\mathsf{C}_8, k)$	1	42	231	294	126	20	1		
$S(3\mathsf{C}_3, k)$	0	36	540	1242	882	243	27	1	
$S(\mathsf{C}_3 \cup \mathsf{C}_6, k)$	0	66	666	1351	910	245	27	1	
$S(\mathsf{C}_4 \cup \mathsf{C}_5, k)$	0	90	750	1415	925	246	27	1	
$S(\mathsf{C}_9, k)$	0	85	735	1407	924	246	27	1	
$S(2\mathsf{C}_3 \cup \mathsf{C}_4, k)$	0	108	1908	5838	5790	2361	433	35	1
$S(2\mathsf{C}_5, k)$	0	150	2250	6345	6025	2400	435	35	1

Those inequalities also show us which transformation replacing a subgraph can strictly increase $\mathcal{A}(G)$.

We now show the validity of four lemmas which will be helpful for proving that $\mathcal{A}(G \cup \mathsf{C}_n) < \mathcal{A}(G \cup \mathsf{C}_{n-3} \cup \mathsf{C}_3)$ for all $n \geq 6$. A direct consequence of this result will be that a graph G that maximizes $\mathcal{A}(G)$ among the graphs with maximum degree 2 cannot contain an induced C_n with $n \geq 6$.

Lemma 37. *$S(\mathsf{C}_n, k) = (k-1)S(\mathsf{C}_{n-1}, k) + S(\mathsf{C}_{n-1}, k-1)$ for all $n \geq 4$ and*

all $k \geq 3$.

Proof. The values in the following table show that the result is true for $n = 4$.

k	2	3	4
$S(\mathsf{C}_4, k)$	1	2	1
$S(\mathsf{C}_3, k)$	0	1	0

For larger values of n, we proceed by induction. So assume $n \geq 5$, let u be a vertex in C_n, and let v and w be its two neighbors in C_n. Let us analyze the set of non-equivalent colorings of C_n that use exactly k colors:

- there are $(k-1)S(\mathsf{C}_{n-2}, k)$ such colorings where v and w have the same color and at least one vertex of $\mathsf{C}_n - u$ has the same color as u;
- there are $S(\mathsf{C}_{n-2}, k-1)$ such colorings where v and w have the same color and no vertex on $\mathsf{C}_n - u$ has the same color as u;
- there are $(k-2)S(\mathsf{C}_{n-1}, k)$ such colorings where v and w have different colors and at least one vertex of $\mathsf{C}_n - u$ has the same color as u;
- there are $S(\mathsf{C}_{n-1}, k-1)$ such colorings where v and w have different colors and no vertex on $\mathsf{C}_n - u$ has the same color as u.

Hence,

$$\begin{aligned} S(\mathsf{C}_n, k) = & \Big((k-1)S(\mathsf{C}_{n-2}, k) + S(\mathsf{C}_{n-2}, k-1)\Big) \\ & + (k-2)S(\mathsf{C}_{n-1}, k) + S(\mathsf{C}_{n-1}, k-1) \\ = & \; S(\mathsf{C}_{n-1}, k) + (k-2)S(\mathsf{C}_{n-1}, k) + S(\mathsf{C}_{n-1}, k-1) \\ = & \; (k-1)S(\mathsf{C}_{n-1}, k) + S(\mathsf{C}_{n-1}, k-1). \end{aligned}$$

□

Lemma 38. *If* $n \geq 7$ *and* $k \leq n$ *then*

$$S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k) = (k-1)S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) + S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k-1) - (-1)^n \delta_k$$

where

$$\delta_k = \begin{cases} 6 & \text{if } k = 3, 4, \\ 1 & \text{if } k = 5, \\ 0 & \text{otherwise .} \end{cases}$$

Proof. The values in the following table show that the result is true for $n = 7$.

k	2	3	4	5	6	7
$S(\mathsf{C}_4 \cup \mathsf{C}_3, k)$	0	18	66	55	14	1
$S(\mathsf{C}_3 \cup \mathsf{C}_3, k)$	0	6	18	9	1	0

For larger values of n, we proceed by induction. Let u be a vertex in C_{n-3}, and let v and w be its two neighbors. We analyze the set of non-equivalent colorings of $\mathsf{C}_{n-3} \cup \mathsf{C}_3$ that use exactly k colors:

- there are $(k-1)S(\mathsf{C}_{n-5} \cup \mathsf{C}_3, k)$ such colorings where v and w have the same color and at least one vertex of $\mathsf{C}_{n-3} \cup \mathsf{C}_3 - u$ has the same color as u;
- there are $S(\mathsf{C}_{n-5} \cup \mathsf{C}_3, k-1)$ such colorings where v and w have the same color and no vertex on $\mathsf{C}_{n-3} \cup \mathsf{C}_3 - u$ has the same color as u;
- there are $(k-2)S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k)$ such colorings where v and w have different colors and at least one vertex of $\mathsf{C}_{n-3} \cup \mathsf{C}_3 - u$ has the same color as u;
- there are $S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k-1)$ such colorings where v and w have different colors and no vertex on $\mathsf{C}_{n-3} \cup \mathsf{C}_3 - u$ has the same color as u.

Hence,

$$\begin{aligned}
S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k) =& \Big((k-1)S(\mathsf{C}_{n-5} \cup \mathsf{C}_3, k) + S(\mathsf{C}_{n-5} \cup \mathsf{C}_3, k-1)\Big) \\
&+ (k-2)S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) + S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k-1) \\
=& \Big(S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) + (-1)^{n-1}\delta_k\Big) \\
&+ (k-2)S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) + S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k-1) \\
=& (k-1)S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) + S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k-1) - (-1)^n\delta_k.
\end{aligned}$$

□

For $n \geq 3$, let Q_n be the graph obtained from P_n by adding an edge between an extremity v of P_n and the vertex at distance 2 from v on P_n.

Lemma 39. *If $n \geq 6$ and $k \leq n$ then $S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k) = S(\mathsf{Q}_n, k) - (-1)^n \rho_k$ where*

$$\rho_k = \begin{cases} 2 & \text{if } k = 3, \\ 1 & \text{if } k = 4, \\ 0 & \text{otherwise .} \end{cases}$$

Proof. The values in the following table show that the result is true for $n = 6$.

k	2	3	4	5	6
$S(2\mathsf{C}_3, k)$	0	6	18	9	1
$S(\mathsf{Q}_6, k)$	0	8	19	9	1

For larger values of n, we proceed by induction. Equations (7.1) and (7.2) give

$$\begin{aligned} S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k) = \quad & S(\mathsf{P}_{n-3} \cup \mathsf{C}_3, k) - S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) \\ = \quad & S(\mathsf{P}_{n-3} \cup \mathsf{P}_3, k) - S(\mathsf{P}_{n-3} \cup \mathsf{P}_2, k) - S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) \\ = \quad & S(\mathsf{P}_n, k) + S(\mathsf{P}_{n-1}, k) - S(\mathsf{P}_{n-1}, k) \\ & - S(\mathsf{P}_{n-2}, k) - S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) \\ = \quad & S(\mathsf{Q}_n, k) + S(\mathsf{Q}_{n-1}, k) - S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, k) \\ = \quad & S(\mathsf{Q}_n, k) + (-1)^{n-1} \rho_k \\ = \quad & S(\mathsf{Q}_n, k) - (-1)^n \rho_k \end{aligned}$$

□

Lemma 40. *The following inequalities are valid for all $n \geq 9$:*

(a) $S(\mathsf{C}_n, k) > S(\mathsf{C}_n, k-1)$ *for all* $k \in \{3, 4, 5\}$;

(b) $S(\mathsf{C}_n, k) > 3S(\mathsf{C}_{n-1}, k-1)$ *for all* $k \in \{3, 4, 5, 6\}$;

(c) $S(\mathsf{C}_n, 4) > 8S(\mathsf{C}_n, 3)$.

Proof. The values in Table 7.1 show that the inequalities are satisfied for $n = 9$. For larger values of n, we proceed by induction. Note that (a) and (b) are clearly valid for $k = 3$ since $S(\mathsf{C}_n, 3) > 3 \geq \max\{S(\mathsf{C}_n, 2)), 3S(\mathsf{C}_{n-1}, 2)\}$.

We may therefore assume $k \in \{4, 5\}$ for (a) and $k \in \{4, 5, 6\}$ for (b). Lemma 37 and the induction hypothesis imply

$$\begin{aligned} S(\mathsf{C}_n, k) =& (k-1)S(\mathsf{C}_{n-1}, k) + S(\mathsf{C}_{n-1}, k-1) \\ >& (k-2)S(\mathsf{C}_{n-1}, k-1) + S(\mathsf{C}_{n-1}, k-2) \\ =& S(\mathsf{C}_n, k-1). \end{aligned}$$

Hence (a) is proved. It follows that the following inequality is valid:

$$\begin{aligned} \frac{1}{k-1}S(\mathsf{C}_{n-1}, k-1) =& \frac{1}{k-1}\Big((k-2)S(\mathsf{C}_{n-2}, k-1) + S(\mathsf{C}_{n-2}, k-2)\Big) \\ <& \frac{1}{k-1}\Big((k-1)S(\mathsf{C}_{n-2}, k-1)\Big) \\ =& S(\mathsf{C}_{n-2}, k-1) \end{aligned}$$

which implies

$$\begin{aligned} S(\mathsf{C}_n, k) =& (k-1)S(\mathsf{C}_{n-1}, k) + S(\mathsf{C}_{n-1}, k-1) \\ >& (k-1)S(\mathsf{C}_{n-1}, k) \\ >& 3(k-1)S(\mathsf{C}_{n-2}, k-1) \\ >& 3S(\mathsf{C}_{n-1}, k-1). \end{aligned}$$

Hence (b) is proved. We thus have

$$\begin{aligned} S(\mathsf{C}_n, 4) =& 3S(\mathsf{C}_{n-1}, 4) + S(\mathsf{C}_{n-1}, 3) \\ >& 25S(\mathsf{C}_{n-1}, 3) \\ >& \frac{25}{3}S(\mathsf{C}_n, 3) \\ >& 8S(\mathsf{C}_n, 3). \end{aligned}$$

which proves (c). □

7.6.2 Upper bounds on $\mathcal{A}(G)$

We are now ready to give upper bounds on $\mathcal{A}(G)$. The following theorem gives a general upper bound on $\mathcal{A}(G)$ that is valid for all graphs G of order n.

Theorem 41. *Let G be a graph of order n, then,*

$$\mathcal{A}(G) \leq n,$$

with equality if and only if $G \simeq \mathsf{K}_n$.

Proof. Clearly,

$$\mathcal{T}(G) = \sum_{k=1}^{n} kS(G,k) \leq n \sum_{k=1}^{n} S(G,k) = n\mathcal{B}(G).$$

Hence, $\mathcal{A}(G) \leq n$, with equality if and only if $S(G,k) = 0$ for all $k < n$, that is if $G \simeq \mathsf{K}_n$. □

Since $\Delta(\mathsf{K}_n) = n - 1$ we immediately get the following corollary to Theorem 41.

Corollary 42. *Let G be a graph of order n and maximum degree $\Delta(G) = n-1$. Then, $\mathcal{A}(G) \leq n$, with equality if and only if $G \simeq \mathsf{K}_n$.*

We now give a more precise upper bound on $\mathcal{A}(G)$ for graphs G of order n and maximum degree $\Delta(G) = n - 2$.

Theorem 43. *Let G be a graph of order $n \geq 2$ and maximum degree $\Delta(G) = n - 2$. Then,*

$$\mathcal{A}(G) \leq \frac{n^2 - n + 1}{n},$$

with equality if and only if $G \simeq \mathsf{K}_{n-1} \cup \mathsf{K}_1$.

Proof. Let m be the number of edges in G, and let $x = \frac{n(n-1)}{2} - m = S(G, n-1)$. Then

$$\begin{aligned}
\mathcal{T}(G) &= \sum_{k=1}^{n-2} kS(G,k) + x(n-1) + n \\
&\leq (n-1) \sum_{k=1}^{n-2} S(G,k) + x(n-1) + n \\
&= (n-1) \sum_{k=1}^{n} S(G,k) + 1
\end{aligned}$$

$$= (n-1)\mathcal{B}(G) + 1.$$

Hence, $\mathcal{A}(G) \leq n - 1 + \frac{1}{\mathcal{B}(G)}$, with possible equality only if $S(G,k) = 0$ for all $k < n-1$. It is proved in [58] that $\mathcal{B}(G) \geq n$, with equality if and only if G is isomorphic to $\mathsf{K}_{n-1} \cup \mathsf{K}_1$ when $n \neq 4$, and G is isomorphic to $\mathsf{K}_3 \cup \mathsf{K}_1$ or C_4 when $n = 4$. Since $S(\mathsf{C}_4, 2) = 1 > 0$ while $S(\mathsf{K}_{n-1} \cup \mathsf{K}_1, k) = 0$ for all $k < n-1$, we conclude that $\mathcal{A}(G) \leq n - 1 + \frac{1}{n} = \frac{n^2-n+1}{n}$, with equality if and only if $G \simeq \mathsf{K}_{n-1} \cup \mathsf{K}_1$.

□

The next simple case is when $\Delta(G) = 1$.

Theorem 44. *Let G be a graph of order n and maximum degree $\Delta(G) = 1$. Then,*

$$\mathcal{A}(G) \leq \mathcal{A}(\left\lfloor \frac{n}{2} \right\rfloor \mathsf{K}_2 \cup (n \bmod 2)\mathsf{K}_1)$$

with equality if and only if $G \simeq \lfloor \frac{n}{2} \rfloor \mathsf{K}_2 \cup (n \bmod 2)\mathsf{K}_1$.

Proof. If G contains two isolated vertices u and v, we know from Theorem 15 that $\mathcal{A}(G + (u,v)) > \mathcal{A}(G)$. Hence, the maximum value of $\mathcal{A}(G)$ is reached when G contains at most one isolated vertex, that is $G \simeq \lfloor \frac{n}{2} \rfloor \mathsf{K}_2 \cup (n \bmod 2)\mathsf{K}_1$. □

We now give a precise upper bound on $\mathcal{A}(G)$ for graphs G with maximum degree 2. Such graphs are composed of a disjoint union of cycles and paths. We will show the effects of transformations replacing connected components by different subgraphs and use those to prove our bound.

We first analyze the impact of the replacement of an induced C_n ($n \geq 6$) by $\mathsf{C}_{n-3} \cup \mathsf{C}_3$.

Lemma 45. *$\mathcal{A}(G \cup \mathsf{C}_n) < \mathcal{A}(G \cup \mathsf{C}_{n-3} \cup \mathsf{C}_3)$ for all $n \geq 6$ and all graphs G.*

Proof. We know from Lemma 36 (a), (b) and (c) that the result is true for $n = 6, 7, 8$. We can therefore assume $n \geq 9$.

Let $f_n(k,k') = S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k)S(\mathsf{C}_n, k') - S(\mathsf{C}_n, k)S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k')$. Theorem 17 shows that it is sufficient to prove that $f_n(k,k') \geq 0$ for all $k > k'$, the

inequality being strict for at least one pair (k, k'). Note that $f_n(n, 2) = 1 > 0$ for n even. Also, $f_n(n, 3) > 0$ for n odd. Indeed, this is true for $n = 9$ since the values in Table 7.1 give $f_n(9, 3) = 85 - 66 = 19$. For larger odd values of n, we proceed by induction, using Lemmas 37 and 38:

$$\begin{aligned} f_n(n,3) &= S(\mathsf{C}_n, 3) - S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, 3) \\ &= \Big(2S(\mathsf{C}_{n-1}, 3) + 1\Big) - \Big(2S(\mathsf{C}_{n-4} \cup \mathsf{C}_3, 3) + 6\Big) \\ &= \Big(4S(\mathsf{C}_{n-2}, 3) + 1\Big) - \Big(2\big(2S(\mathsf{C}_{n-5} \cup \mathsf{C}_3, 3) - 6\big) + 6\Big) \\ &= 4S(\mathsf{C}_{n-2}, 3) - 4S(\mathsf{C}_{n-5} \cup \mathsf{C}_3, 3) + 7 \\ &= 4f_{n-2}(n-2, 3) + 7 > 0. \end{aligned}$$

Hence, it remains to prove that $f_n(k, k') \geq 0$ for all $1 \leq k' < k' \leq n$. Let us start with the cases where $k' \leq 2$ and/or $k \geq n - 1$.

- If $k' \leq 2$ then $f_n(k, k') = S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k)S(\mathsf{C}_n, k') \geq 0$.
- If $k \geq n - 1$ then $S(\mathsf{C}_n, k) = S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k)$ since
  - $S(\mathsf{C}_n, n) = S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, n) = 1$, and
  - $S(\mathsf{C}_n, n-1) = S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, n-1) = \frac{n^2-3n}{2}$.

  Also, it follows from Lemma 39 that $S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k') = S(\mathsf{Q}_n, k') - (-1)^n \rho_{k'}$ and Equations (7.1) and (7.2) give

$$\begin{aligned} S(\mathsf{C}_n, k) &= S(\mathsf{P}_n, k) - S(\mathsf{C}_{n-1}, k) \\ &= (S(\mathsf{Q}_n, k) + S(\mathsf{P}_{n-1}, k)) - (S(\mathsf{P}_{n-1}, k) - S(\mathsf{C}_{n-2}, k)) \\ &= S(\mathsf{Q}_n, k) + S(\mathsf{C}_{n-2}, k). \end{aligned}$$

  Altogether, this gives

$$\begin{aligned} f_n(k, k') = \quad & S(\mathsf{C}_n, k)\Big(S(\mathsf{C}_n, k') - S(\mathsf{C}_{n-3} \cup \mathsf{C}_3, k')\Big) \\ = \quad & S(\mathsf{C}_n, k)\Big((S(\mathsf{Q}_n, k') + S(\mathsf{C}_{n-2}, k')) \\ & - \big(S(\mathsf{Q}_n, k') - (-1)^n \rho_{k'}\big)\Big) \end{aligned}$$

$$= \quad S(\mathsf{C}_n,k)\Big(S(\mathsf{C}_{n-2},k')+(-1)^n\rho_{k'}\Big).$$

Hence,

- if n is even and/or $k'\notin\{3,4\}$, then $f_n(k,k')\geq 0$;
- if n is odd and $k'=3$ then $f_n(k,k')=S(\mathsf{C}_n,k)(S(\mathsf{C}_{n-2},3)-2)\geq 0$;
- If n is odd and $k'=4$ then $f_n(k,k')=S(\mathsf{C}_n,k)(S(\mathsf{C}_{n-2},4)-1)\geq 0$.

We can therefore assume $3\leq k'<k\leq n-2$ and we finally prove that

$$f_n(k,k')\geq\begin{cases}0 & \text{if } k'\geq 6,\\ 7S(\mathsf{C}_n,k) & \text{if } k'\in\{3,4,5\}.\end{cases}$$

The values in the following table, computed with the help of those for C_9 and $\mathsf{C}_6\cup\mathsf{C}_3$ in Table 7.1, show that this is true for $n=9$:

(k,k')	(4,3)	(5,3)	(5,4)	(6,3)	(6,4)	(6,5)	(7,3)	(7,4)
$f_9(k,k')$	8100	21973	55923	16366	53466	32046	4589	16239
$7S(\mathsf{C}_9,k)$	5145	9849	9849	6468	6468	6468	1722	1722

(k,k')	(7,5)	(7,6)
$f_9(k,k')$	12369	2520
$7S(\mathsf{C}_9,k)$	1722	1722

For larger values of n, we proceed by induction. Lemmas 37 and 38 give

$$\begin{aligned}
f_n(k,k') =& S(\mathsf{C}_n,k')S(\mathsf{C}_{n-3}\cup\mathsf{C}_3,k)-S(\mathsf{C}_n,k)S(\mathsf{C}_{n-3}\cup\mathsf{C}_3,k')\\
=& S(\mathsf{C}_n,k')\Big((k-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k)+S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k-1)-(-1)^n\delta_k\Big)\\
& -S(\mathsf{C}_n,k)\Big((k'-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k')+S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k'-1)-(-1)^n\delta_{k'}\Big)\\
=& \Big((k'-1)S(\mathsf{C}_{n-1},k')+S(\mathsf{C}_{n-1},k'-1)\Big)\Big((k-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k)+S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k-1)\Big)\\
& -\Big((k-1)S(\mathsf{C}_{n-1},k)+S(\mathsf{C}_{n-1},k-1)\Big)\Big((k'-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k')+S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k'-1)\Big)\\
& +(-1)^n\delta_{k'}S(\mathsf{C}_n,k)-(-1)^n\delta_kS(\mathsf{C}_n,k')\\
=& (k-1)(k'-1)\Big(S(\mathsf{C}_{n-1},k')S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k)-S(\mathsf{C}_{n-1},k)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k')\Big)\\
& +(k'-1)\Big(S(\mathsf{C}_{n-1},k')S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k-1)-S(\mathsf{C}_{n-1},k-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k')\Big)
\end{aligned}$$

$$
\begin{aligned}
&+(k-1)\Big(S(\mathsf{C}_{n-1},k'-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k)-S(\mathsf{C}_{n-1},k)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k'-1)\Big)\\
&+S(\mathsf{C}_{n-1},k'-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k-1)-S(\mathsf{C}_{n-1},k-1)S(\mathsf{C}_{n-4}\cup\mathsf{C}_3,k'-1)\\
&+(-1)^n\delta_{k'}S(\mathsf{C}_n,k)-(-1)^n\delta_k S(\mathsf{C}_n,k')\\
=&(k-1)(k'-1)f_{n-1}(k,k')+(k'-1)f_{n-1}(k-1,k')\\
&+(k-1)f_{n-1}(k,k'-1)+f_{n-1}(k-1,k'-1)\\
&+(-1)^n\delta_{k'}S(\mathsf{C}_n,k)-(-1)^n\delta_k S(\mathsf{C}_n,k'). \qquad (7.5)
\end{aligned}
$$

Since $\delta_k = 0$ for $k \geq 6$ and $f_{n-1}(k,k') \geq 0$, $f_{n-1}(k-1,k') \geq 0$, $f_{n-1}(k,k'-1) \geq 0$, and $f_{n-1}(k-1,k'-1) \geq 0$ for $k > k'$, we have $f_n(k,k') \geq 0$ for $k > k' \geq 6$.

Therefore, it remains to show that $f_n(k,k') \geq 7S(\mathsf{C}_n,k)$ for $k' \in \{3,4,5\}$. Let $g_n(k,k') = (-1)^n\delta_{k'}S(\mathsf{C}_n,k) - (-1)^n\delta_k S(\mathsf{C}_n,k')$. There are 4 possible cases.

- *Case 1:* $k' \in \{4,5\}$ *and* $k \geq k'+2$.
  We have $g_n(k,k') = (-1)^n\delta_{k'}S(\mathsf{C}_n,k) \geq -6S(\mathsf{C}_n,k)$. Using the induction hypothesis and Lemma 37, Equation (7.5) gives

$$
\begin{aligned}
f_n(k,k') \geq&\Big(7(k-1)(k'-1)S(\mathsf{C}_{n-1},k)+7(k'-1)S(\mathsf{C}_{n-1},k-1)\Big)\\
&+\Big(7(k-1)S(\mathsf{C}_{n-1},k)+7S(\mathsf{C}_{n-1},k-1)\Big)-6S(\mathsf{C}_n,k)\\
=&7(k'-1)S(\mathsf{C}_n,k)+7S(\mathsf{C}_n,k)-6S(\mathsf{C}_n,k)\\
=&(7k'-6)S(\mathsf{C}_n,k)\\
>&7S(\mathsf{C}_n,k).
\end{aligned}
$$

- *Case 2:* $k' \in \{4,5\}$ *and* $k = k'+1$.
  Let us first give a lower bound on $g_n(k,k')$:

  - if n is even and $k = 6$, then $g_n(k,k') \geq 0$;
  - if n is even and $k = 5$, then $g_n(k,k') \geq -S(\mathsf{C}_n,4)$, and we deduce from Lemma 40 (a) that $g_n(k,k') \geq -S(\mathsf{C}_n,5)$;
  - if n is odd, then $g_n(k,k') \geq -\delta_{k'}S(\mathsf{C}_n,k) \geq -6S(\mathsf{C}_n,k)$.

Hence, whatever n and (k,k'), $g_n(k,k') \geq -6S(\mathsf{C}_n,k)$. Since $f_{n-1}(k-1,k') = 0$, using again the induction hypothesis and Lemma 37, we deduce from Equation (7.5) that

$$\begin{aligned} f_n(k,k') \geq & \Big(7(k-1)(k'-1)S(\mathsf{C}_{n-1},k)\Big) \\ & + \Big(7(k-1)S(\mathsf{C}_{n-1},k) + 7S(\mathsf{C}_{n-1},k-1)\Big) - 6S(\mathsf{C}_n,k) \\ = & \Big(7(k'-1)S(\mathsf{C}_n,k) - 7(k'-1)S(\mathsf{C}_{n-1},k-1)\Big) \\ & + \Big(7S(\mathsf{C}_n,k)\Big) - 6S(\mathsf{C}_n,k) \\ = & (7k'-6)S(\mathsf{C}_n,k) - 7(k'-1)S(\mathsf{C}_{n-1},k-1) \end{aligned}$$

Since $k \leq 6$, Lemma 40 (b) shows that $S(\mathsf{C}_{n-1},k-1) \leq \frac{1}{3}S(\mathsf{C}_n,k)$ and we therefore have

$$\begin{aligned} f_n(k,k') &\geq \left(\frac{14k'-11}{3}\right) S(\mathsf{C}_n,k) \\ &\geq 15S(\mathsf{C}_n,k) \\ &> 7S(\mathsf{C}_n,k). \end{aligned}$$

- *Case 3: $k' = 3$ and $k \geq 5$.*
  As in the previous case, we have $g_n(k,k') \geq -6S(\mathsf{C}_n,k)$. The induction hypothesis gives $f_{n-1}(k,k') \geq 7S(\mathsf{C}_{n-1},k)$, $f_{n-1}(k-1,k') \geq 7S(\mathsf{C}_{n-1},k-1)$, $f_{n-1}(k,k'-1) \geq 0$, and $f_{n-1}(k-1,k'-1) \geq 0$. Hence, Equation (7.5) becomes

$$\begin{aligned} f_n(k,k') \geq & 7(k-1)(k'-1)S(\mathsf{C}_{n-1},k) \\ & + 7(k'-1)S(\mathsf{C}_{n-1},k-1) - 6S(\mathsf{C}_n,k) \\ = & 7(k'-1)S(\mathsf{C}_n,k) - 6S(\mathsf{C}_n,k) \\ = & 8S(\mathsf{C}_n,k) \\ > & 7S(\mathsf{C}_n,k). \end{aligned}$$

- *Case 4: $k' = 3$ and $k = 4$.*
  We have $g_n(k,k') = (-1)^n 6S(\mathsf{C}_n,k) - (-1)^n 6S(\mathsf{C}_n,k')$ and we know from

Lemma 40 (a) that $S(\mathsf{C}_n, 4) > S(\mathsf{C}_n, 3)$. Hence, $g_n(4,3) \geq -6(S(\mathsf{C}_n, k) - S(\mathsf{C}_n, k'))$. Using the induction hypothesis, Equation (7.5) gives

$$\begin{aligned} f_n(k,k') &\geq -7(k-1)(k'-1)S(\mathsf{C}_{n-1}, k) - 6\Big(S(\mathsf{C}_n, k) - S(\mathsf{C}_n, k')\Big) \\ &= 42S(\mathsf{C}_{n-1}, k) - 6\Big(S(\mathsf{C}_n, k) - S(\mathsf{C}_n, k')\Big). \end{aligned}$$

We therefore conclude from Lemmas 37 and 40 (c) that

$$\begin{aligned} f_n(4,3) &\geq \frac{42}{3}\Big(S(\mathsf{C}_n, 4) - S(\mathsf{C}_n, 3\Big) - 6\Big(S(\mathsf{C}_n, 4) - S(\mathsf{C}_n, 3)\Big) \\ &= 8\Big(S(\mathsf{C}_n, 4) - S(\mathsf{C}_n, 3)\Big) \\ &> 8\Big(S(\mathsf{C}_n, 4) - \frac{1}{8}S(\mathsf{C}_n, 4)\Big) \\ &= 7S(\mathsf{C}_n, 4). \end{aligned}$$

□

It is easy to check that

- $\mathcal{A}(\mathsf{K}_3 \cup \mathsf{K}_1) = \frac{13}{4} > 3 = \mathcal{A}(\mathsf{C}_4)$, and
- $\mathcal{A}(2\mathsf{K}_3 \cup \mathsf{K}_1) = \frac{778}{175} > \frac{684}{154} = \mathcal{A}(\mathsf{K}_3 \cup \mathsf{C}_4)$.

Hence, $\mathcal{A}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1) > \mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4)$ for $p = 0, 1$. We next prove that this inequality is reversed for larger values of p, that is $\mathcal{A}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1) < \mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4)$ for $p \geq 2$. Proposition 17 is of no help for this proof since, whatever p, there are pairs (k, k') for which $S((p+1)\mathsf{K}_3 \cup \mathsf{K}_1, k)S(p\mathsf{K}_3 \cup \mathsf{C}_4, k') > S(p\mathsf{K}_3 \cup \mathsf{C}_4, k)S((p+1)\mathsf{K}_3 \cup \mathsf{K}_1, k')$, and other pairs for which the inequality is reversed. Also, it is not true that

$$\mathcal{A}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{A}(p\mathsf{K}_3 \cup \mathsf{K}_1) > \mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4) - \mathcal{A}((p-1)\mathsf{K}_3 \cup \mathsf{C}_4)$$

which would have given a simple proof by induction on p. The only way we have found to prove the desired result is to explicitly calculate $\mathcal{A}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1)$ and $\mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4)$. This is what we do next, with the help of two lemmas.

Lemma 46. *If G is a graph of order n, then*

$$\mathcal{B}(G \cup \mathsf{K}_2) = \sum_{k=1}^{n} (k^2 + k + 1)S(G, k),$$

$$\mathcal{T}(G \cup \mathsf{K}_2) = \sum_{k=1}^{n} (k^3 + k^2 + 3k + 2) S(G, k),$$
$$\mathcal{B}(G \cup \mathsf{K}_3) = \sum_{k=1}^{n} (k^3 + 2k + 1) S(G, k),$$
$$\mathcal{T}(G \cup \mathsf{K}_3) = \sum_{k=1}^{n} (k^4 + 5k^2 + 4k + 3) S(G, k).$$

Proof. As observed in [58],

$$S(G \cup \mathsf{K}_r, k) = \sum_{i=0}^{r} \binom{k-i}{r-i} \binom{r}{i} (r-i)!\ S(G, k-i). \tag{7.6}$$

For $r = 2$, this gives $S(G \cup \mathsf{K}_2, k) = k(k-1)S(G,k) + 2(k-1)S(G,k-1) + S(G,k-2)$. Hence,

$$\begin{aligned}
\mathcal{B}(G \cup \mathsf{K}_2) &= \sum_{k=1}^{n+2} S(G \cup \mathsf{K}_2, k) \\
&= \sum_{k=1}^{n+2} \Big(k(k-1)S(G,k) + 2(k-1)S(G,k-1) + S(G,k-2) \Big) \\
&= \sum_{k=1}^{n} k(k-1)S(G,k) + \sum_{k=1}^{n} 2kS(G,k) + \sum_{k=1}^{n} S(G,k) \\
&= \sum_{k=1}^{n} (k^2 + k + 1) S(G,k)
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{T}(G \cup \mathsf{K}_2) &= \sum_{k=1}^{n+2} kS(G \cup \mathsf{K}_2, k) \\
&= \sum_{k=1}^{n+2} \Big(k^2(k-1)S(G,k) + 2k(k-1)S(G,k-1) + kS(G,k-2) \Big) \\
&= \sum_{k=1}^{n} k^2(k-1)S(G,k) + \sum_{k=1}^{n} 2(k+1)kS(G,k) + \sum_{k=1}^{n} (k+2)S(G,k) \\
&= \sum_{k=1}^{n} (k^3 + k^2 + 3k + 2) S(G,k).
\end{aligned}$$

The values for $\mathcal{B}(G \cup \mathsf{K}_3)$ and $\mathcal{T}(G \cup \mathsf{K}_3)$ are computed in a similar way. □

Lemma 47. *If G is a graph of order n, then,*

$$\begin{aligned}\mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) &= \sum_{k=1}^{n}(k^4+k^3+5k^2+6k+4)S(G,k),\\ \mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) &= \sum_{k=1}^{n}(k^5+k^4+9k^3+15k^2+21k+13)S(G,k).\end{aligned}$$

Proof. Let $G' = G \cup \mathsf{K}_3$. Equation (7.6) gives $S(G' \cup \mathsf{K}_1, k) = kS(G',k) + S(G',k-1)$. Hence, it follows from Lemma 46 that

$$\begin{aligned}\mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) &= \sum_{k=1}^{n+4}\Big(kS(G',k)+S(G',k-1)\Big)\\ &= \sum_{k=1}^{n+3}kS(G',k)+\sum_{k=1}^{n+3}S(G',k)\\ &= \mathcal{T}(G')+\mathcal{B}(G')\\ &= \sum_{k=1}^{n}\Big((k^4+5k^2+4k+3)+(k^3+2k+1)\Big)S(G,k)\\ &= \sum_{k=1}^{n}(k^4+k^3+5k^2+6k+4)S(G,k).\end{aligned}$$

Equation (7.6) gives

$$\begin{aligned}&\sum_{k=1}^{n+3}k^2S(G',k)\\ &= \sum_{k=1}^{n}k^2\Big(k(k-1)(k-2)S(G,k)\Big)+\sum_{k=1}^{n+1}k^2\Big(3(k-1)(k-2)S(G,k-1)\Big)\\ &\quad+\sum_{k=1}^{n+2}k^2\Big(3(k-2)S(G,k-2)\Big)+\sum_{k=1}^{n+3}k^2S(G,k-3)\Big)\\ &= \sum_{k=1}^{n}k^3(k-1)(k-2)S(G,k)+\sum_{k=1}^{n}3(k+1)^2k(k-1)S(G,k)\\ &\quad+\sum_{k=1}^{n}3(k+2)^2kS(G,k)+\sum_{k=1}^{n}(k+3)^2S(G,k)\\ &= \sum_{k=1}^{n}\Big(k^3(k-1)(k-2)+3(k+1)^2k(k-1)+3(k+2)^2k+(k+3)^2\Big)S(G,k)\end{aligned}$$

$$= \sum_{k=1}^{n} (k^5 + 8k^3 + 10k^2 + 15k + 9) S(G,k).$$

Hence, using again Lemma 46, we get

$$\begin{aligned}
\mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) &= \sum_{k=1}^{n+4} \Big(k^2 S(G',k) + k S(G',k-1) \Big) \\
&= \sum_{k=1}^{n+3} k^2 S(G',k) + \sum_{k=1}^{n+3} (k+1) S(G',k) \\
&= \sum_{k=1}^{n+3} k^2 S(G',k) + \mathcal{T}(G') + \mathcal{B}(G') \\
&= \sum_{k=1}^{n} S(G,k) \Big((k^5+8k^3+10k^2+15k+9) + (k^4+5k^2+4k+3) \Big) \\
&\quad + \sum_{k=1}^{n} S(G,k)(k^3+2k+1) \\
&= \sum_{k=1}^{n} (k^5 + k^4 + 9k^3 + 15k^2 + 21k + 13) S(G,k).
\end{aligned}$$

□

We are now ready to compare $\mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4)$ with $\mathcal{A}((p+1)\,\mathsf{K}_3 \cup \mathsf{K}_1)$.

Theorem 48.

$\mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4) < \mathcal{A}((p+1)\,\mathsf{K}_3 \cup \mathsf{K}_1)$ *if* $p = 0, 1$ *and*

$\mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4) > \mathcal{A}((p+1)\,\mathsf{K}_3 \cup \mathsf{K}_1)$ *if* $p \geq 2$.

Proof. We have already mentioned that

- $\mathcal{A}(\mathsf{K}_3 \cup \mathsf{K}_1) = \frac{13}{4} > 3 = \mathcal{A}(\mathsf{C}_4)$, and
- $\mathcal{A}(2\mathsf{K}_3 \cup \mathsf{K}_1) = \frac{778}{175} > \frac{684}{154} = \mathcal{A}(\mathsf{K}_3 \cup \mathsf{C}_4)$.

Hence, it remains to prove that $\mathcal{A}(p\mathsf{K}_3 \cup \mathsf{C}_4) > \mathcal{A}((p+1)\,\mathsf{K}_3 \cup \mathsf{K}_1)$ for all $p \geq 2$. So assume $p \geq 2$ and let

$$f(p) = \mathcal{T}(p\mathsf{K}_3 \cup \mathsf{C}_4)\mathcal{B}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{B}(p\mathsf{K}_3 \cup \mathsf{C}_4)\mathcal{T}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1).$$

Since

$$\mathcal{A}\left(p\mathsf{K}_3 \cup \mathsf{C}_4\right) - \mathcal{A}\left((p+1)\,\mathsf{K}_3 \cup \mathsf{K}_1\right) = \frac{f(p)}{\mathcal{B}(p\mathsf{K}_3 \cup \mathsf{C}_4))\mathcal{B}((p+1)\mathsf{K}_3 \cup \mathsf{K}_1)},$$

we have to prove that $f(p) > 0$. Note that Equations (7.1) and (7.2) give

$$\begin{aligned}
S(G \cup \mathsf{C}_4, k) =& S(G \cup \mathsf{P}_4, k) - S(G \cup \mathsf{K}_3, k) \\
=& S(G \cup \mathsf{Q}_4, k) + S(G \cup \mathsf{P}_3, k) - S(G \cup \mathsf{K}_3, k) \\
=& \Big(S(G \cup \mathsf{K}_3 \cup \mathsf{K}_1, k) - S(G \cup \mathsf{K}_3, k)\Big) \\
&+ \Big(S(G \cup \mathsf{K}_3, k) + S(G \cup \mathsf{K}_2, k)\Big) - S(G \cup \mathsf{K}_3, k) \\
=& S(G \cup \mathsf{K}_3 \cup \mathsf{K}_1, k) - S(G \cup \mathsf{K}_3, k) + S(G \cup \mathsf{K}_2, k),
\end{aligned}$$

which implies

$$\begin{aligned}
\mathcal{B}(G \cup \mathsf{C}_4) =& \mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{B}(G \cup \mathsf{K}_3) + \mathcal{B}(G \cup \mathsf{K}_2), \text{ and} \\
\mathcal{T}(G \cup \mathsf{C}_4) =& \mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{T}(G \cup \mathsf{K}_3) + \mathcal{T}(G \cup \mathsf{K}_2).
\end{aligned}$$

Hence, with $G = p\mathsf{K}_3$, we get

$$\begin{aligned}
f(p) =& \mathcal{T}(G \cup \mathsf{C}_4)\mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1)\mathcal{B}(G \cup \mathsf{C}_4) \\
=& \Big(\mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{T}(G \cup \mathsf{K}_3) + \mathcal{T}(G \cup \mathsf{K}_2)\Big)\mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) \\
&- \mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1)\Big(\mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1) - \mathcal{B}(G \cup \mathsf{K}_3) + \mathcal{B}(G \cup \mathsf{K}_2)\Big) \\
=& \mathcal{B}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1)\Big(\mathcal{T}(G \cup \mathsf{K}_2) - \mathcal{T}(G \cup \mathsf{K}_3)\Big) \\
&- \mathcal{T}(G \cup \mathsf{K}_3 \cup \mathsf{K}_1)\Big(\mathcal{B}(G \cup \mathsf{K}_2) - \mathcal{B}(G \cup \mathsf{K}_3)\Big).
\end{aligned}$$

Since $S(G, k) = 0$ for $k < 3$, we deduce from Lemmas 46 and 47 that

$$\begin{aligned}
f(p) &= \sum_{k=1}^{n} a_k S(G,k) \sum_{k=1}^{n} b_k S(G,k) - \sum_{k=1}^{n} c_k S(G,k) \sum_{k=1}^{n} d_k S(G,k) \\
&= \sum_{k=3}^{n} \sum_{k'=3}^{n} (a_k b_{k'} - c_k d_{k'}) S(G,k) S(G,k') \\
&= \sum_{k=3}^{n} (a_k b_k - c_k d_k) S^2(G,k) \qquad (7.7)
\end{aligned}$$

$$+\sum_{k'=3}^{n-1}\sum_{k=k'+1}^{n}(a_kb_{k'}-c_kd_{k'}+a_{k'}b_k-c_{k'}d_k)S(G,k)S(G,k') \qquad (7.8)$$

where

$$\begin{aligned}
a_k &= k^4+k^3+5k^2+6k+4,\\
b_k &= (k^3+k^2+3k+2)-(k^4+5k^2+4k+3)\\
&= -k^4+k^3-4k^2-k-1,\\
c_k &= k^5+k^4+9k^3+15k^2+21k+13, \text{ and}\\
d_k &= (k^2+k+1)-(k^3+2k+1)\\
&= -k^3+k^2-k.
\end{aligned}$$

It is therefore sufficient to prove that the sums defined at (7.7) and (7.8) are strictly positive.

- Let $g(k)=a_kb_k-c_kd_k=k^6+k^5-5k^4-19k^3-19k^2+3k-4$. It can be checked that $g(k)>0$ for all $k>3$. Note that Equation (7.6) gives

$$\begin{aligned}
S(G,3) =& S(p\mathsf{K}_3,3)=6S((p-1)\mathsf{K}_3,3)\\
<& 18S(p-1)\mathsf{K}_3,3)+24S((p-1)\mathsf{K}_3,4)\\
=& S((p\mathsf{K}_3,4)=S(G,4).
\end{aligned}$$

  Because $g(3)=-112$ and $g(4)=2328$, we have that $g(3)S^2(G,3)+g(4)S^2(G,4)>0$, which implies

$$\begin{aligned}
\sum_{k=3}^{n}(a_kb_k-c_kd_k)S^2(G,k) =& g(3)S^2(G,3)+g(4)S^2(G,4)\\
&+\sum_{k=5}^{n}g(k)S^2(G,k)>0.
\end{aligned}$$

  Hence, the sum in (7.7) is strictly positive.

- Let $h(k',k)=a_kb_{k'}-c_kd_{k'}+a_{k'}b_k-c_{k'}d_k$. By definition of a_k,b_k,c_k and d_k we obtain

$$h(k',k)=\Big(k^3-k^2+k\Big)k'^5$$

$$
\begin{aligned}
&- \Big(2k^4 - k^3 + 10k^2 + 6k + 5\Big)k'^4 \\
&+ \Big(k^5 + k^4 + 20k^3 + 7k^2 + 35k + 16\Big)k'^3 \\
&- \Big(k^5 + 10k^4 - 7k^3 + 70k^2 + 35k + 34\Big)k'^2 \\
&+ \Big(k^5 - 6k^4 + 35k^3 - 35k^2 + 30k + 3\Big)k' \\
&- 5k^4 + 16k^3 - 34k^2 + 3k - 8.
\end{aligned}
$$

Let us make a change of variable. More precisely, we substitute k' by $i+3$ and k by $j+i+4$. Since $k' \geq 3$ and $k \geq k'+1$, we get $i \geq 0$ and $j \geq 0$. It is a tedious but easy exercise to check that with these new variables, $h(k',k) = h(i+3, j+i+4) = h'(i,j)$ with

$$
\begin{aligned}
h'(i,j) = &\Big(j^2 + 2j + 3\Big)i^6 + \Big(3j^3 + 25j^2 + 47j + 63\Big)i^5 \\
&+ \Big(3j^4 + 52j^3 + 243j^2 + 437j + 533\Big)i^4 \\
&+ \Big(j^5 + 37j^4 + 338j^3 + 1154j^2 + 2017j + 2267\Big)i^3 \\
&+ \Big(8j^5 + 161j^4 + 997j^3 + 2713j^2 + 4692j + 4873\Big)i^2 \\
&+ \Big(22j^5 + 290j^4 + 1258j^3 + 2729j^2 + 4784j + 4443\Big)i \\
&+ 21j^5 + 172j^4 + 440j^3 + 575j^2 + 1112j + 602.
\end{aligned}
$$

Since $i \geq 0$, $j \geq 0$, and all coefficients in $h'(i,j)$ are positive, we conclude that $h'(i,j) = h(k',k) > 0$ for $3 \leq k' < k \leq n$.

Hence, the sum $\displaystyle\sum_{k'=3}^{n-1} \sum_{k=k'+1}^{n} h(k',k)S(G,k)S(G,k')$ in (7.8) is strictly positive. □

We are now ready to prove the main result of this subsection. Let U_n ($n \geq 3$) be the following graph.

$$
\mathsf{U}_n = \begin{cases}
\frac{n}{3}\mathsf{K}_3 & \text{if } n \bmod 3 = 0, \text{ and } n \geq 3, \\
\frac{n-1}{3}\mathsf{K}_3 \cup \mathsf{K}_1 & \text{if } n = 4 \text{ or } n = 7, \\
\frac{n-4}{3}\mathsf{K}_3 \cup \mathsf{C}_4 & \text{if } n \bmod 3 = 1, \text{ and } n \geq 10, \\
\frac{n-5}{3}\mathsf{K}_3 \cup \mathsf{C}_5 & \text{if } n \bmod 3 = 2, \text{ and } n \geq 5.
\end{cases}
$$

Theorem 49. *If G is a graph of order $n \geq 3$ and maximum degree $\Delta(G) = 2$, then,*

$$\mathcal{A}(G) \leq \mathcal{A}(\mathsf{U}_n),$$

with equality if and only if $G \simeq \mathsf{U}_n$.

Proof. Since $\Delta(G) = 2$, G is a disjoint union of cycles and paths. Now, suppose that G maximizes $\mathcal{A}$ among all graphs of maximum degree 2. Then at most one connected component of G is a path. Indeed, if $G \simeq G' \cup \mathsf{P}_k \cup \mathsf{P}_{k'}$, then Equations (7.3) and (7.4) give $\mathcal{B}(G' \cup \mathsf{P}_k \cup \mathsf{P}_{k'}) = \mathcal{B}(G' \cup \mathsf{P}_{k+k'}) + \mathcal{B}(G' \cup \mathsf{P}_{k+k'-1})$ and $\mathcal{T}(G' \cup \mathsf{P}_k \cup \mathsf{P}_{k'}) = \mathcal{T}(G' \cup \mathsf{P}_{k+k'}) + \mathcal{T}(G' \cup \mathsf{P}_{k+k'-1})$. Moreover, we know from Theorem 15 that $\mathcal{A}(G' \cup \mathsf{P}_{k+k'-1}) < \mathcal{A}(G' \cup \mathsf{P}_{k+k'})$. Hence, Theorem 18 implies that $\mathcal{A}(G) = \mathcal{A}(G' \cup \mathsf{P}_k \cup \mathsf{P}_{k'}) < \mathcal{A}(G' \cup \mathsf{P}_{k+k'})$. Since $(G' \cup \mathsf{P}_{k+k'})$ is of order n and maximum degree 2, this contradicts the hypothesis that G maximizes $\mathcal{A}$.

We know from Lemma 28 that replacing a path P_k of order $k \geq 3$ by a cycle C_k strictly increases $\mathcal{A}(G)$. Moreover, Lemma 45 shows that replacing a cycle C_k of order $k \geq 6$ by $\mathsf{C}_{k-3} \cup \mathsf{K}_3$ increases $\mathcal{A}(G)$. Hence, G is a disjoint union of copies of K_3, C_4 and C_5 and eventually one path that is either K_1 or K_2.

Considering Lemma 36, we know from (d), (e) and (f) that G does not contain K_2, and from (g)-(k) that at most one connected component of G is not a K_3. Hence, if $n \bmod 3 = 0$ then $G \simeq \frac{n}{3}\mathsf{K}_3$ and if $n \bmod 3 = 2$ then $G \simeq \frac{n-5}{3}\mathsf{K}_3 \cup \mathsf{C}_5$. Finally, Theorem 48 shows that $G \simeq \frac{n-1}{3}\mathsf{K}_3 \cup \mathsf{K}_1$ if $n = 4$ or 7, and $G \simeq \frac{n-4}{3}\mathsf{K}_3 \cup \mathsf{C}_4$ if $n \bmod 3 = 1$ and $n \geq 10$.

□

7.7 Computing $\mathcal{A}(G)$ efficiently

As we talked about in previous chapters, it is interesting to be able to compute $\mathcal{A}(G)$ at least for small graphs, and it would be better to have a fast algorithm. Because computing $\mathcal{A}(G)$ is quite fast once we know $S(G, k)$ for $1 \leq k \leq |G|$, we only need to be able to compute the value of $S(G, k)$ for all

k rapidly. In the rest of this section, we explore the different methods we can use to improve computational time when using the *deletion-contraction* rule introduced in Section 7.3.

7.7.1 Naive algorithms

Given a graph G, the values of $S(G,k)$ for $0 \leq k \leq n$ can be computed using the deletion-contraction rule recursively. As a reminder, the deletion-contraction rule states that, for any pair of distinct vertices u and v in a graph G,

$$S(G,k) = S(G-(u,v),k) - S(G_{|(u,v)},k) \qquad \text{if } (u,v) \in E(G), \tag{7.9}$$

$$S(G,k) = S(G+(u,v),k) + S(G_{|(u,v)},k) \qquad \text{if } (u,v) \notin E(G). \tag{7.10}$$

Using this rule, we can develop Algorithm 7 that will compute $S(G,k)$ recursively by always using Equation 7.10. This algorithm only stops once it reaches a complete graph for which,

$$S(\mathsf{K}_n,k) = \begin{cases} 0 & \text{if } k \neq n, \\ 1 & \text{otherwise.} \end{cases}$$

A similar algorithm can be written that will always use Equation 7.9 and stop when the graph is an empty graph. In this case, $S(\overline{\mathsf{K}}_n,k) = \left\{ {n \atop k} \right\}$ where $\left\{ {n \atop k} \right\}$ is a Stirling number of the second kind.

The worst-case scenario of each of these algorithms is actually the base case of the other one. Indeed, if we always add an edge until we reach a complete graph, the worst case is an empty graph and the other way around for the edge-removals. This can be mitigated by merging both algorithms. Instead of always adding or removing an edge, one can simply use either of the two equations of the deletion-contraction rule depending on whether the graph is closer to a complete graph or an empty graph. More precisely, given a graph G with order n and size m, we remove an edge if $m < \frac{n(n-1)}{4}$ and add an edge otherwise.

Algorithm 7: Algorithm $NaiveSGK(G)$

Input: A graph G with n vertices.

Output: The list of the values of $S(G,k)$ for $1 \leq k \leq n$ sorted by value of k.

```
1  if G ≃ K_n then
2      return a list containing n − 1 0s and a 1 at the end
3  else
4      Choose a non-edge (u, v) in G
5      sgkAdd ← NaiveSGK(G + (u, v))
6      sgkMerge ← NaiveSGK(G_|(u,v))
7      sgk ← an empty list of size n
8      foreach i ∈ {1, ..., n − 1} do
9          sgk[i] ← sgkAdd[i] + sgkMerge[i]
10     end
11     sgk[n] ← sgkAdd[n]
12     return sgk
13 end
```

However, in some cases, this method is still inefficient. For example, if a graph is composed of a disjoint union of complete graphs, we could avoid adding or removing an edge and exploit the fact that computing $S(\mathsf{K}_n, k)$ is easy. We will discuss how to improve our algorithms by exploiting such property in the next subsection.

7.7.2 An improved algorithm

If a graph is not connected, the deletion-contraction rule might require adding all the possible edges between the components or removing all the edges of those components. But, if a graph is not connected, we could compute the values of $S(G,k)$ for every connected components, and they will be smaller than the graph. Let G be a graph with connected components $G_1, \ldots, G_l$

and let G_z^* $(1 \leq z \leq l)$ be the graph composed of the disjoint union of the connected components 1 trough i $\left(G_z^* = \bigcup_{i=1}^{z} G_i\right)$. We have that

$$S\left(G_z^*, k\right) = \sum_{i=1}^{k} \sum_{j=0}^{i} S\left(G_{z-1}^*, i\right) S\left(G_z, k-j\right) \binom{i}{i-j} \binom{k-j}{i-j} (i-j)!. \quad (7.11)$$

And, since $G_l^* = G$, we can compute the values for G_1^* trough G_l^* and obtain $S\left(G, k\right)$.

The same idea can be applied if the complement of the graph is disconnected.

co-connected components

Definition 12. Let G be a graph, the *co-connected components* of G are the connected components of the complement of G $(\overline{G})$. If a graph G has more than one co-connected component, we will say that it is *co-disconnected*.

co-disconnected

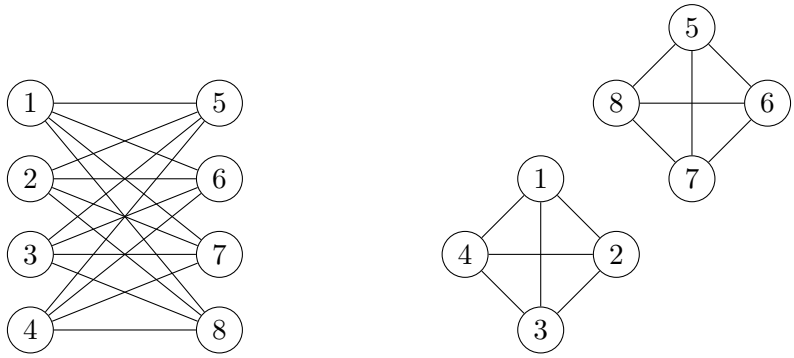
Let G be a co-disconnected graph and let $G_1, \ldots, G_f$ be its co-connected components. Then, $G = G_1 + \cdots + G_f$. We note G_h^+, the graph composed of all the subgraphs G_1 through G_h joined together $(G_h^+ = G_1 + \cdots + G_h)$ for some $1 \leq h \leq f$. Then,

$$S\left(G_h^+, k\right) = \sum_{i=1}^{k-1} S\left(G_{h-1}^+, k-i\right) S\left(G_h^+, i\right). \quad (7.12)$$

And, similarly to the disconnected case, because $G = G_f^+$, we can use this equation to compute $S\left(G, k\right)$.

In Figure 7.4, we can see two graphs where the deletion-contraction rule would be inefficient and using the connected or co-connected components would be faster. The complete bipartite graph on the left is simply a join between two $\overline{\mathsf{K}}_4$ and the values of $S\left(G, k\right)$ can be computed using Equation 7.12. The graph on the right is a union of two K_4 and again, we can compute $S\left(G, k\right)$ using Equation 7.11.

With two special cases, instead of adding or removing arbitrary edges when using the deletion-contraction rule, we can choose edges so that we reach a disconnected or co-disconnected graph by adding or removing as few edges as



(a) A join of two $\overline{\mathsf{K}}_4$. (b) A union of two K_4.

Figure 7.4: Two graphs where there is no need for the deletion-contraction rule.

possible. Such edges are part of a minimum cut, that is, a minimal set of edges of minimum cardinality such that removing them disconnects the graph. Using the edges of a minimum cut would allow us to reach a disconnected graph in a minimal number of steps.

If the graph is closer to being co-disconnected, the same idea can be applied using a minimum cut on the complement of the graph. This minimum cut then contains non-edges in the graph, and adding them transforms it into a co-disconnected graph.

Choosing whether to add or remove an edge when using the deletion-contraction rule then consist in comparing the size of both minimum cuts. If the minimum cut of the graph G is smaller, the graph is closer to a disconnected graph and it is better to remove edges. If, on the other hand, the minimum cut of the complement is smaller than the minimum cut of G, then the graph is closer to being co-disconnected and thus, it is better to add edges. Those minimum cuts can be computed in polynomial time using the Stoer-Wagner algorithm [85].

Adding those optimizations gives Algorithm 8. Because the two cases of adding or removing an edge are very similar, we merged them together. The difference in the algorithm is that the graph G' is either $G+(u,v)$ or $G-(u,v)$

and the variable $factor$ is either 1 or -1.

7.7.3 Performance of the algorithms

In order to compare the naive Algorithm 7 and the alternatives explained in Subsection 7.7.2 and the improved Algorithm 8, we used them to compute the values of $S(G,k)$ for all the graphs up to 9 vertices. The running times are plotted in Figure 7.5. Those times were obtained on a machine using an AMD Ryzen 9 5900X processor with 32 GB of RAM and with the Artix Linux operating system. The first algorithm from top to bottom in the legend is Algorithm 7. The second one uses the same idea always removes an edge when using the deletion-contraction rule until it reaches an empty graph. The third algorithm combines both equations of the deletion-contraction rule depending on the size of the graph. The last algorithm is Algorithm 8. The scale for the execution times is logarithmic. For better comparison, the data is given in Table 7.2.

We can see in Figure 7.5 that Algorithm 8 is the fastest when the order of the graphs increases. Interestingly enough, always adding or removing an edge does not give the same performance. This can be explained by the fact that, when we always remove an edge, the base case of the algorithm is the empty graph. Computing $S\left(\overline{\mathsf{K}}_n, k\right)$ requires computing the Stirling number of the second kind $\left\{{n \atop k}\right\}$ while computing $S(\mathsf{K}_n, k)$ requires simply comparing k with n since a complete graph only accepts one coloring with n colors. Using the size of the graphs to decide whether adding or removing an edge mitigates this to some extent.

7.8 Conclusion

We have established several properties for a recently defined graph invariant, namely the average number $\mathcal{A}(G)$ of colors in the non-equivalent colorings of a graph G. We then looked at bounds for $\mathcal{A}(G)$.

We think that the best possible lower bound on $\mathcal{A}(G)$ for a graph G of order

Algorithm 8: Algorithm *ComplementSGK*(G)

Input: A graph G with n vertices.

Output: The list of the values of $S(G,k)$ for $1 \leq k \leq n$ sorted by value of k.

if $G \simeq \mathsf{K}_n$ **then**
  return a list containing $n-1$ 0s and a 1 at the end
else if $G \simeq \overline{\mathsf{K}}_n$ **then**
  return a list containing $\left\{ {n \atop k} \right\}$ for all $1 \leq k \leq n$
else if G *is not connected* **then**
  Compute $S(G_i,k)$ for all the connected components G_i of G
  Use formula 7.11 to compute $S(G,k)$ for all $1 \leq k \leq n$
else if $\overline{G}$ *is not connected* **then**
  Compute $S(G_i,k)$ for all the co-connected components G_i of G
  Use formula 7.12 to compute $S(G,k)$ for all $1 \leq k \leq n$
else
  $mincut \leftarrow$ a minimum cut of G
  $mincutCompl \leftarrow$ a minimum cut of $\overline{G}$
  if *mincut is smaller than mincutCompl* **then** `// If G is almost disconnected, we will remove an edge.`
    Choose (u,v) in $mincut$
    $G' \leftarrow G - (u,v)$
    $factor \leftarrow -1$
  else `// If G is almost co-disconnected, we will add an edge.`
    Choose a non-edge (u,v) in $mincutCompl$
    $G' \leftarrow G + (u,v)$
    $factor \leftarrow 1$
  end
  $sgkEdge \leftarrow ComplementSGK(G')$
  $sgkMerge \leftarrow ComplementSGK(G_{|(u,v)})$
  $sgk \leftarrow$ an empty list of size n
  foreach $i \in \{1,\ldots,n-1\}$ **do**
    $sgk[i] \leftarrow sgkEdge[i] + factor \cdot sgkMerge[i]$
  end
  $sgk[n] \leftarrow sgkEdge[n]$
  return sgk
end

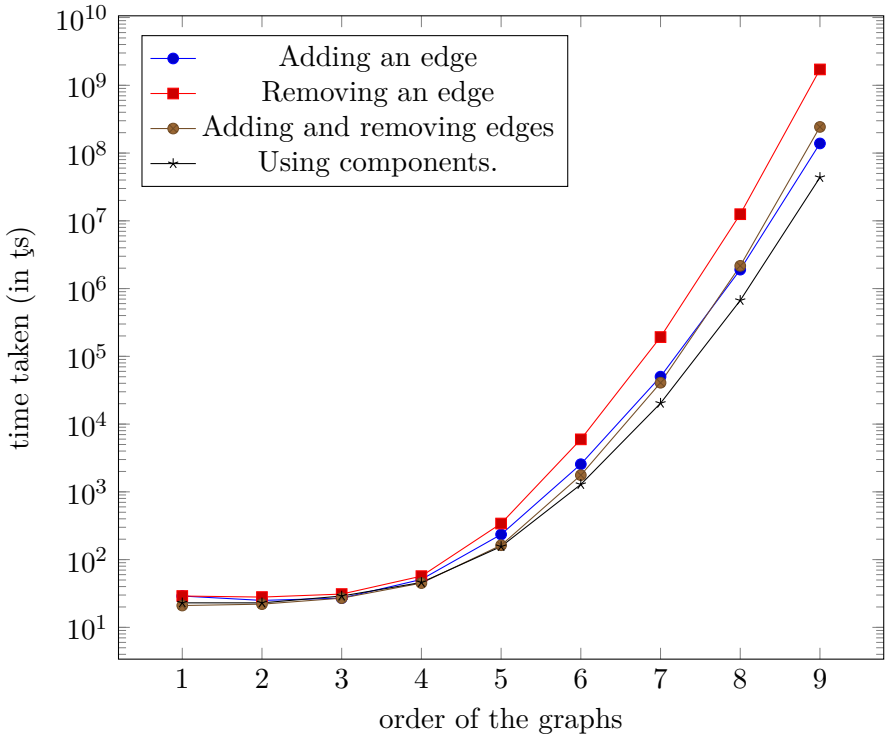


Figure 7.5: Time in microseconds taken by each algorithm to compute the values of $S(G,k)$ for all the graphs of order n.

Table 7.2: Time in microseconds taken by each algorithm to compute the values of $S(G,k)$ for all the graphs of order n.

n	Adding an edge	Removing an edge	Adding and removing	Components
1	29	29	21	23
2	25	28	22	23
3	27	31	27	29
4	51	57	45	46
5	235	341	163	156
6	2,559	5,956	1,763	1,286
7	50,146	$1.92 \cdot 10^5$	40,762	20,320
8	$1.9 \cdot 10^6$	$1.25 \cdot 10^7$	$2.17 \cdot 10^6$	$6.69 \cdot 10^5$
9	$1.39 \cdot 10^8$	$1.72 \cdot 10^9$	$2.43 \cdot 10^8$	$4.37 \cdot 10^7$

n is $\mathcal{A}\left(\overline{\mathsf{K}}_n\right)$. Thanks to counter-examples produced by TransProof, we have shown that despite its apparent simplicity, this conjecture cannot be proven using simple techniques like sequential edge removal. We have then refined this conjecture by proposing lower bounds related to the chromatic number $\chi(G)$ and to the maximum degree $\Delta(G)$ of G and stated three open problems. We have shown that these three conjectures are true for triangulated graphs and for graphs with maximum degree at most 2.

We have also given a general upper bound on $\mathcal{A}(G)$ that is valid for all graphs G, and a more precise one for graphs of order n and maximum degree $\Delta(G) \in \{1, 2, n-2\}$. Thanks to TransProof, we tested that, for all the graphs having up to 10 vertices, given G and a vertex v, $\mathcal{A}(G) > \mathcal{A}(G - v)$. We believe that this is true for all graphs.

The problem of finding a tight upper bound for graphs with maximum degree in $\{3, \ldots, n-3\}$ remains open. Since all graphs of order n and maximum degree $\Delta(G) \in \{1, n-2, n-1\}$ that maximize $\mathcal{A}(G)$ are isomorphic to $\left\lfloor \frac{n}{\Delta(G)+1} \right\rfloor \mathsf{K}_{\Delta(G)+1} \cup \mathsf{K}_{n \bmod (\Delta(G)+1)}$ (but this is not always true for $\Delta(G) = 2$), one could be tempted to think that this is also true when $3 \leq \Delta(G) \leq n-3$. We have checked this statement by enumerating all graphs having up to 12 vertices, using PHOEG [31] and the algorithm described in Section 7.7. We have thus determined that there is only one graph of order $n \leq 12$ and $\Delta(G) \neq 2$ (among more than a hundred billion), namely $\overline{\mathsf{C}}_6 \cup \mathsf{K}_4$, for which such a statement is wrong. Indeed, $\mathcal{A}(\overline{\mathsf{C}}_6 \cup \mathsf{K}_4) = 5.979 > 5.967 = \mathcal{A}(2\mathsf{K}_4 \cup \mathsf{K}_2)$, which shows that $2\mathsf{K}_4 \cup \mathsf{K}_2$ does not maximize $\mathcal{A}(G)$ among all graphs of order 10 and maximum degree 3.

Chapter 8

Concluding remarks and future work

In this document, we studied the possibility of using computers to assist in the writing of a proof by transformation in extremal graph theory. We introduced the notion of the metagraph of transformation which can be computed by TransProof, a tool that was developed during this work as a module of PHOEG, a computer assisted research framework in extremal graph theory. We summarize TransProof's ideas and discuss possible improvements in the next section. Section 8.2 summarizes the results obtained on the eccentric connectivity index, and Section 8.3 summarizes results and conjectures about the average number of colors in the non-equivalent colorings of a graph. Finally, we conclude with Section 8.4 by giving important lessons learned from this work and discussing the place of computers in extremal graph theory.

8.1 TransProof

TransProof was designed to be able to compute transformations of millions of graphs while removing duplicates caused by symmetries in those graphs. The data being stored in a database can then be queried to obtain a better insight about the effect of transformations on graph invariants.

We used TransProof in research to try to find bounds on the eccentric connectivity index and the average number of colors in the non-equivalent colorings of a graph. TransProof showed its usefulness in finding flaws in the proposed transformations to be used in proofs by transformation, saving time and energy. It also produced conjectures about those transformations.

Despite this, TransProof can be improved on several aspects. The first improvement is technical and is about a limitation of the speed at which transformations can be computed. We explain this in Subsection 8.1.1. But speed is not the only important factor when using TransProof. Once the metagraph has been computed and stored in a database, obtaining information from the metagraph is not easy for a user who is not fluent in database query languages. This problem is discussed in Subsection 8.1.2. Finally, the metagraph remains quite large, making manual study difficult. In Subsection 8.1.3, we list ideas and considerations to be able to use a computer to provide help when studying a large number of graphs or transformations.

8.1.1 Overcoming technical limitations

The speed of TransProof and the fact that it works in a multithreaded way combined with the amount of data generated results in the need for the computer to handle this data as fast as it is generated. In many computers, the speed of the storage is slower than the speed of TransProof and the outputted transformations queue up in memory waiting to be written to disk. This means that the memory quickly fills up until the computation needs to be stopped.

Of course, TransProof will limit its speed to make sure this does not happen, but the computation becomes slower. While this will not be a problem for relatively small sets of graphs, it can become frustrating when one wishes to try many different transformations on a large set of graphs.

The problem of having to handle or store large amounts of data as fast as it is produced is a research field known as Big Data. While Big Data is mostly focused on user-produced content such as social networks or search engines, studying the solutions used in this field and adapting them to TransProof

might allow for faster speeds and make the waiting time shorter. But, as we discuss in Subsection 8.1.3, those ideas might be difficult to adapt to graphs.

Another way to circumvent the problem of storage speed is to simply avoid storing the metagraph. Indeed, some queries can be answered without the need for a database, such as finding arcs of the metagraph that are not improving arcs. However, this requires being able to either compute the studied invariant fast enough or to retrieve pre-computed values from memory. In this last case, we are limited in the number of graphs we can handle as the memory is limited and often smaller than disk space for computers. This method would then be better suited for relatively small classes of graphs and transformations producing many instances.

In practice, large metagraphs are usually computed at the beginning of the research process and are mainly used to study the effect of simple transformations. In this case, computation time is not as problematic as the time taken to answer queries. They become more critical in a later stage when verifying if a transformation produces non-improving arcs in the metagraph. In this stage, more complex and numerous transformations are being considered. But, because this is a later stage in the writing of the proof, the number of graphs for which non-extremality is yet to be proven is smaller, making computing a metagraph faster. Thus, because TransProof is already quite fast, a more important problem is usability. We discuss this next.

8.1.2 User-friendlyness

Using TransProof, there are two main interactions needed from a user: the definition of a transformation and the writing of queries about the metagraph. While both are possible in the current state of TransProof, they are not easy for most users.

Defining a transformation can be done in TransProof by implementing it using the existing library that has been developed alongside it. In order to facilitate the definition of a transformation, a specific language has been defined to produce code automatically. However, this language currently assumes that

the basis of a transformation has a fixed number of vertices. It is not possible, for example, to use a basis corresponding to a complete graph with an arbitrary number of vertices. Also, because the conversion into code is done using a Rust macro, it still requires compiling code. This makes TransProof difficult to use for non-programmers. Therefore, a better and more user-friendly interface needs to be developed.

As we explained in Subsection 5.4.2, a transformation could be defined using a hypergraph model and then fed to TransProof which would use the *subgraph method* to compute the instances. It is then necessary to provide an easy-to-use interface for defining this hypergraph as well as adapt the *subgraph method* to handle the hypergraph constraints such as an arbitrary number of vertices inside a hyperedge. For example, a hyperedge might correspond to a maximal clique with not constraint on the order of this clique. For specific structures such as cliques or cycles, this can be done using existing algorithms to iterate over matching subgraphs [16, 36, 44].

Once the transformation has been defined and the metagraph has been constructed, it is necessary to also provide a way to query the database. For this, common and useful queries need to be gathered in order to produce an easy-to-use interface that removes the need for complex query languages. These queries and the definition of a transformation using a hypergraph would then be added to the web interface for PHOEG that is currently being developed.

However, the results of queries on the metagraph can still produce large amounts of information, making it difficult to extract interesting facts. We consider this problem in the next subsection.

8.1.3 Studying the results of TransProof

Because TransProof uses exact methods on complete classes of graphs, we usually limit the order of the graphs and, by doing this, limit the number of graphs to be considered. Usually, transformations will be computed on graphs up to order 10 and sometimes more for smaller classes of graphs such as trees. Despite this, the number of possibilities even for simple transformations is

exponential. For example, if we consider the connected graphs on 9 vertices and search for edges that, when removed, produce an increase in the diameter, we find 607 523 such cases.

Finding interesting patterns in this large number of edge removals can be difficult when manually going through all those cases. Thus, it would be interesting to use data analysis algorithms to extract features and patterns. As we presented in Subsection 5.3.1, those edge removals can be seen as labelled graphs and extracting patterns from a set of graphs is a known problem and tools such as Gaston [79], Gspan [93] or FFSM [61] exist to extract subgraphs that appear in all the graphs or in a large percentage of them. However, these tools are thought to be used in practical applications where vertices and edges have their own meaning, such as chemistry or networks. Because extremal graph theory is interested in the structure of the graphs, regardless of a context, such results might not be interesting. For example, such software might tell us that all our graphs contain a path on 3 vertices but will ignore the fact that those graphs are all cycles of different orders.

Instead of focusing on graph-specific methods, one might try to apply more general data mining methods such as clustering and see if the arcs of the meta-graph or the graphs where no transformation actually improves the invariant actually fall into different situations. For example, some edge removals could be transforming a cycle into a path, while others could consist in removing an edge from a clique. However, many data analysis methods use geometric methods and require being able to compute a distance between two points. And computing the distance between two graphs requires taking isomorphism into account. Indeed, if two graphs G and H are isomorphic, their distance should be zero, even if they are represented differently. Thus, usual metrics would not work.

Specific distances have been defined, such as the HIM metric [66] But the distance used needs to have a meaning related to the studied invariant. For example, if we study the chromatic number and obtain information related to the average distance in graphs, it might not be interesting for the problem at

hand. Therefore, one distance might not suffice. Instead, one might prefer having a set of distances related to different types of invariants such as colors, distances, or cliques and choose the best fitted one for the studied invariant when using data mining algorithms. We could then try applying data mining algorithms using one of those distances in order to extract information about TransProof's output.

Another way to compute the distance between two graphs is by computing their distance in a metagraph for some transformations. Because of the way the metagraph is constructed, it already accounts for isomorphism. This can be exploited in data mining algorithms but also be used directly to extract interesting information. For example, given a graph invariant $\mathcal{I}$, a metagraph M and a graph G that corresponds to a vertex in the metagraph, we could compute the minimal distance $\lambda(G)$ between G and any other graph $H \in V(M)$ such that $\mathcal{I}(G) < \mathcal{I}(H)$. Then, $\max_{G \in V(M)} \lambda(G)$ is the minimum number of transformations required to strictly increase $\mathcal{I}$ for any graph in the metagraph.

For example, let us consider the metagraph M for the rotation having all connected graph of order n as vertex set. For a graph G, $\lambda(G)$ is the minimum number of rotation required to obtain a graph G' such that $\mathcal{I}(G) < \mathcal{I}(H)$. If $\min_{G \in V(M)} \lambda(G) = 2$, this means that a proof by transformation would require at least two rotations as there is at least one graph for which a single rotation does not strictly increases $\mathcal{I}$.

When using simple transformations, the minimum value of λ as well as the cases where it is attained can give an idea of a more complex transformation and the amount of data outputted could be smaller as we do not output all non-improving arcs.

Such transformations could then be pre-computed and stored in PHOEG's database to avoid having to recompute them every time. However, if we use precomputed transformations, the metagraph built using those transformations must be strongly connected. That is, there should be a path from any graph to any other graph in the metagraph. The set of the four transformations that are the edge-removal, edge-addition, vertex-removal and vertex-addition

does have these properties.

For commonly fixed invariants in extremal graph theory problems (such as connectivity, size or the maximum degree), different sets of transformations could also be pre-computed that do not change these invariants. We could then consider smaller metagraphs by considering every value of those fixed invariants separately. For example, if we study graphs with n vertices and m edges, we will consider $\binom{n}{2}$ metagraphs (one for each possible value of m) but they will be considerably smaller. For example, for all graphs on 9 vertices, we will consider 28 metagraphs with the largest one having 34 040 vertices instead of a single one having 274 668 for all the graphs on 9 vertices.

But the problem of extracting meaningful information from a large set of graphs, is not specific to TransProof and is much more general. And, as we saw when studying the eccentric connectivity index, a simple answer such as knowing that there are non-improving arcs in a metagraph can prove very useful. We summarize the results obtained about this invariant in the following section.

8.2 The eccentric connectivity index

Given two integers n and p ($p \leq n$), we wrote a first paper [32] describing the graphs of order n with p pendant vertices that maximize the eccentric connectivity index and proving their extremality using a proof by transformation. This proof was written using TransProof to help test our transformation ideas.

A second paper [55] was written characterizing the graphs with fixed order and fixed diameter having the largest eccentric connectivity index as well as graphs with only a fixed order. This paper was not presented in this thesis as it does not use transformations.

Giving an upper bound on $\xi^c(G)$ for graphs with fixed order n and fixed size m proved more complex. We conjecture that the extremal graph in this case is the graph $\mathsf{E}_{n,D,k}$ constructed by joining each vertex of a clique K_{n-D-1}

to vertices u_0 and u_1 of a path $u_0, u_1, \ldots, u_D$ and k vertices of the clique to u_2 where $D = \left\lfloor \frac{2n+1-\sqrt{17+8(m-n)}}{2} \right\rfloor$ and $k = m - \binom{n-D+1}{2} - D + 1$.

Conjecture 8. *Let G be a connected graph of order n with m edges, where $n-1 \leq m \leq \binom{n-1}{2}$. Also, let*

$$D = \left\lfloor \frac{2n+1-\sqrt{17+8(m-n)}}{2} \right\rfloor \text{ and } k = m - \binom{n-D+1}{2} - D + 1.$$

Then $\xi^c(G) \leq \xi^c(\mathsf{E}_{n,D,k})$, with equality if and only if $G \simeq \mathsf{E}_{n,D,k}$ if $D > 3$ or $D = 3$, $k = n-4$ and G is the graph constructed from a path $u_0 - u_1 - u_2 - u_3$, by joining $1 \leq i \leq n-3$ vertices of a clique K_{n-4} to u_0, u_1, u_2 and the $n-4-i$ other vertices of K_{n-4} to u_1, u_2, u_3.

One should note that the maximum value of ξ^c for graphs with m edges does not necessarily increase when m grows. For example, using PHOEG, we checked, that the conjecture is true when $n = 9$ and $17 \leq m \leq 22$ and we can see in Table 8.1 that the value of $\xi^c(\mathsf{E}_{n,d,k})$ does not increases when m increases. Thus, a proof by induction on the number of edges might be difficult.

Table 8.1: ξ^c does not necessarily increase when the size of the graphs increases.

	$\mathsf{E}_{9,5,3}$	$\mathsf{E}_{9,4,0}$	$\mathsf{E}_{9,4,1}$	$\mathsf{E}_{9,4,2}$	$\mathsf{E}_{9,4,3}$	$\mathsf{E}_{9,4,4}$
m	17	18	19	20	21	22
ξ^c	134	132	132	132	132	132

To prove this conjecture using a proof by transformation, one would require transformations that do not change the number of edges. The most common one is the rotation but there exists graphs such as the two from Figure 8.1 where any rotation does not produce a graph with a higher value of $\xi^c(G)$. At least a second transformation would then be required or a more complex transformation than the rotation. It would, however, be interesting to characterize the graphs where no rotation increases ξ^c in order to know if

they belong to special classes of graphs. Also, identifying the cases where a rotation does increase ξ^c is essential for a proof by transformations.

Figure 8.1: Any rotation in those two graphs will not increase ξ^c.

Studying bounds on ξ^c with different constraints such as a fixed maximum clique size or a fixed maximum degree might also provide a better insight about the conjecture. Even more so because the conjectured extremal graph are defined by making a clique adjacent to vertices of a path.

A second invariant based on the colorings of a graph was also studied with the help of TransProof. This study led to two papers with several conjectures that are presented in the next section.

8.3 The average number of colors in the colorings of a graph

After establishing properties for the average number of colors in the non-equivalent colorings of a graph as well as the effect of some transformations on this invariant, we studied possible lower bounds in [60] for this invariant, and we were able to produce the three conjectured lower bounds given below. Those conjectures were proven for triangulated graphs and graphs with maximum degree at most 2.

Conjecture 23. *Let G be a graph of order n. Then,*

$$\mathcal{A}(G) \geq L_2(n, \chi(G))$$

with equality if and only if $G \simeq \mathsf{K}_{\chi(G)} \cup (n-\chi(G))\mathsf{K}_1$.

Conjecture 24. *Let G be a graph of order n. Then*

$$\mathcal{A}(G) \geq L_3(n, \Delta(G)+1)$$

with equality if and only if $G \simeq \mathsf{K}_{1,\Delta(G)} \cup (n-\Delta(G)-1)\mathsf{K}_1$.

Conjecture 25. *Let* G *be a graph of order* n*, then,*

$$\mathcal{A}(G) \geq L_1(n)$$

with equality if and only if $G \simeq \overline{\mathsf{K}}_n$.

Thanks to TransProof, we could see that Conjecture 25 cannot be proven using simple methods such as removing an edge, as there are graphs where removing any edge will strictly increase $\mathcal{A}(G)$. Moreover, TransProof showed us that, for any graph G up to order 10, removing a vertex will strictly decrease $\mathcal{A}(G)$. We conjecture that this is true for all graphs. But showing it requires being able to compute a lower or upper bound on the value $\mathcal{A}(G) - \mathcal{A}(G-v)$ for a vertex v in G. This result might then be useful in a proof by induction on the number of vertices of the graphs.

Conjecture 50. *Let* G *be a graph of order* n *and* v *be a vertex of* G*,*

$$\mathcal{A}(G) > \mathcal{A}(G-v).$$

Another potentially useful transformation consist in isolating a vertex v. That is, removing all the edges incident to v. We know this transformation does not always decrease $\mathcal{A}(G)$. Indeed, in the two graphs of Figure 8.2, isolating the squared vertex does not produce a graph with a lower value of $\mathcal{A}(G)$. But it is unknown if there is a graph such that isolating any vertex does not decrease $\mathcal{A}(G)$. If such a graph exists, its order is at least 11 as we could check that no such graph exists with less vertices. If, on the contrary, no such graph exists and if we characterize the cases where the transformation works, we could use it in a proof by transformation for either one of the three conjectures.

When studying upper bounds for $\mathcal{A}(G)$ in [59], we gave an upper bound for graphs with a fixed order n as well as for the graphs with fixed order and maximum degree $\Delta(G) \in \{1, 2, n-2\}$. We conjectured that for all graphs G of order n with maximum degree $3 \leq \Delta(G) \leq n-3$, $\mathcal{A}(G) \leq$

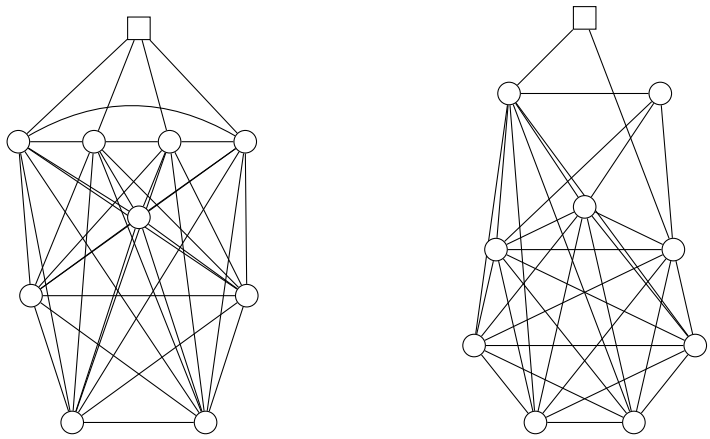


Figure 8.2: Isolating the squared vertex in either of those two graphs does not decrease $\mathcal{A}(G)$

$\mathcal{A}\left(\left\lfloor \frac{n}{\Delta(G)+1} \right\rfloor \mathsf{K}_{\Delta(G)+1} \cup \mathsf{K}_{n \bmod (\Delta(G)+1)}\right)$. However, thanks to the improvements in Algorithm 8 which uses transformations to compute $\mathcal{A}(G)$, we were able to find that the graph $\overline{\mathsf{C}}_6 \cup \mathsf{K}_4$ is a counter-example to this conjecture, and that it is the only counter-example among all the graphs up to order 12 which is more than a hundred billion graphs.

Before trying to prove this conjecture, one has to first study this counter-example in order to know if it is a special case or a family of counter-examples. This would probably require a theoretical analysis as the number of non-isomorphic graphs becomes too large to use an exact method when $n \geq 13$.

8.4 Conclusion

In Section 3.2, we listed different tools with varying levels of complexity. We also mentioned that most of those tools in extremal graph theory aimed towards conjecturing. With TransProof, we moved to the next step, that is, to help researchers prove the conjectures generated by such tools.

Similarly to some of those computer-assisted conjecturing systems, and unlike others, TransProof is not automated and does not use heuristics. Instead,

it uses a simple idea that is computing the metagraph of transformations. But this idea already proved useful by giving insight on how invariants behaved under some transformations and by producing counter-examples, thereby saving us time that would have been used in trying to prove an incorrect result.

This shows that complexity in computer-assisted proving systems is not a requirement. Of course, simplicity is not a requirement either. There is room for improvement in TransProof. Being able to extract meaningful information from a large set of graphs or applications of a transformation, such as common structures or similar values of an invariant, could prove valuable to researchers and not only when using TransProof. Heuristics or artificial intelligence might then be a valid option.

However, in computer-assisted proving, because a proof needs to be understood in order to be accepted, complex computer systems need to be careful that their results remain understandable by humans.

We conclude this work with a citation from Isaac Asimov [8, p. 187]:

> "The machine is only a tool after all, which can help humanity progress faster by taking some of the burdens of calculations and interpretations off its back. The task of the human brain remains what it has always been; that of discovering new data to be analyzed, and of devising new concepts to be tested."

Bibliography

[1] R. Absil, E. Camby, A. Hertz, and H. Mélot. A sharp lower bound on the number of non-equivalent colorings of graphs of order n and maximum degree $n-3$. *Discrete Applied Mathematics*, 234:3–11, 2018.

[2] R. Absil and H. Mélot. Digenes: genetic algorithms to discover conjectures about directed and undirected graphs. *arXiv preprint arXiv:1304.7993*, 2013.

[3] M. Aigner, G. M. Ziegler, and A. Quarteroni. *Proofs from the Book (fourth edition)*, pages 235–239. Springer, 2010.

[4] M. O. Albertson. The irregularity of a graph. *Ars Combinatoria*, 46:219–225, 1997.

[5] H. Alzer. On Engel's Inequality for Bell Numbers. *Journal of Integer Sequences*, 22, 2019. Article 19.7.1.

[6] M. Aouchiche. *Comparaison automatisée d'invariants en théorie des graphes*. École polytechnique, 2006.

[7] K. I. Appel and W. Haken. *Every planar map is four colorable*, volume 98. American Mathematical Society, 1989.

[8] I. Asimov. The evitable conflict. *Astounding Science Fiction*, 45(4):48–68, 1950.

[9] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, et al. *The Coq proof assistant reference manual: Version 6.1.* PhD thesis, Inria, 1997.

[10] F. K. Bell. A note on the irregularity of graphs. *Linear Algebra and its Applications*, 161:45–54, 1992.

[11] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(7):1–13, 2013.

[12] V. Brankov and D. Stevanovic. An invitation to newGRAPH. *Rendiconti del Seminario Matematico di Messina, Serie II*, 25:211–216, 2003.

[13] G. Brinkmann, K. Coolsaet, J. Goedgebeur, and H. Mélot. House of Graphs: a database of interesting graphs. *Discrete Applied Mathematics*, 161(1-2):311–314, 2013.

[14] G. Brinkmann, J. Goedgebeur, and B. D. McKay. The generation of fullerenes. *Journal of chemical information and modeling*, 52(11):2910–2918, 2012.

[15] G. Brinkmann, B. D. McKay, et al. Fast generation of planar graphs. *MATCH Communications in Mathematical and in Computer Chemistry*, 58(2):323–357, 2007.

[16] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[17] V. Bruyère, G. Joret, and H. Mélot. Trees with given stability number and minimum number of stable sets. *Graphs and combinatorics*, 28(2):167–187, 2012.

[18] V. Bruyère and H. Mélot. Fibonacci index and stability number of graphs: a polyhedral study. *Journal of combinatorial optimization*, 18(3):207–228, 2009.

[19] E. Camby and G. Caporossi. Studying graphs and their induced subgraphs with the computer: GraphsInGraphs. *Cahiers du GERAD*, pages 1–11, 2016.

[20] J. Cano, J.-M. Díaz-Báñez, C. Huemer, and J. Urrutia. The edge rotation graph. *Graphs and Combinatorics*, 29(5):1207–1219, 2013.

[21] G. Caporossi. Variable neighborhood search for extremal vertices: The AutoGraphiX-III system. *Computers & Operations Research*, 78:431–438, 2017.

[22] G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs: 1 the AutoGraphiX system. *Discrete Mathematics*, 212(1):29–44, 2000.

[23] J. Cardinal, M. Labbé, S. Langerman, E. Levy, and H. Mélot. A tight analysis of the maximal matching heuristic. In *International Computing and Combinatorics Conference*, pages 701–709. Springer, 2005.

[24] A. Cayley. On the mathematical theory of isomers. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 47(314):444–447, 1874.

[25] Y. Cheng, P. Ding, T. Wang, W. Lu, and X. Du. Which category is better: benchmarking relational and graph database management systems. *Data Science and Engineering*, 4(4):309–322, 2019.

[26] J. Christophe, S. Dewez, J. Doignon, S. Elloumi, G. Fasbender, P. Grégoire, D. Huygens, M. Labbé, H. Mélot, and H. Yaman. Linear inequalities among graph invariants: using GraPHedron to uncover optimal relationships. *Networks: An International Journal*, 52(4):287–298, 2008.

[27] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pages 149–159, 2001.

[28] D. M. Cvetković, L. L. Kraus, and S. K. Simić. Discussing graph theory with a computer I. Implementation of graph theoretic algorithms. *Publikacije Elektrotehničkog fakulteta. Serija Matematika i fizika*, (716/734):100–104, 1981.

[29] C. De La Higuera, J.-C. Janodet, É. Samuel, G. Damiand, and C. Solnon. Polynomial algorithms for open plane graph and subgraph isomorphisms. *Theoretical Computer Science*, 498:76–99, 2013.

[30] E. DeLaVina. Graffiti. pc. *Graph Theory Notes of New York*, 42(3):26–30, 2002.

[31] G. Devillez, P. Hauweele, and H. Mélot. PHOEG Helps to Obtain Extremal Graphs. In *Operations Research Proceedings 2018: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR)*, pages 251–257, 2019.

[32] G. Devillez, A. Hertz, H. Mélot, and P. Hauweele. Minimum Eccentric Connectivity Index for Graphs with Fixed Order and Fixed Number of Pendant Vertices. *Yugoslav Journal of Operations Research*, 29(2), 2019.

[33] R. Diestel. *Graph Theory*. Springer-Verlag, second edition edition, 2017.

[34] D. Dimitrov, S. Brandt, and H. Abdo. The total irregularity of a graph. *Discrete Mathematics & Theoretical Computer Science*, 16, 2014.

[35] G. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961.

[36] E. Dixon and S. Goodman. An all cycle algorithm for undirected graphs. In *Proceedings of the ACM annual conference*, pages 430–5, 1973.

[37] F. M. Dong, K. M. Koh, and K. L. Teo. *Chromatic polynomials and chromaticity of graphs*. World Scientific Publishing Company, 2005.

[38] T. Došlić, A. Graovac, and O. Ori. Eccentric Connectivity Index of Hexagonal Belts and Chains. *MATCH Communications in Mathematical and in Computer Chemistry*, 65:745–752, 2011.

[39] B. Duncan. Bell and Stirling numbers for disjoint unions of graphs. *Congressus Numerantium*, 206, 2010.

[40] B. Duncan and R. B. Peele. Bell and Stirling numbers for graphs. *Journal of Integer Sequences*, 12, 2009. Article 09.7.1.

[41] H. Dureja, S. Gupta, and A. Madan. Predicting anti-HIV-1 activity of 6-arylbenzonitriles: Computational approach using superaugmented eccentric connectivity topochemical indices. *Journal of Molecular Graphics and Modelling*, 26(6):1020–1029, 2008.

[42] V. Fack, S. Lievens, and J. Van der Jeugt. On the diameter of the rotation graph of binary coupling trees. *Discrete mathematics*, 245(1-3):1–18, 2002.

[43] S. Fajtlowicz. On Conjectures of Graffiti. *Discrete Mathematics*, 72:113–118, 1988.

[44] H. N. Gabow and E. W. Myers. Finding all spanning trees of directed and undirected graphs. *SIAM Journal on Computing*, 7(3):280–287, 1978.

[45] D.T. Galvin, D.and Thanh. Stirling numbers of forests and cycles. *Electronic Journal of Combinatorics*, 20(1), 2013. Paper P73.

[46] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[47] F. Gavril. Some NP-complete problems on graphs. Technical report, Computer Science Department, Technion, 2011.

[48] G. Gonthier et al. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[49] S. Gupta, M. Singh, and A. Madan. Eccentric distance sum: A novel graph invariant for predicting biological and physical properties. *Journal of Mathematical Analysis and Applications*, 275(1):386–401, 2002.

[50] I. Gutman. Topological indices and irregularity measures. *Bulletin of the international mathematical virtual institute*, 8:469–475, 2018.

[51] P. Hansen, A. Hertz, R. Kilani, O. Marcotte, and D. Schindl. Average distance and maximum induced forest. *Journal of Graph Theory*, 60(1):31–54, 2009.

[52] P. Hansen and H. Mélot. Variable neighborhood search for extremal Graphs. 6. Analyzing bounds for the connectivity index. *Journal of Chemical Information and Computer Sciences*, 43(1):1–14, 2003.

[53] P. Hansen and H. Mélot. Variable neighborhood search for extremal graphs. 9. Bounding the irregularity of a graph. *Dimacs series in discrete mathematics and theoretical computer science*, 69:253, 2005.

[54] R.S. Haoer, K.A. Atan, A.M. Khalaf, and R. Hasni. Eccentric connectivity index of unicyclic graphs with application to cycloalkanes. In *International Conference on Research and Education in Mathematics (ICREM7)*, pages 211 – 214, 2015.

[55] P. Hauweele, A. Hertz, H. Mélot, B. Ries, and G. Devillez. Maximum eccentric connectivity index for graphs with given diameter. *Discrete Applied Mathematics*, 268:102–111, 2019.

[56] A. Hertz, A. Hertz, and H. Mélot. Using graph theory to derive inequalities for the Bell numbers. *Journal of Integer Sequences*, 24, 2021. Article 21.10.6.

[57] A. Hertz, V. Lozin, B. Ries, V. Zamaraev, and D. de Werra. Dominating induced matchings in graphs containing no long claw. *Journal of Graph Theory*, 88(1):18–39, 2018.

[58] A. Hertz and H. Mélot. Counting the number of non-equivalent vertex colorings of a graph. *Discrete Applied Mathematics*, 203:62–71, 2016.

[59] A. Hertz, H. Mélot, S. Bonte, G. Devillez, and P. Hauweele. Upper bounds on the average number of colors in the non-equivalent colorings of a graph. *arXiv preprint arXiv:2105.01120*, 2021.

[60] A. Hertz, H. Mélot, S. Bonte, and G. Devillez. Lower Bounds and properties for the average number of colors in the non-equivalent colorings of a graph. *arXiv preprit arXiv:2014.14172*, 2021.

[61] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Third IEEE international conference on data mining*, pages 549–552, 2003.

[62] A. Ilić and I. Gutman. Eccentric connectivity index of chemical trees. *MATCH. Communications in Mathematical and in Computer Chemistry*, 65(3):731–744, 2011.

[63] O. F. Inc. The On-Line Encyclopedia of Integer Sequences. `https://oeis.org/A000088`, 2021. Accessed: 2021-11-19.

[64] E. B. Jarrett. Edge rotation and edge slide distance graphs. *Computers & Mathematics with Applications*, 34(11):81–87, 1997.

[65] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. Society for Industrial and Applied Mathematics, 2007.

[66] G. Jurman, R. Visintainer, M. Filosi, S. Riccadonna, and C. Furlanello. The HIM glocal metric and kernel for network comparison and classification. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[67] M. Karelson. *Molecular descriptors in QSAR/QSPR*. Wiley-Interscience, 2000.

[68] Z. Kereskényi-Balogh and G. Nyul. Stirling numbers of the second kind and Bell numbers for graphs. *Australasian Journal of Combinatorics*, 58:264–274, 2014.

[69] S. Klabnik and C. Nichols. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019.

[70] V. Kumar, S. Sardana, and A. K. Madan. Predicting anti-HIV activity of 2,3-diaryl-1,3-thiazolidin-4-ones: computational approach using reformed eccentric connectivity index. *Journal of molecular modeling*, 10:399–407, 2004.

[71] J. M. Lucas. The rotation graph of binary trees is Hamiltonian. *Journal of Algorithms*, 8(4):503–535, 1987.

[72] B. McKay. Description of graph6, sparse6 and digraph6 encodings. `https://users.cecs.anu.edu.au/~bdm/data/formats.txt`, 2015. Accessed: 2022-04-06.

[73] B. D. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2):306–324, 1998.

[74] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[75] H. Mélot. Facet defining inequalities among graph invariants: the system GraPHedron. *Discrete Applied Mathematics*, 156(10):1875–1891, 2008.

[76] T. Miyazaki. The complexity of mckays canonical labeling algorithm. In *Groups and Computation II*, volume 28, pages 239–256, 1997.

[77] M.J. Morgan, S. Mukwembi, and H.C. Swart. A lower bound on the eccentric connectivity index of a graph. *Discrete Applied Mathematics*, 160:248 – 258, 2012.

[78] M.J. Morgan, S. Mukwembi, and H.C. Swart. Extremal regular graphs for the eccentric connectivity index. *Quaestiones Mathematicae*, 37:435 – 444, 2014.

[79] S. Nijssen and J. N. Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77–87, 2005.

[80] A. Odlyzko and L. Richmond. On the number of distinct block sizes in partitions of a set. *Journal of Combinatorial Theory, Series A*, 38(2):170–181, 1985.

[81] A. Peeters, K. Coolsaet, G. Brinkmann, N. Van Cleemput, and V. Fack. GrInvIn in a nutshell. *Journal of mathematical chemistry*, 45(2):471–477, 2009.

[82] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[83] S. Sardana and A. K. Madan. Application of graph theory: relationship of molecular connectivity index, Wiener's index and eccentric connectivity index with diuretic activity. *MATCH Communications in Mathematical and in Computer Chemistry*, 43:85–98, 2001.

[84] V. Sharma, R. Goswami, and A. Madan. Eccentric connectivity index: a novel highly discriminating topological descriptor for structure property and structure activity studies. *Journal of chemical information and computer sciences*, 37(2):273–282, 1997.

[85] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[86] R. Todeschini and V. Consonni. *Handbook of molecular descriptors*, volume 11. Wiley-VCH, 2008.

[87] P. Turán. On an extremal problem in graph theory. *Középiskolai Matematikai és Fizikai Lapok*, 48(436-452):137, 1941.

[88] J. R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *Journal of Experimental Algorithmics (JEA)*, 15:1–1, 2011.

[89] L. Von Collatz and U. Sinogowitz. Spektren endlicher grafen. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 21, pages 63–77. Springer, 1957.

[90] M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In *International Conference on Theorem Proving in Higher Order Logics*, pages 33–38, 2008.

[91] H. Wiener. Structural determination of paraffin boiling points. *Journal of the American Chemical Society*, 69(1):17–20, 1947.

[92] R. Wilson. *Four Colors Suffice: How the Map Problem Was Solved-Revised Color Edition*. Princeton university press, 2013.

[93] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, 2002.

[94] J. Yang and F. Xia. The Eccentric Connectivity Index of Dendrimers. *International Journal of Contemporary Mathematical Sciences*, 5:2231 – 2236, 2010.

[95] J. Zhang, B. Zhou, and Z. Liu. On the minimal eccentric connectivity indices of graphs. *Discrete Mathematics*, 312:819 – 829, 2012.

[96] B. Zhou and Z. Du. On eccentric connectivity index. *MATCH Communications in Mathematical and in Computer Chemistry*, 63(1):181–198, 2010.

Index

Glossary of notations

$Aut(G)$ The automorphism group of a graph G.

$\mathcal{A}(G)$ The average number of colors in the non-equivalent colorings of a graph G.

T_n The total number of blocks in all partitions of a set of n elements.

A_n The average number of blocks in a partition of a set of n elements, which can be defined as $\mathsf{A}_n = \frac{\mathsf{T}_n}{\mathsf{B}_n}$.

B_n A Bell number. The number of ways to partition a set of n elements.

$\overline{G}$ The complement of a graph G.

K_n A complete graph on n vertices.

M_n The graph obtained from K_n by removing a maximum matching and, if n is odd, an additional edge adjacent to the unique vertex that still has degree $n-1$.

$G_{|(u,v)}$ The graph (of order $n-1$) obtained from G by identifying the vertices u and v.

C_n A cycle on n vertices.

$deg_G(u)$ The degree of a vertex u in a graph G.

$dist_G(u,v)$ The distance between two vertices u and v in a graph G.

$e_G(u)$ The eccentricity of a vertex u in a graph G.

$\xi^c(G)$ The eccentric connectivity index of a graph G.

(u, v) An edge between vertices u and v in a graph.

$E(G)$ The edge set of a graph G.

$\overline{\mathsf{K}}_n$ An empty graph on n vertices, that is, with no edges.

$\mathsf{E}_{n,D,k}$ The graph of order n constructed from a path $u_0 - u_1 - \ldots - u_D$ by joining each vertex of a clique K_{n-D-1} to u_0 and u_1, and k vertices of the clique to u_2.

$\mathsf{E}_{n,D}$ The graph constructed from a path $u_0 - u_1 - \ldots - u_D$ by joining each vertex of a clique K_{n-D-1} to u_0, u_1 and u_2.

$\mathsf{H}_{n,p}$ For $n \geq 4$ and $p \leq n-3$, the graph of order n obtained by adding a dominating vertex to the graph of order $n-1$ having p vertices of degree 0, and $n-1-p$ vertices of degree 1 if $n-p$ is odd or $n-2-p$ vertices of degree 1 and one vertex of degree 2 if $n-p$ is even.

$\mathrm{imb}_{(u,v)}$ The imbalance of an edge (u, v).

$\mathrm{irr}(G)$ The irregularity of a graph G.

$G \simeq H$ The graph G is isomorphic to the graph H.

$\Delta(G)$ The maximal degree in a graph G.

$N(u)$ The set of neighbors of a vertex u, that is, vertices that are adjacent to u.

$\mathcal{B}(G)$ The total number of non-equivalent colorings of a graph G.

$|G|$ The order (number of vertices) of a graph G.

P_n A path on n vertices.

Q_n For $n \geq 3$, the graph obtained from P_n by adding an edge between an extremity v of P_n and the vertex at distance 2 from v on P_n.

$rot(u, v, w)$ The edge-rotation consisting in removing the edge (u, v) and adding the edge (u, w).

$S(G, k)$ The number of non-equivalent colorings of a graph G that use *exactly* k colors.

$||G||$ The size (number of edges) of a graph G.

S_n A star on n vertices, that is, a vertex made adjacent to all the vertices of a cycle on $n - 1$ vertices.

$\mathsf{S}_{n,x}$ The graph of order n obtained by linking all vertices of a stable set of $n - x$ vertices with all vertices of a clique K_x.

$\left\{ {n \atop k} \right\}$ A Stirling number of the second kind. The number of ways to partition a set of n elements into k non-empty subsets.

$\mathcal{T}(G)$ The total number of color classes in the non-equivalent colorings of a graph G.

$V(G)$ The vertex set of a graph G.