Impact of Symmetries in Graph Clustering

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

(Dr. Ing.)

von der Fakultät für Wirtschaftswissenschaften

des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M. Sc. Fabian Ball

Tag der mündlichen Prüfung:	16. Januar 2019
Referent:	Prof. Dr. Andreas Geyer-Schulz
Korreferent:	Prof. Dr. Stefan Nickel

Kurzfassung

Diese Dissertation beschäftigt sich mit der durch die Automorphismusgruppe definierten Symmetrie von Graphen und wie sich diese auf eine Knotenpartition, als Ergebnis von Graphenclustering, auswirkt. Durch eine Analyse von nahezu 1700 Graphen aus verschiedenen Anwendungsbereichen kann gezeigt werden, dass mehr als 70 % dieser Graphen Symmetrien enthalten. Dies bildet einen Gegensatz zum kombinatorischen Beweis, der besagt, dass die Wahrscheinlichkeit eines zufälligen Graphen symmetrisch zu sein bei zunehmender Größe gegen Null geht. Das Ergebnis rechtfertigt damit die Wichtigkeit weiterer Untersuchungen, die auf mögliche Auswirkungen der Symmetrie eingehen. Bei der Analyse werden sowohl sehr kleine Graphen (< 100 Knoten/Kanten) als auch sehr große Graphen (> 10 000 000 Knoten/> 25 000 000 Kanten) berücksichtigt.

Weiterhin wird ein theoretisches Rahmenwerk geschaffen, das zum einen die detaillierte Quantifizierung von Graphensymmetrie erlaubt und zum anderen Stabilität von Knotenpartitionen hinsichtlich dieser Symmetrie formalisiert. Eine Partition der Knotenmenge, die durch die Aufteilung in disjunkte Teilmengen definiert ist, wird dann als stabil angesehen, wenn keine Knoten symmetriebedingt von der einen in die andere Teilmenge abgebildet werden und dadurch die Partition verändert wird. Zudem wird definiert, wie eine mögliche Zerlegbarkeit der Automorphismusgruppe in unabhängige Untergruppen als lokale Symmetrie interpretiert werden kann, die dann nur Auswirkungen auf einen bestimmten Bereich des Graphen hat. Um die Auswirkungen der Symmetrie auf den gesamten Graphen und auf Partitionen zu quantifizieren, wird außerdem eine Entropiedefinition präsentiert, die sich an der Analyse dynamischer Systeme orientiert. Alle Definitionen sind allgemein und können daher für beliebige Graphen angewandt werden. Teilweise ist sogar eine Anwendbarkeit für beliebige Clusteranalysen gegeben, solange deren Ergebnis in einer Partition resultiert und sich eine Symmetrierelation auf den Datenpunkten als Permutationsgruppe angeben lässt.

Um nun die tatsächliche Auswirkung von Symmetrie auf Graphenclustering zu untersuchen wird eine zweite Analyse durchgeführt. Diese kommt zum Ergebnis, dass von 629 untersuchten symmetrischen Graphen 72 eine instabile Partition haben. Für die Analyse werden die Definitionen des theoretischen Rahmenwerks verwendet. Es wird außerdem festgestellt, dass die Lokalität der Symmetrie eines Graphen maßgeblich beeinflusst, ob dessen Partition stabil ist oder nicht. Eine hohe Lokalität resultiert meist in einer stabilen Partition und eine stabile Partition impliziert meist eine hohe Lokalität.

Bevor die obigen Ergebnisse beschrieben und definiert werden, wird eine umfassende Einführung in die verschiedenen benötigten Grundlagen gegeben. Diese umfasst die formalen Definitionen von Graphen und statistischen Graphmodellen, Partitionen, endlichen Permutationsgruppen, Graphenclustering und Algorithmen dafür, sowie von Entropie. Ein separates Kapitel widmet sich ausführlich der Graphensymmetrie, die durch eine endliche Permutationsgruppe, der Automorphismusgruppe, beschrieben wird. Außerdem werden Algorithmen vorgestellt, die die Symmetrie von Graphen ermitteln können und, teilweise, auch das damit eng verwandte Graphisomorphie Problem lösen.

Am Beispiel von Graphenclustering gibt die Dissertation damit Einblicke in mögliche Auswirkungen von Symmetrie in der Datenanalyse, die so in der Literatur bisher wenig bis keine Beachtung fanden.

Abstract

This dissertation is about how the graph symmetry, which is defined by the graph's automorphism group, has an impact on the result of a graph clustering algorithm—a partition of nodes. By the analysis of nearly 1700 graphs from diverse application domains we can show that more than 70 % of these graphs contain symmetries. This is a contradiction to the theoretical result from combinatorics which says that the probability of increasingly large random graphs to be symmetric tends to zero. The result justifies the importance of further research that investigates the possible impact of this symmetry. We incorporate very small graphs (< 100 nodes/edges) as well as very large graphs (> 10 000 000 nodes/> 25 000 000 edges) in the analysis.

In addition, we present a theoretical framework that, on the one hand, allows a detailed quantification of graph symmetry and, on the other hand, formalizes the stability of node partitions regarding this symmetry. A partition of nodes—which is defined by the division of the node set into disjoint subsets—is said to be stable if there is no symmetry that maps nodes from one subset to another by altering the partition at the same time. Furthermore, we define how we can interpret a possible decomposition of the automorphism group into independent subgroups as local symmetry, which then has only an impact on a restricted area of the graph. To allow a quantification of the impact of the symmetry on the whole graph and on partitions of it, we also present a special entropy definition that has its origin in the analysis of complex dynamical systems. All our definitions are generic and can thus be applied on any graphs. For some of them, even the applicability for arbitrary cluster analyses is possible, as long as they result in a partition of data points on which a symmetry relation in form of a permutation group can be defined.

A second analysis is carried out to investigate the actual impact of symmetry on graph clustering. It comes to the result that out of 629 graphs a total of 72 have an unstable partition. We employ the definitions from the theoretical framework in the analysis. Moreover, we find the local structure of the symmetry to be the most significant factor that determines if the partition is stable or not. A very local symmetry mostly results in a stable partition and, contrarily, a stable partition mostly implies a local symmetry.

However, before the above results are described and defined, we give an extensive introduction to the diverse basic knowledge upon which they build. This includes the formal definitions of graphs and statistical graph models, partitions, finite permutation groups, graph clustering and algorithms for that, as well as the entropy concept. A separate chapter is dedicated to graph symmetry, which is described by a finite permutation group—the automorphism group of the graph. Aside from that, we present algorithms that are able to identify the graph symmetry. Some of them are even capable to solve the strongly related graph isomorphism problem.

All in all, this thesis exemplarily gives insights into a possible impact of symmetry in data analysis, which was considered—if at all—only scarcely in the literature.

Danksagung

Ich möchte diese Gelegenheit sehr gerne nutzen um einigen Personen zu danken. Allen voran ist natürlich Herr Prof. Dr. Andreas Geyer-Schulz zu nennen, der mich über die Jahre in vielen Belangen unterstützt hat. In zahlreichen Diskussionen wurden so neue Ideen generiert, Fehler entdeckt und verbessert sowie neue, erstaunliche Erkenntnisse gewonnen.

Direkt daran anschließend sind meine (teilweise ehemaligen) Kollegen am Lehrstuhl zu erwähnen. Seien es die vielen gemeinsamen Mittagessen, fachliche Diskussionen, das gemeinschaftliche Korrigieren von Klausuren oder auch einfach nur das nette Beisammensein bei diversen Gelegenheiten oder bei Gesprächen auf dem Flur: Es war mir immer eine Freude. Danke Abdolreza, Andreas S., Diana, Jens, Leon, Marvin, Maxi, Thomas, Victoria-Anne.

Besonderer Dank gilt Daniel, der sich auch kurzfristig noch die Mühe gemacht hat große Teile meiner Arbeit aus unabhängiger Perspektive und mit fachlichem Verstand zu korrigieren.

Ein sehr großes Dankeschön gilt der gesamten Open Source Community. Durch den Einsatz zahlreicher Menschen, die dafür häufig ihre Freizeit opfern, entstehen viele gute Anwendungen und Software-Bibliotheken die kostenlos durch jeden genutzt werden können. Diese Arbeit wäre ohne die Verwendung vieler Open Source Projekte in dieser Form nicht so einfach möglich gewesen. Auch die unabhängige Nachvollziehbarkeit und Überprüfbarkeit wird dadurch nicht unnötig erschwert. Insbesondere hervorzuheben sind LaTeX mit seinen vielen Zusatzpaketen, das komplette `Python` Ökosystem mit den Paketen `NumPy`, `SciPy`, `pandas`, `IPython`, `SymPy`, `Matplotlib`, `networkX`, das `R` Ökosystem, und auch die Entwickler und Unterstützer von Linux samt der Vielzahl an Distributionen. Weiterhin gilt der Dank den Autoren von `nauty` und `saucy` für die Bereitstellung des Quellcodes.

Elisa möchte ich dafür danken, dass sie mich die ganze Zeit über begleitet hat und über die ganzen Jahre geduldig geblieben ist. Meinen Eltern danke ich für ihre Unterstützung über die gesamte Studienzeit hinweg, aber natürlich auch schon davor und darüber hinaus. Zu guter Letzt möchte ich noch allen Freunden, Bekannten und Verwandten danken die mich immer wieder gefragt haben was ich denn überhaupt mache, wie man das für Laien verständlich erklären kann, und wann ich denn (endlich) fertig werde. Dieser gesunde Druck von außen hat mich zum Nachdenken angeregt und immer wieder angespornt.

Karlsruhe, im Januar 2019 *Fabian Ball*

Contents

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Symmetry of Graphs 8
- 1.3 Scope of the Thesis 12
- 1.4 Outline 12

2 Foundations **15**
- 2.1 Graph Theory 15
  - 2.1.1 Types of Graphs 15
  - 2.1.2 Graph Representations 17
  - 2.1.3 Graph Properties and Further Definitions 24
  - 2.1.4 Graph Classes 27
- 2.2 Graph Models 28
  - 2.2.1 Properties of Real-World Networks 29
  - 2.2.2 Erdős-Rényi Random Graph Model 31
  - 2.2.3 Preferential Attachment 33
  - 2.2.4 Copying Nodes 34
  - 2.2.5 Small-World Model 34
- 2.3 Partitions 35
  - 2.3.1 General Definition 35
  - 2.3.2 Graph Partitions 36
- 2.4 Finite Permutation Groups 37
  - 2.4.1 Permutations 37
  - 2.4.2 Permutation Groups 40
  - 2.4.3 Products of Permutation Groups 46
- 2.5 Graph Clustering 49
  - 2.5.1 Modularity 50
  - 2.5.2 The Randomized Greedy Algorithm and its Extensions 55
  - 2.5.3 Label Propagation 57
  - 2.5.4 Other Approaches 58
- 2.6 Entropy 58
  - 2.6.1 General Definition 58
  - 2.6.2 Entropy of Graphs 59

3 Graph Morphisms **61**
3.1 Graph Isomorphism 62
3.2 Graph Automorphism 64
3.2.1 Simplification of Graphs 67
3.2.2 Further Concepts 68
3.2.3 Symmetry Bounds 68
3.3 Graph Invariants 70
3.4 Algorithms 71
3.4.1 Efficient Algorithms for Special Graphs 72
3.4.2 General Algorithms 76
3.4.3 Practical Algorithms 78
3.4.4 Different Approaches and Algorithms for Related Problems 88
3.5 Other Graph Symmetry Concepts 89

4 Symmetry Analysis of Real-world Graphs **93**
4.1 Measures of Graph Symmetry 94
4.1.1 Measures Based on the Group Size 94
4.1.2 Network Redundancy 95
4.2 Description of the Analysis Procedure 96
4.2.1 Network Data 97
4.2.2 Downloading the Datasets 97
4.2.3 Analysis Procedure 98
4.3 Analysis Results 99
4.3.1 Simple Graphs 100
4.3.2 Simplified Graphs 106
4.3.3 A Comparison of the Symmetries in Simple and Simplified Graphs . . . 111
4.3.4 Discussion 111

5 Graph Symmetries **113**
5.1 Reasons for Symmetry in Graphs 113
5.2 Local and Global Symmetry 117
5.2.1 Definitions of Local and Global Symmetry 117
5.2.2 Examples 120
5.2.3 Implications of Local Symmetry 121
5.3 Weak Symmetry Based on Clustering 122

6 Partition Stability **125**
6.1 Search Space Partition 125
6.2 Equivalent Stability Definitions 127
6.2.1 Splitting the Automorphism Group 127
6.2.2 Partition of Blocks 128

6.2.3 Partition Refinement Lattice . . . 129
6.2.4 Equivalence of the Definitions . . . 130
6.2.5 Fast Partition Stability Analysis . . . 132
6.3 Impact of Specific Graph Structures on Modularity Optimal Partitions . . . 133
6.3.1 Star Graphs . . . 133
6.3.2 Complete Bipartite Subgraphs . . . 135
6.4 Kolmogorov-Sinai Entropy . . . 142
6.4.1 Dynamical Systems . . . 143
6.4.2 Ergodic Theory and Entropy . . . 144
6.4.3 Kolmogorov-Sinai Entropy of a Graph . . . 148

7 Impact of Symmetry on Clustering Partitions of Real-world Graphs . . . 163
7.1 Measurement . . . 163
7.2 Hypotheses . . . 164
7.3 Description of the Analysis Procedure . . . 166
7.4 Analysis Results . . . 166
7.4.1 General . . . 167
7.4.2 Comparison of Graphs with Stable and Unstable Partitions . . . 170
7.4.3 Predicting Partition Stability: First Approach . . . 174
7.4.4 Graphs with Unstable Modularity Optimal Partitions . . . 175
7.4.5 Predicting Partition Stability: Refined Approach . . . 183
7.4.6 Discussion . . . 184

8 Conclusion and Outlook . . . 187

Appendices . . . 189
A Fraction of Affected Nodes for Fixed Normalized Redundancy . . . 189
B Graph Symmetry Analysis Procedure . . . 190
B.1 Detailed Description of the Analysis Procedure . . . 190
B.2 Detailed Results . . . 197
C Dependency of Modularity and Density . . . 206
D Algorithms . . . 207
D.1 Create Orbit Partition from Generators . . . 207
D.2 Test Coarser-or-Equal for Partitions . . . 209
D.3 Decompose Set of Generators into Support Disjoint Sets . . . 210
E Karush-Kuhn-Tucker Conditions for Non-Linear Constrained Optimization . . . 211

List of Figures . . . 213

List of Tables . . . 215

Bibliography . . . 217

1 Introduction

1.1 Motivation

Symmetry is deeply rooted in daily human life. Right in the morning by looking into the mirror, two symmetries become apparent: (i) The reflection of the own face, which is literally mirrored, and (ii) the symmetry of the face itself along a vertical axis, which splits the face into two halves. These two very graphic examples of symmetry are by far not the only ones that can be recognized directly by observation. However, we claim both are not recognized explicitly, as humans are used to these symmetries due to their experience. In contrast, if the symmetry is "broken" (i.e. symmetry vanishes by adding/removing something so that the object becomes asymmetric), a human will become aware of the asymmetry. There is even evidence that high facial symmetry is of crucial importance for perceived attractiveness (Perrett et al., 1999; Scheib et al., 1999).

Considerations of preferring symmetry over asymmetry go back to the Stone Age. For instance, Reber (2002) discusses the question, why *Homo erectus* created symmetric stone tools as part of a discussion section of the archaeological findings (Wynn, 2002). He argues that the reason for this symmetry is unlikely due to a perceptual preference only, because symmetry preference was also observed for other species as well. As these other species (e.g. fishes) do not have a desire of fine arts, which could explain such a preference, Reber (2002, p. 416) comes to the conclusion that "[p]reference for symmetry is a more basic affective reaction than an evaluation based on aesthetic or symbolic value". Other ancient examples, as part of an introduction to symmetry in several branches of science, are illustrated by Weyl (1952). He shows images of a Greek statue (p. 7), of the design of a Sumerian vase of King Entemena (p. 8, see Figure 1.1), and—among other examples—symmetric floor patterns of an ancient Greek house (p. 11).

We motivate the occurrence of symmetry in diverse scientific disciplines in the following paragraphs. This overview is not aiming for an extensive literature review but to give ideas in what forms symmetry can appear. A separate section is dedicated to examples of symmetry in graphs (Section 1.2). Moreover, for the sake of convenience, we try to be as informal as possible, because formal definitions of symmetry and other foundations follow just after this chapter.

However, let us briefly give an informal description of symmetry groups. The main idea formalized by a symmetry group is that the combination of multiple symmetries is again a symmetry. Think again of looking into a mirror (see Figure 1.2 for an illustration of this example). The image you see of yourself is a reflection along the vertical axis (Figure 1.2a). Imagine that if you would take a photo of the mirror image (that is how you see yourself in the

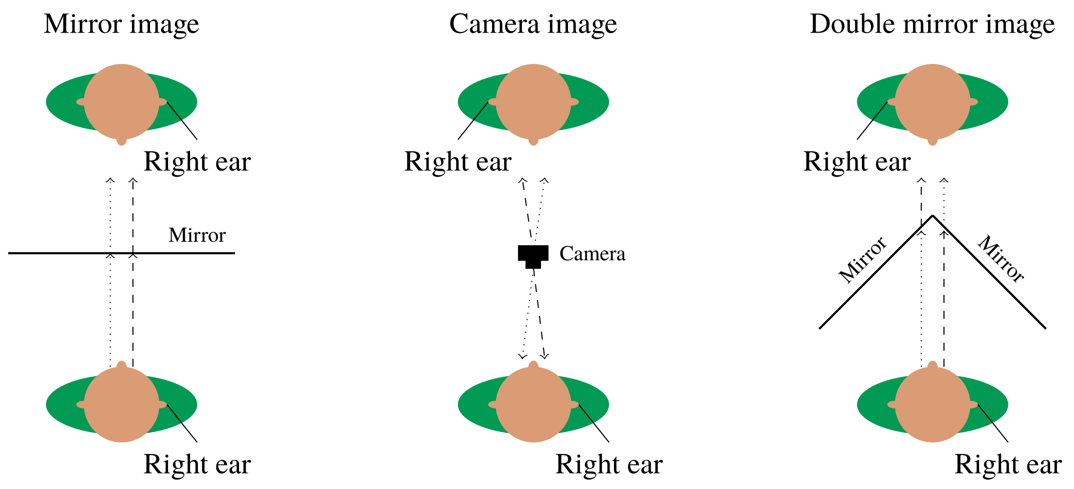
Figure 1.1: The vase of King Entemena[a], which is given as an example of ancient symmetry by Weyl (1952, p. 8). The engravings show a "lion headed eagle with spread wings [. . .] whose claws grips a stag".

[a] `https://commons.wikimedia.org/wiki/File:Vase_Entemena_Louvre_AO2674.jpg`, photographed by user "Jastrow", version "09:45, 24 June 2006", as of September 2018, licensed under public domain.

mirror) and a second frontal photo would be taken of you standing in front of the mirror (that is how someone else sees you; Figure 1.2b). Then your right ear on the photo of the mirror image is on the right side. On the photo taken of you directly, it would be on the left side. If there are two mirrors arranged in front of you as shown in Figure 1.2c, you would see the combination of the reflections each mirror implies. This means you see yourself the same as someone else sees you, which is of course identical to the image a camera would take of you. A symmetry group that captures this case contains exactly these two symmetries: The identity—which is how someone else sees you—and the reflection along the vertical axis. The combination of two reflections is again a symmetry, namely the identity. Combining identity and a reflection yields the reflection (like taking a photo of yourself in a mirror).

Natural Sciences Further publications that deal with symmetry of the human body are, for instance, articles by Hu et al. (2014), who investigate symmetry of the brain, and van Dongen (2018), who presents a study that had the goal to identify associations between bodily asymmetry and behavioral lateralization. Lateralization is the term for the asymmetric specialization of the brain for different tasks.

The technical term in biology for a symmetric reflection along one axis is *bilateral symmetry* and Shi et al. (2018) investigate the bilateral symmetry of leaves. To quantify the symmetry,



(a) Example of a reflection that occurs when someone looks into a mirror.

(b) Example of the "identity image" if a photo of a person is taken. Note that cameras have a built-in lens, which assures that not a mirrored image is taken.

(c) Example of the combination of two reflections that just gives the identity. The image is mirrored twice, so that the left part of the person in front of the mirror will be seen on the right side and vice versa.

Figure 1.2: Simple example of a symmetry group that contains an "identity" and a vertical reflection. The lower part of each subfigure corresponds to the reality, the upper part corresponds to the image under the given symmetry.

they develop a "simple indicator" that is an error measure, which sums up the differences of subdivided areas of a leaf along the symmetry axis. Consequently, the smaller the error, the higher the bilateral symmetry. The results of their analysis of the leaves of ten different plant species shows that bamboo plants (the investigation took place in China) have very symmetric leaves compared to others.

Certainly, symmetry not only plays a role in biology—though it is the science where symmetry can often be observed by the human eye—but also in other branches of science. Especially in physics, symmetry is crucial in many aspects as Feynman et al. (1963, Chapter 52) nicely illustrate. The abstract definition of it is that a physical phenomenon, situation, or experiment is transformed and the result stays the same, i.e. it is invariant (Feynman et al., 1963, p. 52–1). The known operations that apply to this definition are (Feynman et al., 1963, Table 52–1):

- Translation in space
- Translation in time
- Rotation through a fixed angle
- Uniform velocity in a straight line
- Reversal of time
- Reflection of space

- Interchange of identical atoms/particles
- Quantum-mechanical phase
- Matter/Antimatter

As a plausible example for translational symmetry, the authors explain that an experiment (represented by some “apparatus”) that takes place under the exact same conditions at another place or point in time gives exactly the same results, as the physical laws that influence the experiment are invariant with respect to these transformations. In a more recent publication, Gross (1996) sums up “[t]he role of symmetry in fundamental physics” (Gross, 1996, p. 14256). He especially elaborates on *gauge symmetries*, which are “formulated only in terms of the laws of nature”, in contrast to the “global symmetries”, which are defined on “transformations of a physical system” (all Gross, 1996, p. 14258). Due to Gross (1996), gauge theory provides the fundamental basis of the standard model of physics.

A more thorough examination on symmetries in classical physics (i.e. not quantum physics) is provided by Brading et al. (2003); Brading and Castellani (2007). They distinguish *symmetry principles* and *symmetry arguments*: The former describes the invariance of solutions that we mentioned above, the latter means the inference of consequences that emerge from a symmetric phenomenon. One statement, which can be derived thereof, is that symmetry is not broken without reason, so there must be an explicit cause for the symmetry to vanish. Brading and Castellani (2007) also elaborate on the connection of symmetry to (mathematical) group theory, which allows to formalize and describe the symmetry itself.

Philosophy of Science The deep rootedness of symmetry in physics can be seen by the fact that it is as present as decades and even centuries ago (e.g. Christodoulakis et al., 2018). However, not only natural sciences involve symmetry considerations as the next examples will show. Van Fraassen (1989) and Hon and Goldstein (2006) lift the consideration of symmetry to a more philosophical level. In his book’s third part, called “Symmetry as Guide to Theory”, van Fraassen (1989, p. 236) comes up with the slogan that “[p]roblems which are essentially the same must receive essentially the same solution” as requirement for symmetry. He furthermore relates the concepts “equivalence relation”, “partition”, and “group of transformations” to each other and considers them as similar. As these concepts are general, they also apply to the symmetry of graphs, which is introduced in Section 3.2. Nonetheless, the authors of both books also use many references to the literature from physics (partly including the ones cited above), which underlines the importance and long standing history of symmetry in physics.

Applied Sciences After we shed some light on symmetry considerations from the point of view of natural sciences, mainly physics and philosophy, we now turn towards applied sciences. Itai and Rodeh (1990) propose an algorithm that has the goal to break symmetry in a problem arising in the context of distributed computing. The problem statement is as follows: Given n

indistinguishable processors that are arranged in a ring structure so that each of them can send messages only to its predecessor and successor, then (i) choose one processor as a leader and (ii) determine n. This problem statement implies that the actual number of processors is unknown and that it is not possible to distinguish any two processors by some property. Choosing a leader means that all processors have the information which one of them is the chosen leader, i.e. the indistinguishability caused by symmetry is broken by the appointment of one arbitrary processor. Although this may sound not very hard to solve, there is a fundamental problem to it, as either the computation time is infinite or the result will be erroneous if a deterministic method is used (Itai and Rodeh, 1990, p. 61). It is important to note that a method that is able to solve the two subproblems stated above must not be executed by an independent observer but on every processor at the same time. Because of this, the overall problem setting is symmetric and the authors state in their conclusion that "[s]ince its conception, distributed programming had to face the problem of symmetry" (Itai and Rodeh, 1990, p. 85). As a consequence, if n is unknown in advance, there is no solution to this completely symmetric problem other than the one that may not terminate or is erroneous.

Another technical problem that has to deal with symmetries arises in the design of circuits, where an engineer arranges and connects several functional components. Often, components of the same type could be rearranged without changing the overall design structure. The logical functionality of the connected components can be transformed into a graph representation that preserves the symmetry of the circuit design. Darga et al. (2004) were inspired by this type of problem to construct an algorithm (`saucy`) that solves the graph automorphism problem, which is a solution to the symmetry problem. More details on the problem transformation and how the algorithm works is discussed in Section 3.4.3 of this thesis. We also mention this example in the next section, where we elaborate on symmetry of graphs.

An idea to exploit the existence of symmetry to compress (biomedical) images is presented by Bairagi (2015). Nearly perfect symmetry of many parts of the human body (like the brain) is the base of the proposed lossless compression algorithm. Instead of saving the complete image data, only half of it is saved. The split of the image follows the symmetry axis of the depicted object and additionally some differential information is saved that allows to recreate the original image. The (small) differences are a result of imperfect symmetry.

Arts, Aesthetics, and Architecture Symmetry also plays a role in arts and architecture. For instance, Wonka et al. (2003) present a design grammar that allows automatic architectural design (e.g. of buildings) by applying the derivation rules of the grammar. The used shape patterns define possible symmetry of elements (e.g. repeated occurrence of windows at the front of a building) and a control grammar distributes design decisions along the hierarchy of grammar derivations (e.g. to prevent different looking windows). Park (2001) analyzes actual architectural designs of houses (the floor plans of them) utilizing the mathematical framework of symmetry groups, which we will introduce in Section 2.4.

Darvas (2017) gives a brief historical review of interdisciplinary applications of symmetry with a strong focus on aesthetics and decorative arts. He first introduces the seven "frieze groups", which describe how an object can be repeatedly arranged along a straight line. In the rest of the article he reviews several concepts, mostly from natural sciences, that have an impact on aesthetics, e.g. the shape of crystals.

It is also noteworthy that many religious symbols are bilaterally and/or rotationally symmetric (Figure 1.3). Interpretations of the symbols in their religious context are beyond this thesis' scope and we solely refer to them as further examples of symmetry in a common context.

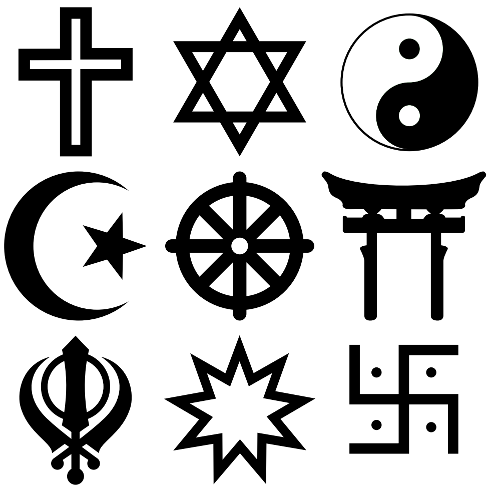


Figure 1.3: Religious symbols[a] that are symmetric. From top to bottom and from left to right: Christian Cross, Jewish Star of David, Taoist Taijitu (up to colors), Islamic Star and crescent, Buddhist Wheel of Dharma, Shinto Torii, Sikh Khanda, Bahá'í star, Jain Swastika.

[a] `https://commons.wikimedia.org/wiki/File:Symmetric_religious_symbols.svg`, created by users "Rursus", "Klem", and "Salix alba", version "15:24, 17 February 2011", as of May 2018, licensed under public domain.

Data Analysis and Economics In data analysis, Viana (2005, 2006, 2007) shows how canonical decomposition of the labeling of data can be applied. The idea behind this analysis framework is that there can be defined a symmetry relation on the set of labels that indexes some structured data. He presents several examples, one of them involves the distinction of Latin letters by their symmetry properties: Some letters (like **G** or **R**) cannot be transformed without changing them, others (like **A** or **V**) are vertically symmetric, some letters (like **C** or **D**) are horizontally symmetric, and some are both (like **O** or **H**), which also involves rotational point symmetry. For a sequence of letters, a data vector is connected to it by counting for how many letters each of the four symmetries applies. Another example (Viana, 2007, p. 327) is based on strings of DNA (deoxyribonucleic acid), which are encoded using the alphabet $A = \{A, C, G, T\}$ corresponding to the four nucleobases *adenine*, *cytosine*, *guanine*, and *thymine*. For sequences of length three (so called *triplets*), there exists a symmetry that shuffles the positions of the different letters. If every triplet is identified by some unique label, this symmetry also affects these labels.

Murtagh (2009) deals with “[s]ymmetry in [d]ata [m]ining and [a]nalysis” and presents non-trivial examples of symmetry in data, which is discussed below. The guiding thread is the existence of hierarchy, which “is fundamental for interpreting data and the complex reality” (Murtagh, 2009, p. 177). Hierarchy, e.g. as a result of hierarchical clustering, can be depicted with a so called dendrogram, which is a special type of tree. On a dendrogram there can be defined an ultrametric, which is defined simliar to a metric, but with the stronger triangle inequality $d(x, z) \le \max\{d(x, y), d(y, z)\}$ (instead of $d(x, z) \le d(x, y) + d(y, z)$). This means that all data points $y, z, \ldots$ that lie in another branch of the dendrogram for some given data point x have the same distance $d(x, y) = d(x, z) = \ldots$ and are, therefore, equivalent (independent of their distances based on the data attributes in an Euclidean space). An example that involves the often used *Iris* dataset is given by Murtagh (2009, pp. 181 f.). Furthermore, the author describes a p-adic encoding of a dendrogram and a distance definition for this, which leads to symmetry, and he shows how a hierarchy can be represented by permutations (see definitions in Section 2.4). Murtagh (2009, p. 195) concludes that symmetries, which “express observed and measured reality”, should be sought in data analyses.

Another relatively practical example for the utilization of symmetry in data analysis is given by Jabbour et al. (2013). They extend the famous `apriori` algorithm for the mining of frequent itemsets (Agrawal and Srikant, 1994) to involve symmetry-based pruning. A frequent itemset is simply a subset of a universe of items (e.g. products in a store) that is witnessed very often within an observed set of transactions (e.g. purchased carts in a store). To detect symmetry, the input data is transformed into a bipartite graph representation, where the automorphism group of the resulting graph captures the symmetry (details on graphs and automorphisms are defined in Sections 2.1 and 3.2). Items are equivalent if there exists a mapping on the set of transactions and on the set of all items so that the observed set of transactions stays the same. If equivalent items are found and one of them is recognized as infrequent, then, automatically, all other items must be infrequent, too. This allows to remove (prune) these items from the following iterations of the search procedure. Although the datasets used for their benchmarks do not contain many symmetries, Jabbour et al. (2013) encounter a speedup of up to 23 % compared to the original algorithm (mainly due to a decreased number of database scans).

Park et al. (2008) present an evaluation of three algorithms that are capable of recognizing symmetry in images. These algorithms are not limited to detect only symmetry of a whole image, but are able to find symmetric areas within the image. Examples involve the four “atomic symmetries” (Park et al., 2008, p. 2) of the two dimensional Euclidean space, namely *translation*, *rotation*, *reflection*, and *glide-reflection*. Example images shown by the authors contain, e.g., tires (rotational symmetry), faces and trees (reflection), or “wallpapers” (i.e. translational symmetry by shifting copies of an object). They come to the conclusion that the detection of symmetry in real-world images still has much potential to increase in accuracy, as less than 20 % of the symmetries were detected if multiple symmetries exist. The highest detection rate could be achieved for single reflections or rotations in artificial images.

How symmetry has an influence on voting in social choice theory is shown by Saari (1988). He applies the same theory (finite permutation groups) that underlies the symmetry of graphs to social choice, where permutations (reorderings of elements) can be used to transform one ranking into another. A ranking is just an ordered sequence of alternatives in descending order of preference, and a change in the ranking corresponds to a change in the voter's opinion. The author illustrates how such changes in opinion can lead to unexpected results, especially if not only rankings of all possible alternatives are considered but also rankings of subsets of the alternatives. An example of an unexpected effect is given in the introduction of the article: A majority vote comes to the result that milk is preferred over wine and beer. However, as milk is not available (this implies a change of opinion), suddenly beer is the new choice, although a majority of the voters actually prefers wine over beer.

Scientific Journals Before we present some motivating examples that are directly related to graphs (which are the main concern of this thesis), let us point out that there also exist several scientific journals dedicated to symmetry—either interdisciplinary or for a specialized field. The "Journal of Geometry and Symmetry in Physics"[1] first appeared in 2004 and published a total of 47 volumes as of the beginning of 2018. Since 2009, the interdisciplinary journal "Symmetry"[2] released ten volumes (one per year) with an increasing number of issues over the years (quarterly publications in the beginning, monthly since 2016). The oldest journal (in terms of its specialization expressed by the name of the title) is "Symmetry: Culture and Science"[3]. It is published by the "Symmetrion"[4], which is an "international institute [that] promotes interdisciplinary and holistic studies bridging different disciplines, science and art, and different cultures". The first volume goes back to 1990 and four numbers per year have been released until today.

1.2 Symmetry of Graphs

Symmetry of graphs is tightly coupled to the graph isomorphism and automorphism problems, whereas the latter is a special case of the former. The formal details on this, as well as some application examples, follow in Chapter 3. An informal definition, which relates to the relatively obvious geometric symmetry of two or three dimensional objects (like the symmetry of the human face in the very beginning of this thesis), is to say that a graph is symmetric (contains symmetry) if it is possible to map parts of the graph onto each other without changing its structure. Again, for formal details we refer to Section 3.2.

Purely mathematical publications are not discussed here, as the goal of this section is to point out the importance of graph symmetry in a scientific application context. The study of the automorphism group of graphs goes back to, e.g., Pólya (1937) and Frucht (1939), and has been

[1] `https://www.emis.de/journals/JGSP/`, as of May 2018
[2] `http://www.mdpi.com/journal/symmetry`, as of May 2018
[3] `http://journal-scs.symmetry.hu/`, as of May 2018
[4] `http://symmetry.hu/symmetrion/`, as of May 2018

part of active research in pure mathematics since then. The work of Pólya (1937) concentrates mainly on the (practical) combinatorial problem to enumerate the number of theoretically possible different chemical compounds of isomers. This is, however, equivalent to the number of automorphic drawings of a graph that models a chemical formula (Pólya, 1937, p. 210). Frucht (1939) answers the question whether there always exists a graph that has a given finite abstract group as its automorphism group. He shows that, in fact, there exist infinitely many such graphs, and that they are connected. Informally, this means that for each finite symmetry in general (which is described by a group) there always exists a symmetric graph with exactly the same symmetry.

Two motivational examples we have already discussed are—on the one hand—the one of Darga et al. (2004), who want to solve logical satisfiability problems that are derived from circuit design problems and can themselves be transformed into graph representations. These graphs are highly symmetric and knowing these symmetries allows to reduce the complexity of the satisfiability problem. On the other hand, Jabbour et al. (2013) create a graph representation of a frequent itemset mining problem that contains symmetry inherent in the data. To solve the symmetry problem, they use the method motivated and presented by Darga et al. (2004). Both of these problem transformations can be seen as examples of the application of the Theorem of Frucht (1939), which was just mentioned.

A survey on symmetry in interconnection networks[5] is given by Lakshmivarahan et al. (1993). Interconnection networks occur in the design of parallel computers, where each processor is described by a node and the edges represent data connections between them. The authors distinguish static and dynamic networks, but only cover the former, as the latter involves dynamical changes of the connections. They list symmetry as one property that makes a good network and investigate symmetry in general by discussing several Cayley graphs. A Cayley graph is a graph itself, which is a possible representation of a group (see permutation groups in Section 2.4) based on a generating set of the group.

The article of Zhao and Parhami (2018) goes in a similar direction: The authors propose a symmetry-based method that allows an efficient embedding of virtual networks into physical ones (i.e. computer networks that consist of actual hardware components). Therefore, the task is to find the best mapping of the virtual network structure to the physical network topology. Their method utilizes an intermediate symmetric graph (the symmetric *agency graph*), which acts as an auxiliary model between the source and destination networks. Due to the authors, this procedure simplifies the original embedding problem (Zhao and Parhami, 2018, p. 5).

Ben-David et al. (2006, section 6.3) notice in their article on clustering stability that "[s]ymmetry [l]eads to [i]nstability of [s]pectral [c]lustering". Spectral clustering is a method to identify clusters, which can be characterized by dense areas (submatrices) in a data matrix after

[5] Often, the terms *network* and *graph* are used interchangeably and we follow this convention. One difference between the terms is the degree of formality: A network is relatively informal and means the connection structure between entities in some way, whereas the term *graph* is the mathematical name that is tied to formal definitions. Consequently, graphs can be used to model networks.

reordering rows and columns, utilizing the eigenvectors of the (quadratic) matrix. However, symmetry of a matrix is expressed by a reordering of rows and columns that results in the same matrix. This operation is invariant concerning the eigenvalues and thus can lead to unstable situations. Spectral clustering is also applicable to graph clustering, as every graph can be represented by a quadratic adjacency matrix (see Section 2.1) and the symmetry of this matrix corresponds to the symmetry of the graph.

As an expression of the coherence of symmetry in aesthetics in the human perception and the representation of graphs, Welch and Kobourov (2017) investigate how the drawing of graphs (i.e. the graphical representation on a plane, see Section 2.1.2.3)—which is arbitrary up to a chosen layout—can be measured in terms of human perceived shapeliness. Their idea is based on Gestalt theory, which goes back to Wertheimer (1922, 1923, among others), and deals with these perceptions. The authors conducted a study where voters had to decide between two different drawings of the same graph which they liked better. The results are compared to three different measures (*Purchase* measure, *Klapaukh* measure, and *stress*), each considering layout symmetries in another way (e.g. mirror and rotational symmetry). The measures quantify a given drawn layout by a number. For the two former ones, a high value (at most one) reflects a large amount of layout symmetry, for the latter one, smaller values are desirable. An interesting finding (Welch and Kobourov, 2017, section 6.2) is the intransitivity of the three measures, which they attribute to a multi-faceted nature of symmetry: The *Purchase* measure has a higher agreement to the voters decision than the *Klapaukh* measure, and the *Klapaukh* measure is better than *stress*. However, the implication that then also the *Purchase* measure is better than *stress* is not true. It is important to note that not only symmetry of the graphs themselves, but also perceived symmetry of the drawing (i.e. even if graph symmetry is absent) plays a role for the voting. This result is in line with Gestalt theory, which suggests that humans perceive the entirety of an object before they sense the smaller parts of which the object is composed of (Welch and Kobourov, 2017, p. 341).

Wu and Antsaklis (2010) study the stability of complex control systems in which multiple agents interact with each other. This interaction at a certain point in time is represented by a graph. They analyze how many symmetric subsystems can be added to the complete system until it becomes unstable. Subsystems are equivalent if they have identical dynamics and interactions (Wu and Antsaklis, 2010, p. 198). The authors evaluate their stability considerations, e.g., on a cyclic interconnection structure (i.e. each agent interacts with its follower) with and without a center that allows bidirectional interaction (i.e. each circularly arranged agent interacts with the center and vice versa). The former case (with center)—which is quite similar to the interconnection network of independent processors discussed above—turns out to be unstable, whereas the latter case (without center) is stable.

All the above examples from the literature have in common that they see symmetry (i) as a desirable property or (ii) as a given “phenomenon” that has to be dealt with in a positive way. We close this section by shortly introducing some more general articles that regard symmetry in a more analytic environment.

Zenil et al. (2014) show the correlation of the automorphism group size, as a measure of graph symmetry (see also Section 4.1), with the *Kolmogorov complexity*, as a measure of structural complexity. Generally, Kolmogorov complexity can be used to measure randomness in a string of symbols and thus as a measure of compression. Strings that can be transformed in a lossless way (by using a different encoding) to be shorter than their input have a lower Kolmogorov complexity. Informally, Kolmogorov complexity is the length of the shortest program that is able to produce the symbol string as its result. The authors use a so called *block decomposition* method and apply it on the adjacency matrices of graphs. They are decomposed into non-overlapping submatrices and for each submatrix its occurrences are counted. As symmetry always implies a certain redundancy, the authors expect graphs with a decreasing complexity to have an increasing amount of symmetry. The expectations of Zenil et al. (2014) are met, as they clearly find a negative relation between Kolmogorov complexity and graph symmetry by empirically validating both measures for small and medium sized real-world graphs as well as for generated artificial networks (see also Section 2.2 for graph models). However, as also graphs with low symmetry and low complexity have been found, the authors conjecture that their measure (the block decomposition) does also capture other structural redundancies that are not related to graph symmetry.

Garlaschelli et al. (2010) and Garrido (2011) provide reviews on the topic of symmetry in (complex) networks. Garlaschelli et al. (2010) put a strong emphasis on the distinction of exact symmetry (which is discussed in this thesis) and stochastic symmetry. Garrido (2011) reviews several different symmetry conceptions in the literature. He also provides a definition of "fuzzy symmetry", which is—as stochastic symmetry—another form of weak symmetry. However, the level of detail of the content with a direct connection to graph symmetry that Garrido (2011) addresses is relatively vague. Both reviews have in common that they go into the details of several network models, which try to mimic certain properties of real-world/complex networks, and of community structure of networks, which is obtained through clustering. Nevertheless, possible effects of symmetry on clustering—as they are investigated in this thesis—are not addressed.

An effect of graph symmetry on clustering that is not discussed any further in this thesis is presented by Ball and Geyer-Schulz (2018b). The authors show that all measures that are capable to compare two graph partitions in terms of their structure are affected by graph symmetry. As a direct consequence, equivalent partitions cannot be recognized by these measures. They also prove that it is impossible to define such a comparison measure without taking the symmetry into account in general. As a solution to this problem, they also present how distance-based measures can be transformed so that they are defined on equivalence classes of partitions. Consequently, the transformed measures are invariant under the graphs' symmetry, i.e. they are not affected by the symmetry.

1.3 Scope of the Thesis

The previous sections have shown that symmetry in its entirety and symmetry of graphs in particular are part of past and current research. Especially in physics, symmetry effects are part of the models for a long time. Symmetry of graphs is a topic mainly rooted in discrete (applied) mathematics and has not gained much attraction in other research areas such as data analysis or social network analysis.

For instance, Newman (2010) thoroughly introduces many graph theoretic aspects that act as a basis for the analysis of networks, but he does not mention graph symmetry at all. Interestingly, the same author introduced the famous modularity measure, which is—nearly 15 years after publication—still the most widely used quality criterion for graph clustering (Newman and Girvan, 2004). This fact should not be understood as critique towards the author but as an evidence that there seems to be a gap between mathematical theory (which is well developed) and practical application (in a scientific sense). Aside from that, Newman (2003, p. 195) reviews that "[s]ociologists have concentrated on [. . .] *structural equivalence*", which he defines as nodes with exactly the same neighbors. But that is just the informal definition of symmetry. The author further argues that exact structural equivalence is unusual.

This thesis makes an effort to narrow this gap by providing deeper insights in both, theoretical and practical results on the existence of symmetry in graphs emerging from real observations and on the actual impact on graph clustering. The main focus lies on the result of graph clustering, which is a partition of the graph's nodes. Symmetry clearly has also an impact on the clustering algorithms; for instance, symmetry induces equivalent intermediate steps during the algorithms' execution. However, this issue will not be investigated further.

1.4 Outline

The content of this thesis spans eight chapters (including this introductory one) and five appendices. Chapter 2 is a comprehensive introduction of foundations, which are either needed for the upcoming content or are helpful to understand the complex relations of different concepts. We begin with the definition of graphs and focus on the two different representations that are most often found in the literature. One is based on matrices and the other is based on a set-theoretic definition. We also briefly discuss graph drawing, which is the graphical embedding of a graph on a plane (e.g. also a pencil drawing on a sheet of paper). We move on and review the most well known graph models and properties that are present in real-world graphs (i.e. graphs modeled as abstraction of real circumstances). Next, a definition of partitions is given in general and also particularly for graphs. The definitions are not only important in the context of graph clustering but also for graph symmetry. However, before we formalize graph symmetry in Chapter 3, an introduction to finite permutation groups is given, as they are the fundamental base of graph symmetry. After that, a definition of graph clustering is presented, which especially involves the presentation of clustering algorithms. Some of them are used in later chapters of the thesis.

The chapter ends with the introduction of entropy, again, in general, and entropy of graphs in particular.

In Chapter 3 different kinds of graph morphisms are presented with a strong focus on automorphisms, which are used to capture graph symmetry. All existing graph automorphisms form the automorphism group (which we informally called symmetry group in the previous sections) and it is a finite permutation group. We also extensively review different algorithms that are capable to compute the graph symmetry: First, efficient algorithms for special graphs are described and then, second, we review general algorithms that are applicable for any graphs. Third, a large part deals with algorithms that are actually implemented and can be used in practice. Most of them follow a systematic backtracking strategy that acts on a search tree and one of them (called `saucy`) is used in later chapters. As graph symmetry in terms of automorphisms is only one (relatively strong) symmetry concept, we also briefly discuss some other concepts.

The results of an empirical analysis of a large number of graphs concerning symmetry are presented in Chapter 4. These are in some way the baseline of all further considerations, as they justify the necessity for conducting research on this topic by showing the actual existence of symmetry in many graphs. In connection with the analysis, we first elaborate on measures that allow a quantification of graph symmetry. After that, we describe the analysis procedure itself, which involves the retrieval of graph data from an internet graph-data repository. An extended description of this procedure can be found in Appendix B. The chapter ends with the presentation and discussion of the results. The key finding is that of the about 1700 analyzed graphs over 74 % contain symmetries. Most of the content of this chapter is published (Ball and Geyer-Schulz, 2018a).

Chapters 5 and 6 introduce advanced theoretical concepts concerning graph symmetry and partition stability, which are then used in Chapter 7 for a second empirical analysis. In Chapter 5 the focus lies on graph symmetry and we begin by listing several reasons why graphs can be symmetric. A description for each reason is given as well, and we provide literature references to underpin our explanations. Next, we define local symmetry, which allows a quantification of graph symmetry, but from a different point of view as defined in Chapter 4. The idea that stands behind the given definitions is as follows: It makes a difference for potential symmetry effects if the existing graph symmetry affects one large part of the graph instead of several smaller parts. In the latter case, it is possible to decompose the symmetry into independent subsymmetries. The chapter ends with a definition of weak graph symmetry that is based on graph clustering.

The Chapter 6 deals with partition stability. In the beginning, we define how the symmetry of a graph affects the space of graph partitions, which is also the space of possible graph clustering solutions. The important part of it is that symmetry induces equivalence classes of partitions. Subsequently, we present three equivalent partition stability definitions and each of them allows a different point of view on the stability problem. Two of the definitions are combined in an algorithm that efficiently allows to test partition stability. After this, we prove for two special classes of symmetric graphs, which often function as building blocks for larger graphs, that their symmetry does not affect partition stability when modularity clustering is used (that is the clustering definition used in this thesis), at least theoretically. An entropy definition that treats graph symmetry in the light of dynamical systems ends the chapter. The definition allows to quantify the stability of a single graph partition as well as the uncertainty of the complete graph. As this measure is used in the next chapter, we also provide simplifications that allow an efficient computation.

The second last chapter can be seen as the climax that also gives this thesis its title. In Chapter 7 a second empirical analysis is described, which has the goal to investigate if graph symmetry actually impacts the clustering results. Additionally, we formulate some hypotheses that involve, e.g., the idea that partitions of graphs with a local symmetry should be less likely to be affected by this symmetry. The analysis is spread over several steps that build on each other. The most interesting findings are:

- Graph symmetry **does have** an effect on clustering.
- Locality of the graph symmetry strongly influences this possible effect.
- Partitions of highly symmetric graphs can be affected "accidentally", independent of the locality of the symmetry.

This thesis ends with a conclusion and an outlook in Chapter 8. It wraps up the results and points towards future research topics that emerge from our findings or are closely connected to them.

2 Foundations

This chapter introduces many basic definitions. At some points, special integer sequences are presented and we will then reference the excellent "Online Encyclopedia of Integer Sequences" (OEIS Foundation Inc., 2017), which originates from the original work of Sloane (1973).

By $\mathbb{N}$ we mean the set of natural numbers including zero, $\mathbb{B} = \{0, 1\}$ is the set of binary numbers, and $\mathbb{R}$ is the set of real numbers. The phrase "iff" stands for "if, and only if"; "w.l.o.g." means "without loss of generality".

2.1 Graph Theory

We define the mathematical concept of graphs in this section. Section 2.1.1 begins by distinguishing different types of graphs and we present them without any formal definitions, but simply by graphical visualizations. The formal definitions of the types follow in Section 2.1.2, where we show different representations of the abstract concept of graphs. All representations are equivalent but depict different points of view. After that, we present some properties of graphs and additional definitions in Section 2.1.3. We end this section on graph theory by presenting some classes of graphs in Section 2.1.4. Examples of the demonstrated concepts and definitions are given throughout the text.

2.1.1 Types of Graphs

A graph G is an abstract mathematical structure that consists of nodes and edges. A node in its pure mathematical definition is an object free from any interpretative meaning except from its existence in the defined graph. Edges are relations between pairs of nodes and their interpretation is not part of the mathematical definition, as well. This thesis only takes finite graphs into account and, therefore, the numbers of nodes and edges are both finite. The number of nodes is $n \in \mathbb{N}$ and the number of edges is $m \in \mathbb{N}$.

2.1.1.1 Simple Graphs

Simple graphs consist of n nodes and have between 0 and $n(n-1)/2 = \binom{n}{2}$ edges. This means that between any two nodes either an edge exists or not. Two nodes that are connected by an edge are called *adjacent*, an edge that is connected to a node is called *incident* (to this node). In simple graphs, only edges that are incident to two different nodes are allowed. An edge incident to the same node twice is called a *loop* and a graph without loops is *loop-free*. An example is given in Figure 2.1.

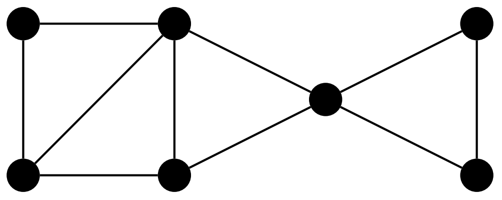


Figure 2.1: A simple graph with $n = 7$ nodes and $m = 10$ edges.

2.1.1.2 Directed Graphs

A directed graph (digraph) is defined similar as a simple graph, but each edge has a particular direction. This means it either directs to a node or away from it. Another difference to simple graphs is the maximum number of edges. It is $n(n-1)$, as between any two nodes either no edge exists, an edge from one to the other node exists, or two edges in either direction exist. Figure 2.2 shows an example. Depending on the graphical representation, two edges in either direction between two nodes can also be visualized as one edge with arrows at both ends or one edge without any arrows.

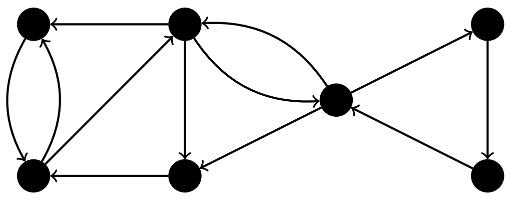


Figure 2.2: A directed graph with $n = 7$ nodes and $m = 12$ directed edges.

2.1.1.3 Weighted Graphs

A weighted graph is a simple graph that additionally has edge weights as shown in Figure 2.3. Which values the weights can have depends on the interpretation of the weights. Especially a weight of 0 depends on the context because it can either be interpreted as “no edge” or as “edge with weight 0”. An example is shown in Figure 2.3.

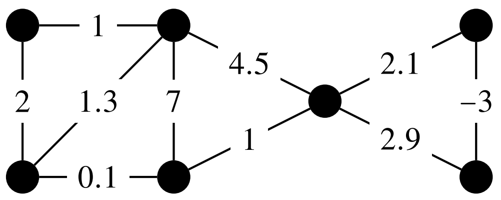


Figure 2.3: A weighted graph with $n = 7$ nodes and $m = 10$ weighted edges.

2.1.1.4 Multigraphs

Multigraphs are like simple graphs, but they can have multiple edges between pairs of nodes. Figure 2.4 shows an example. Depending on the representation, multiple edges can also be interpreted as weighted edges. The number of edges between two nodes is then simply interpreted as the weight of the edge.

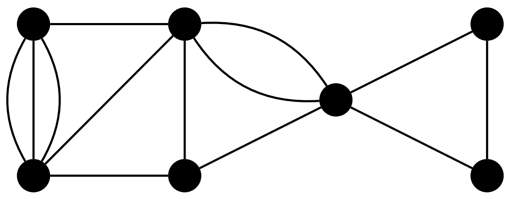


Figure 2.4: A multigraph with $n = 7$ nodes and $m = 13$ edges.

2.1.1.5 Discussion

The presented graph types above give only an overview of types that are often defined in the literature. It is sometimes hard to exactly distinguish different types, because some characteristics of graphs can also be interpreted as a property a graph can have. An example are loops. All types above must not have loops, but we did not present a “loop graph” type. However, in some cases it is important to allow loops in a graph, independent if it is a directed graph, a multigraph, and so on. On the contrary, the term “simple graph” refers exactly to the (informal) definition above, which means a simple graph must not have loops. If loops are possible, we simply speak of an undirected, unweighted graph without multiple edges that may has loops, instead of a simple graph with loops.

It also can be necessary to combine multiple graph types, e.g., a directed and weighted graph or a graph with directed and undirected edges. These examples show that it is unrewarding to only think of disjoint graph types. Instead, we believe it is more helpful to think of properties of graphs that can be combined. We go into more detail in Section 2.1.3.

Nonetheless, we want to mention another type of graph, which is different to all the examples and explanations above. A *hypergraph* is a graph that consists of nodes, but the relations between those nodes are not restricted to be defined on pairs of nodes. Such relations are called *hyperedges* and an hyperedge is a relation between any number of nodes in the graph.

2.1.2 Graph Representations

We present two types of representation, namely matrix-based and set-based representations. As for this thesis mostly simple graphs are of relevance, we mainly focus only on the definitions for simple graphs given the different representations. At the end of this section we describe briefly some other representations.

2.1.2.1 Matrix Representation

A graph G can be defined by a matrix $\boldsymbol{A} \in X^{n \times n}$. X is some “useful” number system, in the simplest form $X = \mathbb{B}$. The binary matrix $\boldsymbol{A}$ determines direct relations between the n nodes: If

$a_{ij} = 1$, there is an edge from the node determined in row i to the node determined in row j. Note that a_{ij} references the matrix entry in row i and column j.

For simple graphs, the relation $a_{ij} = a_{ji},\ \forall i, j = 1, \ldots, n$ must hold and $a_{ii} = 0,\ \forall i$, as loops are not allowed. $\boldsymbol{A}$ is symmetric if the graph is simple. If we want to represent directed graphs, only $a_{ii} = 0,\ \forall i$ must hold. The matrix $\boldsymbol{A}$ is called *adjacency matrix*, as it represents the node adjacencies. If only $\boldsymbol{A}$ is given, we cannot always distinguish if the graph is directed or not, as $a_{ij} = a_{ji},\ \forall i, j = 1, \ldots, n$ could hold even if the graph is directed.

We can compute the number of edges of a directed graph from the adjacency matrix as

$$m^{\rightarrow} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}. \tag{2.1}$$

If G is undirected, Equation 2.1 counts edges twice. Therefore, we compute m for undirected graphs as

$$m = \sum_{i=1}^{n} \sum_{j=i}^{n} a_{ij} \tag{2.2}$$

to count only the right upper triangle of $\boldsymbol{A}$ (note the index definition of the inner sum).

Example 1. *As one might have noticed, we tried to avoid giving nodes (and edges) any unique identifiers/ids, labels, colors, or whatever distinguishing property they could have. The reason is that, as, e.g., Biggs (1993, p. 7) states, a labeling of nodes is arbitrary. However, for this example and the following ones we introduce a labeling of the nodes of the graphs in Figures 2.1–2.4. This is necessary to have a clear connection between the visualization of the graphs and their corresponding matrix representations. The labeling is shown in Figure 2.5 and the label corresponds to the row and column indices of the adjacency matrices.*

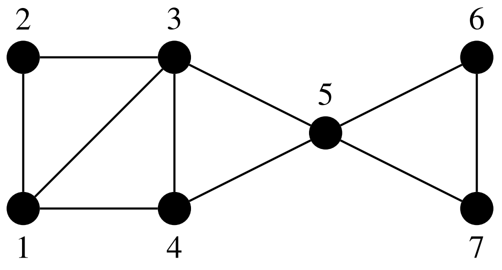


Figure 2.5: An arbitrary labeling of the nodes of the graphs in Figures 2.1–2.4.

The matrix

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.3}$$

is the adjacency matrix of the labeled simple graph shown in Figure 2.5. As it has no loops, the diagonal of $\boldsymbol{A}$ *contains only zeros. The number of rows and columns corresponds to the number of nodes* $n = 7$*;* $m = 10$ *is the sum of all matrix entries above the diagonal. Due to the symmetry of the matrix, the sums of the entries above and below the diagonal are equal.*

Similarly, the matrix

$$\boldsymbol{A}^{\rightarrow} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{2.4}$$

is the adjacency matrix of the directed graph shown in Figure 2.2 (after labeling the nodes). It is not symmetric along the diagonal and the number of edges is the sum over all of its entries, which is $m^{\rightarrow} = 12$*.*

If we choose $X = \mathbb{R}$, we can also denote weighted graphs in adjacency matrix representation and for $X = \mathbb{N}$ the same is true for multigraphs. Nonetheless, the information on how to interpret the matrix is important to prevent misconceptions, as $\mathbb{B}^{n\times n} \subset \mathbb{N}^{n\times n} \subset \mathbb{R}^{n\times n}$ holds. Loops can easily be represented by non-zero entries on the matrix' diagonal.

Example 2. *The matrix*

$$\boldsymbol{A}^{w} = \begin{pmatrix} 0 & 2 & 1.3 & 0.1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1.3 & 1 & 0 & 7 & 4.5 & 0 & 0 \\ 0.1 & 0 & 7 & 0 & 1 & 0 & 0 \\ 0 & 0 & 4.5 & 1 & 0 & 2.1 & 2.9 \\ 0 & 0 & 0 & 0 & 2.1 & 0 & -3 \\ 0 & 0 & 0 & 0 & 2.9 & -3 & 0 \end{pmatrix} \tag{2.5}$$

represents the weighted graph shown in Figure 2.3 and

$$\boldsymbol{A}^{m} = \begin{pmatrix} 0 & 3 & 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 2 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.6}$$

represents the multigraph in Figure 2.4.

Other matrix representations of graphs are the incidence matrix or the Laplacian matrix: The incidence matrix $\boldsymbol{D}$ is an $n \times m$-matrix where nodes are determined by rows and edges by columns (thus $\boldsymbol{D}$ is not necessarily symmetric). Biggs (1993, p. 24) defines $\boldsymbol{D}$ for the directed case as

$$d_{ij}^{\rightarrow} = \begin{cases} 1 & \text{edge in column } j \text{ directs to node } i \\ -1 & \text{edge in column } j \text{ directs away from node } i \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

whereupon the values for "directs to" and "directs away from" are chosen arbitrarily and thus can be swapped. For directed graphs this reduces to

$$d_{ij} = \begin{cases} 1 & \text{edge in column } j \text{ is connected to node } i \\ 0 & \text{otherwise} \end{cases} . \tag{2.8}$$

This definition also allows an easy representation of hypergraphs, as it is possible to describe the incidence of edges with more than just two nodes. Multigraphs can also be represented (directed and undirected). For weighted graphs or graphs with loops, the definitions in Equations 2.7 and 2.8 have to be modified.

Example 3. *The incidence matrix corresponding to the simple graph in Figure 2.1 is*

$$\boldsymbol{D} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{2.9}$$

with $\boldsymbol{D} \in \mathbb{B}^{7\times 10}$, *hence* $n = 7$ *and* $m = 10$. *The row indices correspond to the labels introduced in Figure 2.5 and each column represents an edge. Similarly,*

$$\boldsymbol{D}^{\rightarrow} = \begin{pmatrix} -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \tag{2.10}$$

is the incidence matrix of the directed graph shown in Figure 2.2. Each of the $m = 12$ *columns contains exactly one entry that is* 1 *and one that is* -1 *to represent the directions. For instance, the* *first column corresponds to the edge from node* 1 *to node* 2*, and the second column corresponds to the edge between the same nodes but in the opposite direction.*

The Laplacian matrix of a graph is defined as (e.g. Biggs, 1993, p. 27)

$$\boldsymbol{L} = \boldsymbol{\Delta} - \boldsymbol{A}, \tag{2.11}$$

where $\boldsymbol{\Delta}$ is the degree matrix of G. $\boldsymbol{\Delta}$ has the same dimensions as $\boldsymbol{A}$ but with non-zero entries on the diagonal only (as for the identity matrix $\mathbf{1}$):

$$\delta_{ij} = \begin{cases} \deg(i) & i = j \\ 0 & \text{otherwise} \end{cases}. \tag{2.12}$$

$\deg(\cdot)$ is the degree function, which depends on the directedness of the graph. If G is directed, one can distinguish between indegree and outdegree of a node. Indegree is the number of "incoming" edges, i.e. the number of nodes relating to (pointing to) the considered node

$$\deg_{+}^{\rightarrow}(i) = \sum_{j=1}^{n} a_{ji} \tag{2.13}$$

and outdegree is the number of nodes the considered node relates to

$$\deg_{-}^{\rightarrow}(i) = \sum_{j=1}^{n} a_{ij}, \tag{2.14}$$

respectively. In undirected graphs, the indegree and outdegree are the same, thus

$$\deg(i) = \deg_{-}^{\rightarrow}(i) = \deg_{+}^{\rightarrow}(i). \tag{2.15}$$

Example 4. *The degree matrix of the simple graph in Figures 2.1 and 2.5 results in*

$$\boldsymbol{\Delta} = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}. \tag{2.16}$$

The Laplacian is

$$\begin{aligned}
\boldsymbol{L} = \boldsymbol{\Delta} - \boldsymbol{A} &= \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 \\ -1 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 4 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}.
\end{aligned} \tag{2.17}$$

From this point of view, it is clear that any $n \times n$-Matrix can be interpreted as a (possibly weighted, directed, . . .) graph and this correspondence allows to use the linear algebra toolset on graphs. The analysis of graphs via its describing matrices is captured by spectral graph theory (e.g. Biggs, 1993; Chung, 1997; Brouwer and Haemers, 2012), which aims to analyze “graph related” matrices. Brouwer and Haemers (2012, p. 2) define the spectrum of a graph as the spectrum of its adjacency matrix $\boldsymbol{A}$ and the Laplace spectrum as the spectrum of the Laplacian $\boldsymbol{L}$. The spectrum is the set of eigenvalues of a matrix and its multiplicities. Spectra of graphs can be used to characterize graphs (Brouwer and Haemers, 2012, chapter 14) or as input for graph clustering algorithms (see Section 2.5.4).

2.1.2.2 Set Representation

There is another widely used notation for graphs, which is based on set theory (e.g. Godsil and Royle, 2001, p. 1). A graph G is defined by a 2-tuple (V, E) and the components correspond to the sets of object types a graph consists of: Nodes and edges. In correspondence to the matrix representations given above, the node set is

$$V = \{1, 2, \ldots, i, \ldots, n\} \tag{2.18}$$

and the node labels match the row indices of the according adjacency matrix $\boldsymbol{A}$. E denotes the edge set and for directed graphs

$$E^{\rightarrow} \subseteq V \times V \tag{2.19}$$

applies. This means an edge simply is a tuple of two nodes expressing the relation between both. Therefore,

$$a_{ij} = 1 \iff (i,j) \in E^{\rightarrow} \tag{2.20}$$

holds. For simple graphs, this can be relaxed to sets instead of ordered tuples, thus

$$E \subseteq \{\{i,j\} \mid i,j \in V,\ i \neq j\} \tag{2.21}$$

and

$$a_{ij} = a_{ji} = 1 \iff \{i,j\} \in E. \tag{2.22}$$

We can directly derive $|V| = n$ and $|E| = m$ for directed and undirected graphs. To shorten notation, we sometimes simply write ij ($\in E$) when we mean the edge between i and j. Occasionally we also write $V(G)$ (or $E(G)$) instead of simply writing V (or E) to emphasize that we mean the set of nodes (or edges) of the graph G.

When we want to express multigraphs in a set-based representation, the set of edges E must be replaced by a multiset that allows to contain the same object (an edge) multiple times. Weighted graphs cannot be expressed directly by this representation.

Example 5. *The labeled simple graph shown in Figure 2.5 is represented in set-based notation as $G = (V, E)$ with*

$$V = \{1,2,3,4,5,6,7\} \tag{2.23}$$

and

$$E = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{3,4\},\{3,5\},\{4,5\},\{5,6\},\{5,7\},\{6,7\}\}. \tag{2.24}$$

For the directed graph in Figure 2.2 the notation results in $G^{\rightarrow} = (V, E^{\rightarrow})$ with

$$E^{\rightarrow} = \{(1,2),(1,3),(2,1),(3,2),(3,4),(3,5),(4,1),(5,3),(5,4),(5,6),(6,7),(7,5)\}. \tag{2.25}$$

2.1.2.3 Other Representations

Another possible representation, which we already implicitly used in Section 2.1.1, is to simply draw a graph on a plane (called *graph drawing*). This representation sounds to be simple and intuitive, but graph drawing brings up a whole new bunch of problems, as one normally wants the visualization to “look good”. What that means is on the one hand quite subjective, on the other hand there are properties (e.g. planarity) and methods (e.g. Kamada and Kawai, 1989; Fruchterman and Reingold, 1991; Adai et al., 2004; Martin et al., 2011) to visualize graphs in an appealing manner. Quite intuitively, the computation of a sophisticated layout is as least as hard as the analysis of certain properties of the graphs themselves. This results from the arbitrariness of the positioning of the nodes on the plane.

Example 6. *One possible drawing of the simple graph G with $n = 7$ nodes and $m = 10$ edges was already given in Figure 2.5. The chosen layout is of course handmade and emphasizes the structure of G. In Figure 2.6, the same graph is drawn with another far more generic layout: The nodes are arranged in a circular manner, ordered by their arbitrary labels we have introduced.*

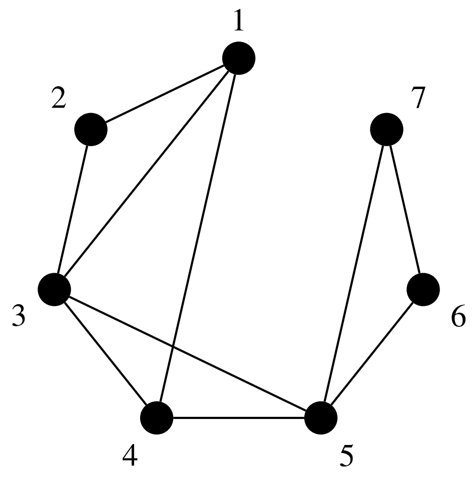


Figure 2.6: A circular layout of the same graph as in Figure 2.5.

Another representation of graphs is in terms of functions that describe the relations between the nodes or even properties of single nodes. For a simple graph, the function $e : V \times V \to \mathbb{B}$ describes the edge relations. This function can be replaced/modified for a weighted graph so that $w : V \times V \to \mathbb{R}$ describes weighted edges. The definition of the relation functions influences, e.g., if loops exist or not. Additionally, a function $c : V \to \mathbb{N}$ could represent a coloring of the nodes. This function-based representation is very flexible and allows multiple properties for a graph, each represented by a separate function.

2.1.3 Graph Properties and Further Definitions

So far, we have been quite vague on the distinction of "types" and "properties" of graphs and we clarify this now. The properties, like weights, directed edges, loops, and so on, are all *constructive properties*. That means a graph that has these properties has them *by definition*. For instance, a weighted graph is always weighted, even if all edge weights are either just one or zero. Every weighted graph is contained in the *nondenumerable* set $\boldsymbol{Gr}_w$, which contains *any* weighted graph with n nodes and m weighted edges. The *denumerable* set of simple graphs $\boldsymbol{Gr}$ is a subset of $\boldsymbol{Gr}_w$, as each graph with edge weights zero or one is also contained in the set of weighted graphs.

The properties we introduce in this section are not present by definition but depend on the actual graph, i.e. on the element $G \in \boldsymbol{Gr}$ (or $\in \boldsymbol{Gr}_w$, etc.). We focus on simple graphs, but the properties can also be defined for other graph types. The first property is the *mean degree* for every node of an undirected graph. Trivially, the sum of all degrees is

$$2m = \sum_{i=1}^{n} \deg(i) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} = 2|E|, \tag{2.26}$$

because we count each edge twice. From this follows the mean degree (see Newman, 2010, p. 134)

$$\overline{\deg} = \frac{2m}{n}. \tag{2.27}$$

The maximum of $\overline{\deg}$ is of course $n-1$, which is reached if every node is connected to every other node (see also Section 2.1.4). If that is the case,

$$\frac{2m}{n} = n - 1 \iff m = \frac{n(n-1)}{2} = \binom{n}{2} \tag{2.28}$$

holds, as we already motivated in Section 2.1.1. With this we can define another property, which is the *density* of a graph, and it is the fraction of actual and maximum possible edges (also Newman, 2010, p. 134)

$$\rho = \frac{m}{\binom{n}{2}}. \tag{2.29}$$

Example 7. *The average node degree of the simple graph shown in Figures 2.1 and 2.5 is*

$$\overline{\deg} = \frac{2 \cdot 10}{7} = \frac{20}{7}, \tag{2.30}$$

i.e. every node is adjacent to nearly three other nodes on average. The density of the graph is given by

$$\rho = \frac{10}{\binom{7}{2}} = \frac{10}{\frac{7\cdot(7-1)}{2}} = \frac{10}{21}. \tag{2.31}$$

Another important property of a graph is its *connectedness*. A simple graph is connected, if there exists a path from an arbitrarily chosen node to every other node in the graph. A path from the node i to the node j is a sequence of indices of the adjacency matrix

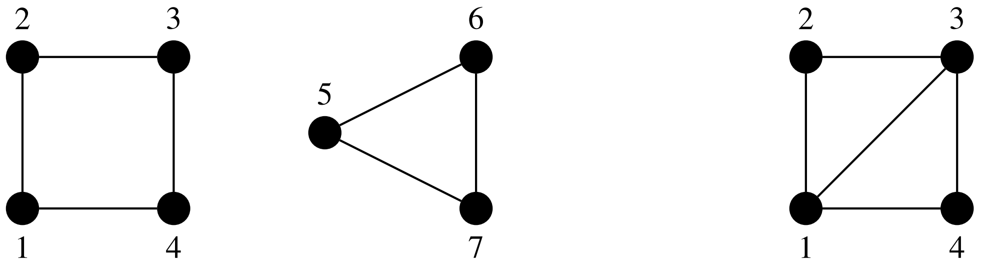
$$\text{path}(i, j) = i, k_1, k_2, \ldots, k_l, j, \quad a_{ik_1} = a_{k_1k_2} = \ldots = a_{k_lj} = 1. \tag{2.32}$$

If a graph is not connected (disconnected), there exists a reordering of the rows and columns of $\boldsymbol{A}$ so that each disconnected part of the graph is determined by a distinct sub-matrix, and the remaining row and column entries are all zero (as no edge to another part exists). Such reorderings are called permutations and the resulting matrix manifests a graph, which is isomorphic to the original one. We go into much further detail on this topic in Chapter 3.

Not a property but a convenient definition is the one of a *subgraph*. A subgraph G' of a graph G is defined by subsets $V' \subseteq V$ and $E' \subseteq E$. Godsil and Royle (2001, p. 3) distinguish between *spanning* and *induced* subgraphs. Spanning subgraphs have exactly the same node set $V' = V$

but a proper subset $E' \subset E$, which is the same as removing some edges from G. An induced subgraph is formed by a proper subset $V' \subset V$ and $E' = \{uv \in E \mid u, v \in V'\} \subset E$, which means V' induces G'. Alternatively, an induced subgraph can be created by a proper subset $E' \subset E$ together with the node set V' that contains all nodes that are adjacent by edges in E'.

Example 8. *The spanning subgraph of the simple graph shown in Figures 2.1 and 2.5 that is formed by the edge set* $E' = \{\{1,2\},\{1,4\},\{2,3\},\{3,4\},\{5,6\},\{5,7\},\{6,7\}\}$ *is shown in Figure 2.7a. A subgraph that is induced by the node set* $V' = \{1,2,3,4\}$ *is shown in Figure 2.7b.*



(a) A spanning subgraph of the simple graph shown in Figure 2.5. The graph is disconnected, as, e.g., no path exists from node 1 to node 5.

(b) A node induced subgraph of the simple graph shown in Figure 2.5.

Figure 2.7: Two examples for subgraphs.

Another interesting concept is the complement of a graph, noted as $\bar{G}$ and defined as $\bar{G} = (\bar{V}, \bar{E})$ with $\bar{V} = V$ and

$$\bar{E} = \{uv \mid u, v \in V,\ u \neq v,\ uv \notin E\}\,. \tag{2.33}$$

The complement of a connected graph is not necessarily connected again as we will see in an example in Section 2.1.4.

Example 9. *The complement of the graph shown in Figure 2.6 is shown in Figure 2.8.*

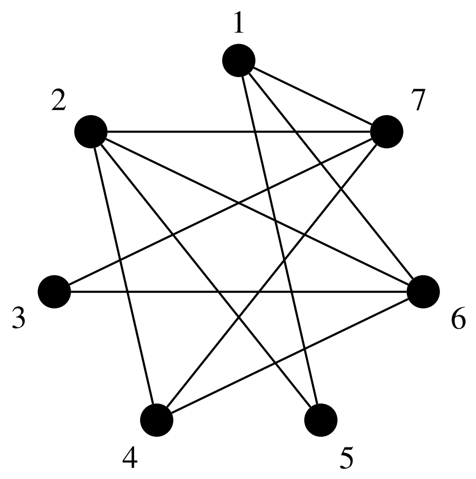


Figure 2.8: The complement graph of the graph in Figure 2.6.

2.1.4 Graph Classes

Efforts are made to classify graphs based on their properties and/or type. A comprehensive overview is provided by Brandstädt et al. (1999); de Ridder et al. (2010), but we only point out some simple and trivial classes here. Classes (or families) do not partition the space of all graphs, as a graph may belong to more than one class. Classes can also form hierarchies and the more specialized a class is (i.e. being of lower hierarchy), the more properties does a graph of this class fulfill.

Probably one of the simplest classes are the complete graphs. A complete graph has edges between all of its n nodes and is denoted K_n with

$$E(K_n) = \{uv \mid u, v \in V(K_n), u \neq v\}\,. \tag{2.34}$$

For the number of edges, one easily calculates $m = \binom{n}{2}$, as this is the total number of node pairs. The complement $\bar{K}_n$ is disconnected and also has a name (and forms a class for all n): The empty graph, as it has of course no edges at all. A complete subgraph $K_{n'}$ of a graph G is called clique.

Another class is characterized by its node degrees. A graph is called k-regular if each node has exactly degree k (for some k fixed). 2-regular graphs are somewhat special, as they are uniquely determined for n fixed and up to isomorphism (we will define the meaning of this phrase in Chapter 3). To construct a graph C_n where every node has exactly degree two, we can "pick" one node at random (say 1) and connect it with any of the remaining nodes $v \in V(C_n) \setminus \{1\}$. Without loss of generality, we connect it to 2 and now need a "partner" for this node again so that the degree is two. Let this node be 3. If we continue this procedure, we produce a chain of nodes, where each node has degree two, except for 1, where we started at, and k, where we paused at. If, for any $k > 1$, we connected k to any existing node j in this chain (except to 1 and without producing multiple edges), j would have degree three and we failed 2-regularity. If we connect k to 1 and $k < n$, both newly connected nodes would have degree two, but the created graph is not connected, as disconnected nodes remain. The only way to end the overall procedure is to continue the chaining up to $k = n$ and *then* connect n to 1. With this, the chain turns into a cycle and thus the name: Cycle graph C_n. For instance, Brandstädt et al. (1999, p. 17) do not name these two classes as such but call them "special graphs".

This brings us to the next class. A graph is called k-partite if the node set V can be divided into k disjoint subsets V_i ($i = 1, \ldots, k$) so that

$$\forall V_i : \{uv \mid u, v \in V_i\} \cap E = \emptyset \tag{2.35}$$

holds. That means only edges between nodes of different subsets are allowed but no "intra-subset" edges. For $k = 2$ these graphs are called bipartite. Clearly, each graph C_n for n even is bipartite and *every* graph is trivially n-partite (if $|V| = n$). Bipartite graphs can be used to model relationships between some entities that have different properties. An example are Petri-nets, which consist of *places* and *transitions*. Places may only be connected by directed edges to transitions and vice versa. Furthermore, a hypergraph $\mathcal{H} = (V, \mathcal{E})$ can be expressed as a bipartite graph G with $V(G) = V(\mathcal{H}) \cup I(\mathcal{E})$. I is a mapping that assigns each hyperedge in $\mathcal{E}$ to some node of G. All nodes in $V(\mathcal{H})$ that are connected by a hyperedge in $\mathcal{H}$ are connected to the corresponding node in $I(\mathcal{E})$. The concept of completeness can also be extended to k-partite graphs. Such a graph has edges between all pairs of nodes of all disjoint subsets of nodes. The notation is similar as already stated: For instance, $K_{s,t}$ is the complete bipartite graph of $n = s + t$ nodes and $m = s \cdot t$ edges.

A definition closely related to k-partite graphs is the chromatic number χ. It expresses how many colors are needed to color all nodes under the constraint that no two neighbors have the same color. The chromatic number is defined as

$$\chi(G) = \arg\min_{k} \{\forall uv \in E(G) : color_k(u) \neq color_k(v), \} \tag{2.36}$$

where $color_k(\cdot)$ is the function that assigns a node one of k available colors. If a graph has chromatic number $\chi(G)$, it is, therefore, $\chi(G)$-partite as no two nodes of the same color are adjacent by definition.

The last class we want to mention here are planar graphs. Planarity means that a graph can be drawn on a plane without any crossing edges (bent edges are allowed). Clearly, the graph in Figure 2.5 is planar.

2.2 Graph Models

In this section we want to give an overview of different models of graphs. A graph model is mostly determined by a constructive procedure that produces (normally given some parameters) an instance of a certain type of graph. The constructive definition normally allows for a wide range of analyses and the computation of indices, which distinguish the different models.

The main difference between the graph classes in Section 2.1.4 and the graph models described here is that the latter are statistical models, which try to characterize how a certain graph originates from a random graph generating process. That means the focus lies on the stochastic generation process instead of the separation into classes by strict properties (like the node degree). This does of course not mean there may not exist constructive algorithms for graphs belonging to a certain class, too. As a consequence, graphs that follow a certain model (or are created using a probabilistic constructive model) are described by statistical qualities as, e.g., the distribution of node degrees or the mean path length. All the models are based either on the random rewiring and/or addition of edges to an existing set of nodes and edges or on the successive random addition of nodes and edges to a given graph.

We only want to give a brief overview of the most well-known models. Newman (2003, 2010) and Albert and Barabási (2002) provide extensive surveys of such network models and their properties. We loosely follow the structure of Newman (2010) in the following subsections.

2.2.1 Properties of Real-World Networks

Before specific graph models are described, some properties of real-world networks are presented. An in-depth overview and analysis is given in (Newman, 2003) and (Newman, 2010, chapter 8). Those properties are of crucial importance, as they can and must be used to distinguish and classify different networks, depending on their originating domain of application. Moreover, it is helpful to have a basic understanding of the properties for the later analysis and argumentation concerning symmetries and their impact on clustering. The principal properties are

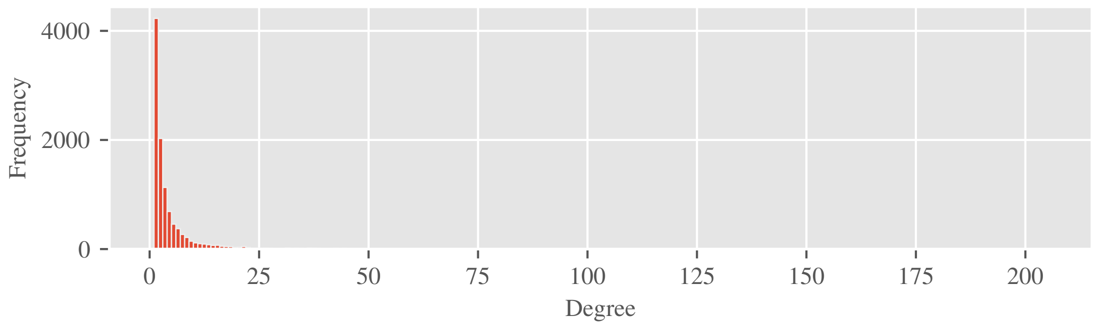
1. the existence of large components,
2. relatively short paths between any pairs of nodes,
3. the distribution of node degrees, and
4. the existence of modular community structures.

We will describe them next and/or in the following subsections.

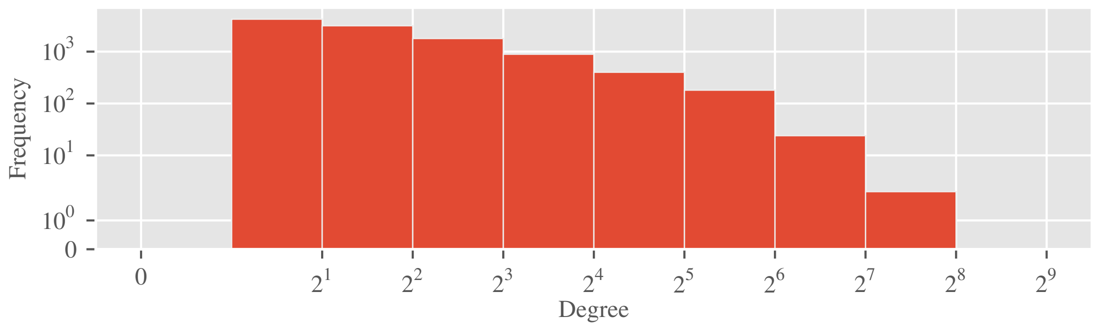
A requirement we already mentioned in Section 2.1.3 is the connectedness of the graphs we are interested in. From the clustering point of view this makes sense, as the search for modular substructures in the graph is pointless for already separated components (see the details in Sections 2.3 and 2.5). One could say that disconnected components are already naturally clustered. The requirement can also be justified by the actual existence of one large component in most real-world networks as Newman (2010, pp. 235 ff.) argues. He claims that situations that would result in many small disconnected components when modeled as a graph are normally not of interest for the analysis.

The second property—short paths between any two nodes on average—is motivated and discussed in Section 2.2.5, where a model that builds on this so called *small-world* effect is presented. However, the finding that stands behind this effect is that the average path length between two nodes roughly grows logarithmic in the number of nodes n. As a result, even in very large graphs it is likely that any two nodes are not too far away from each other.

Another major characteristic of a real-world network is its distribution of node degrees. Plotting the histogram of the degrees for a certain graph often shows a right-skewed and exponentially decreasing distribution. This means there are many nodes with a low degree and substantially less nodes with a high degree. The property of this distribution is called *power-law* (Newman, 2017) and a node u has degree k with a probability $\mathbb{P}\left(\deg(u) = k\right) \propto k^{-\alpha}$ ("$\propto$" means "is proportional to"). α is a parameter that usually takes values between 2 and 3. Because of this proportionality, graphs whose node degrees follow a power-law distribution are often referred to as "scale-free" since the slope of the distribution of degrees is independent of the graphs' size. The parameter can also be estimated from empirical data by using a maximum likelihood estimator (Clauset et al. (2009) and Newman (2010, p. 255)), the use of a linear regression on the log-log scaled data is not recommended. An example of the degree distribution of a real-world network is shown in Figure 2.9.



(a) The histogram of degrees with linearly scaled axes.



(b) The histogram of degrees with log-log scaled axes.

Figure 2.9: The degree histogram of the PGP network (Boguñá et al., 2004). The plot (a) shows the rapid drop of occurrences of nodes with high degree. Using a log-log scale in (b) shows a nearly linear dependency of the log-corrected data between degree and its frequency (at least in the range between 1 and 2^6), which is an indication for a power-law distribution.

Networks obeying a power-law distribution—at least approximately—often emerge from a growth process where more and more nodes and edges are randomly added to the graph over time. One constructive model is described in Section 2.2.3.

A third important property that is evident in real-world networks is a significant degree of clustering, i.e. groups of nodes exist that are somehow more connected to each other than to all other nodes in the graph. Measures of this phenomenon are clustering coefficients; a global version is presented in more detail in Section 2.2.2. Newman (2010, p. 265) furthermore defines a local clustering coefficient, which simply counts the relative connectedness of the neighbors of a node between each other: Let $N(u, G) := \{v \mid uv \in E(G)\}$ be the set of all neighbors of u in G, then the local clustering coefficient is defined as

$$\mathrm{Cl}(u, G) := \frac{|\{vw \in E(G) \mid v, w \in N(u, G),\ v \neq w\}|}{\binom{|N(u,G)|}{2}}. \tag{2.37}$$

Global clustering coefficients can be defined in a similar way by taking the fraction of the actual appearance of some pattern (i.e. a subgraph) and the theoretical maximum of occurrences of this pattern. One reason for a high degree of clustering, which is mentioned by Ravasz and Barabási (2003), is a possible hierarchical organization that is intrinsic to the graph data.

A much more specific approach is presented by Milo et al. (2002), who search for so called "motifs"—another term for the above "patterns". They detected that different motifs occur with different frequency in different types of networks (e.g. gene regulation networks, food webs, electronic circuits, the World Wide Web). To quantify the frequencies, each analyzed graph was compared to a randomized version of it that retained the distribution of node degrees. All networks were directed, but in the supplementary material also results for undirected networks are presented. Additionally, only motifs of a maximum of four nodes were taken into account. The authors only briefly describe their method used in the supplementary material. In the end, searching for motifs/patterns in a graph is equivalent to the subgraph isomorphism problem (see Section 3.1), which is not easy to solve.

Song et al. (2005) investigate complex networks by the means of the properties "scale-free" and "small-world". Interestingly, they find evidence for so called self-similarity, i.e. recurring properties, after successively scaling down the network by replacing groups of nodes that are connected under the condition to have a shortest distance to each other smaller than a certain maximum distance. Mislove et al. (2007) present an analysis of large online social networks. Their results confirm power-law, "scale-free", and "small-world" properties as well as a high degree of group formation.

2.2.2 Erdős-Rényi Random Graph Model

One of the first statistical network models is named after the Hungarian mathematicians Pá(u)l Erdős and Alfréd Rényi (ER-model in short). They prove several results on the connectivity of random graphs (Erdős and Rényi, 1957, 1960). An earlier study of random networks is by Solomonoff and Rapoport (1951).

Erdős and Rényi (1957) describe a random graph as a graph with n nodes and m edges, which is randomly chosen with equal probability from the set of all such graphs $\boldsymbol{Gr}_{n,m}$. This model—often called $G(n, m)$—is slightly altered to the model $G(n, p)$, which makes the study of the properties of those graphs easier (Newman, 2010, p. 400). Here, p is the probability for each of the $\binom{n}{2}$ edges to be chosen and, therefore, added to the graph. To approximate the model with fixed m, one can set $p = m/\binom{n}{2}$ to obtain a random graph with an expected number of m edges. Another direct consequence of this construction is a mean node degree of $c = (n - 1)p$. Some rather simple considerations (see Newman (2010, p. 401f.) for the details) allow to conclude that the node degrees of $G(n, p)$ follow a binomial distribution, which, for large n, can be approximated by a Poisson distribution.

Random graphs following this rather simple model are widely studied, but are, however, not suitable to describe networks that are observed in reality (real-world networks). The main reason for this is the absence of a clustering structure in random graphs, which can be quantified by the so called clustering coefficient (Newman, 2010, pp. 198 ff., 262 ff., 402 f.): It is basically a measure of transitivity of node triples in a graph and several definitions exist. The most common one is the quotient between the number of triangles and the number of connected triples. The set of triangles is intuitively defined as

$$\mathrm{Triangles}(G) := \{\{u, v, w\} \subseteq V(G) \mid uv, uw, vw \in E(G)\} \tag{2.38}$$

and the set of connected triples as

$$\mathrm{Triples}(G) := \{(u, v, w) \mid u, v, w \in V(G),\ uv, vw \in E(G)\}\,. \tag{2.39}$$

The clustering coefficient is then

$$\mathrm{Cl}(G) := \frac{3 \cdot |\mathrm{Triangles}(G)|}{|\mathrm{Triples}(G)|}. \tag{2.40}$$

The factor 3 is due to the fact that each triangle implies three connected triples. So $\mathrm{Cl}(G)$ is a relative measure of actually existing transitive relations (u is connected to v and v is connected to w implies u is connected to w) in contrast to the number of maximal possible transitive connections. Newman (2010, p. 199, footnote 22) mentions that the origin of the term "clustering coefficient" has nothing to do with graph clustering (see Section 2.5) without giving a reference. We believe, however, that it has a connection to it: Because the higher the value of C, the more "clustered" is the network in a sense that a high level of transitivity means that many nodes are highly connected to each other. That is, at least partially, a good definition of what a cluster is.

Albert and Barabási (2002, p. 59, figure 9) visualize the discrepancy between the clustering coefficients observed in real-world networks and $\mathrm{Cl}(G(n, p))$, the (expected) clustering coefficient of a graph obtained by the ER-model (ER-graph). Due to the construction of ER-graphs follows

$$\mathrm{Cl}(G(n, p)) = \frac{3 \cdot \binom{n}{3} \cdot p^3}{\binom{3}{1} \cdot \binom{n}{3} \cdot p^2} = p \tag{2.41}$$

with p^3 (p^2) the probability that three nodes form a triangle (connected triple), $\binom{n}{3}$ the number of subsets of length three, and $\binom{3}{1}$ the number of arrangements of three nodes in a connected triple (up to the node labels): (u, v, w), (u, w, v), and (v, u, w).

Yet, other more general random graph models exist and they are based on using different degree distributions, which allow a very precise modeling of the structure of (parts of) graphs (see especially Newman, 2010, chapter 13). The creation procedure is mostly similar to the ER-model, but it respects the different node degrees so that the specified distribution (or sequence of fixed node degrees) is obeyed. We do not go into further detail here, as it is a completely new topic on itself, which does not add anything to the basic understanding of random graph models.

Examples for applications of random graph theory are given by Kang and Petrášek (2014). They mention *phase transition* of, e.g., molecules like water. The aggregation states are solid, liquid, and gas. A random graph can be used to model the possible percolation of a liquid through the material. Bonds between molecules are represented by edges and a large parameter value p results in a very densely connected random graph, which corresponds to a solid or liquid aggregation state. Conversely, a small value of p results in a graph that contains many disconnected components, which corresponds to a gas. A liquid can only percolate through two parts of a material if there is no bond between them. Other applications mentioned by Kang and Petrášek (2014) are social sciences, "man-made" networks, life sciences, and brain networks.

2.2.3 Preferential Attachment

In contrast to pure random graph models, which are constructed from a fixed number of nodes and a fixed number of edges or an edge probability, the preferential attachment model describes evolution over time by the addition of new nodes that are connected to existing nodes following a non-uniform distribution (Barabási and Albert, 1999). The Barabási-Albert model is effectively a special case of the much older but less known model of Price (1965, 1976); however, the former is the one most often referred to (Newman, 2010, p. 500).

The idea of both models is to add a node that will be connected to a subset of the existing nodes, whereas the probability to connect an existing node with the newly added node is proportional to the degree of the existing node. This is motivated by Price (1965) as follows: Consider a newly published scientific paper that cites a subset of the existing ones, how likely are citations of popular and often cited papers in contrast to less known ones?

The creation algorithm starts with an initial number of nodes n_0 (often $n_0 = 2$) and in each iteration, one new node is added to the graph. It is connected to c of the existing nodes and the probability for an existing node u to be chosen is proportional to its degree $\deg(u)$. This means that nodes with a higher degree are more likely to be chosen. The probability that a node in the graph has degree k is given by Newman (2010, p. 502):

$$p_k(c) = \begin{cases} \frac{2c(c+1)}{k(k+1)(k+2)} & k \geq c \\ 0 & \text{else} \end{cases} . \tag{2.42}$$

In the limit, $p_k(c) \propto k^{-3}$ holds, which means the model creates graphs whose node degree distribution follows a power-law with $\alpha = 3$.

There exist several extensions of this model in terms of, e.g., vanishing nodes/edges or non-linear preferential attachment in the node degree. Newman (2010, pp. 514 ff.) provides information on these, but we do not go into further detail on this topic. Our purpose of presenting several graph models is to give an understanding of how practically occurring networks are composed.

2.2.4 Copying Nodes

Another interesting model (or class of models, as several similar forms exist) is to simply copy an existing node with all its connections—sometimes involving an error. Even without the exact graph symmetry definitions, which follow in Section 3.2, it is clear that an exact copy of a node represents a symmetry, as both nodes, old and new, have the same connectivity structure within the graph. Furthermore, these models also generate graphs whose node degree sequence follows a power-law, but with a different structural arrangement of the edges (Newman, 2010, pp. 537 ff.).

This behavior is also described by Chung et al. (2003), who present a node duplication model for biological networks. It is noticeable that their model produces networks whose node degrees follow the power-law but with an exponent $\alpha < 2$, which contradicts the often made statement that $2 \leq \alpha \leq 3$ normally holds (see Section 2.2.1 and e.g. Newman (2010, p. 258)). But as Chung et al. (2003) state in their discussion, this gives evidence for different network structures between domains of application.

2.2.5 Small-World Model

The small-world model goes back to the famous experiment by Milgram (1967), who gave a sample of people in the United States the task to pass a message to a randomly chosen target person that was only known by name and address. The restriction was to only pass the message to a person from the circle of acquaintances that seemed to be the one most likely to have the target person as an acquaintance themselves. The astonishing result was that the median amount of intermediates between the person who was given the message first and the target person was only five.

The interconnectedness between all nodes in a (connected) graph can be quantified by the mean distance between all nodes

$$l := \frac{1}{n} \sum_{u \in V} \left(\frac{1}{n-1} \sum_{v \in V,\ u \neq v} d(u,v) \right). \tag{2.43}$$

The inner weighted sum is simply the average distance from a node u to all other nodes in the graph and $d(u,v)$ is the length of the shortest path(s) between two nodes. Therefore, a small value of l indicates the existence of average shortest paths between all nodes in the graph.

Watts and Strogatz (1998) present a constructive model that starts with a cycle graph C_n. Then, each node becomes directly connected to its $k = c - 2$ nearest neighbors, and, as a result, the graph becomes c-regular. The clustering coefficient of this graph G_{WS} is $\mathrm{Cl}(G_{\mathrm{WS}}) = \frac{3(c-2)}{4(c-1)}$ (Newman, 2010, p. 554), which tends to $\frac{3}{4}$ for larger c. In contrast to the original model, Newman and Watts (1999) continue to add edges between randomly chosen pairs of nodes, instead of randomly rewiring existing edges. These edges are called “shortcuts”, as they normally provide a shorter path from the nodes near one end of the edge to nodes near the other end of the edge. They find that for relatively small p, which represents the probability that an edge is added, the clustering coefficient stays quite high while the mean distance between all nodes decreases significantly. However, the model does not produce a degree distribution that follows a power-law. Furthermore, for fixed c and p, the average distance between all nodes in the graph increases only slowly (by order $\ln(n)$) if n is increased.

2.3 Partitions

We will define a partition as a set of non-overlapping subsets of some set. Other concepts, such as overlapping or fuzzy sets, are not the focus of this work and will not be discussed in the following.

2.3.1 General Definition

A partition P is defined on a finite set S ($|S| = n > 0$) as follows:

$$\bigcup_{C\in P} C = S \tag{2.44}$$

$$C \cap C' = \varnothing \quad C \neq C',\ \forall C, C' \in P \tag{2.45}$$

$$C \neq \varnothing \quad \forall C \in P \tag{2.46}$$

As S is finite, so is P. Informally, P is a set of subsets of S; therefore, any axioms defined on sets hold. The cardinality $|P| \in \mathbb{N} \setminus \{0\}$ is the number of elements of the partition. For the general case, $C \in P$ is called a cell. Throughout the rest of this thesis this definition is meant when we talk about partitions if not stated otherwise. Sometimes a shorter notation besides the "correct" set-wise one is useful, we abbreviate $\{\{a, b, \ldots\}, \{c, d, \ldots\}, \ldots\}$ by $a, b, \ldots|c, d, \ldots|\ldots$ or even $ab\ldots|cd\ldots|\ldots$ if the meaning is clear.

Let $a_i := |C_i|,\ C_i \in P,\ \forall i = 1, \ldots, |P|$ be the associated set cardinalities. Then the type of a partition is the partially ordered tuple $\left(a_{i_1}, a_{j_2}, \ldots \mid \forall k \leq l :\ a_{i_k} \geq a_{j_l}\right)$. By additionally defining $\#x := |\{C \in P \mid |C| = x,\ x \in \mathbb{N}\}|$ as the number of cells of cardinality x, we can write the partition type in a shorter way as $\left(b_1^{\#b_1}, b_2^{\#b_2}, b_3^{\#b_3}, \ldots \mid (b_i \in \{|C| \mid C \in P\}) \wedge (\forall i : b_i > b_{i+1})\right)$. Of course the invariant $\sum_{b\in\{|C||C\in P\}} \#b \cdot b = |S|$ holds.

What we call type of a partition is better known as the partition of an integer, as an integer $n > 0$ can be decomposed into integers $a_i > 0$ with the associated composition operation "+". The partition number $p(n)$ (Andrews, 1998, pp. 1 f.) determines in how many different ways the integer n can be decomposed, and, analogously, how many types exist to partition a set of length $|S|$ (OEIS Foundation Inc., 2017, sequence A000041).

The number of decompositions of a set S into partitions of cardinality $|P| = k$ is given by the so called Stirling number of the second kind (OEIS Foundation Inc., 2017, sequence A008277):

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} i^n, \quad 1 \leq k \leq n. \tag{2.47}$$

Consequently, summing these numbers over all k results in the number of possible partitions of S (OEIS Foundation Inc., 2017, sequence A000110), which is

$$B(n) = \sum_{k=1}^{n} S(n,k). \tag{2.48}$$

It is called the Bell number and it increases rapidly (e.g. for $n = 20$ follows $B(20) =$ 51,724,158,235,372). More details on the theory of partitions of sets are given by Andrews (1998, pp. 214 ff.).

$\boldsymbol{P}(S)$ denotes the set of all partitions of S and it can be associated with a partial order "$\leq$" where $\forall P, Q \in \boldsymbol{P}(S)$:

$$P \leq Q \iff \bigcup_{C \in Q} \{C \cap C' \mid C' \in P,\ C \cap C' \neq \varnothing\} = P. \tag{2.49}$$

Informally, when $P \leq Q$, each cell of P is a subset of exactly one cell of Q and P is *finer* than Q (Q is coarser than P). This also implies (a bit counterintuitive) $|P| \geq |Q|$. Furthermore, there exists a sequence of unions of sets (cells) of P that results in Q. This sequence is not necessarily unique. The set of partitions in $\boldsymbol{P}(S)$ together with the partial order (a *poset*) can be represented by a simple graph $H = (\boldsymbol{P}(S), E)$ where an edge exists between partitions P, Q (the nodes) iff (w. l. o. g.) $P \leq Q$ and joining exactly two cells in P results in Q. When this graph is drawn on a plane, starting with the partition of single elements (corresponding to $S(n,n)$ at the bottom) and then successively adding partitions that are adjacent to this partition, the graph is called a *Hasse diagram*. Each "level" $l = 1, 2, \ldots, n$ in this diagram corresponds to exactly those partitions that are counted by $S(n, n-l+1)$. Each pair of consecutive levels induces a bipartite subgraph, as, of course, for all partitions P, P' on the same level neither $P \leq P'$ nor $P' \leq P$ holds. An example is shown in Figure 2.10.

2.3.2 Graph Partitions

A partition of a graph is normally defined on its node set V, although it could be defined on the set of edges, too. We call the partition $P_\perp := \{\{v\} \mid v \in V\}$ the singleton partition of a graph and conversely $P_\top := \{V\}$ the trivial partition. These partitions are the only ones having the partition types (1^n) and (n^1), respectively.

A partition P of a graph's node set always induces another graph, called *quotient graph*, which has P itself as node set and there exists an edge between two cells of P if there is at least one edge between two nodes in either of the cells. If there exists an edge between nodes of the same cell, this node in the quotient graph has a loop. A special version of this type of induced graph is defined and used in Section 5.3.

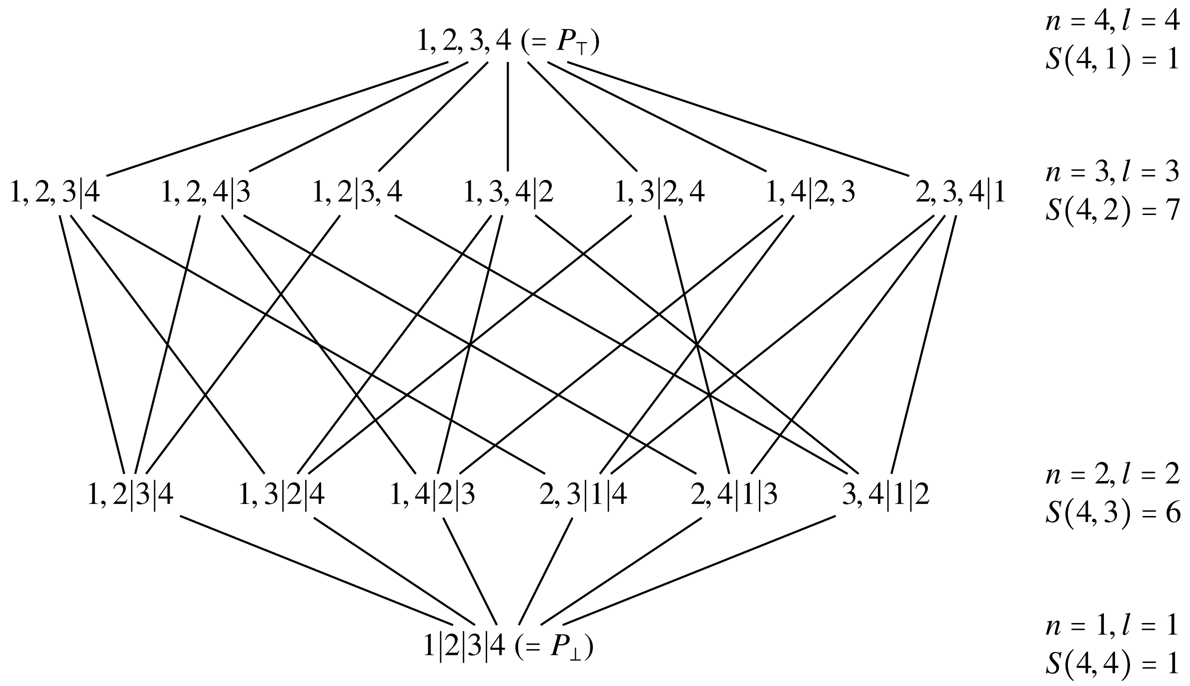


Figure 2.10: Hasse diagram for $\boldsymbol{P}(\{1, 2, 3, 4\})$ and the corresponding Stirling numbers (second kind) per level. Each path from the node at level 1 to level 4 (or n in general) is called a join path. The total number of partitions of four objects is $B(4) = 15$. All partitions on level 2 have type $(2, 1^2)$, on level 3 some have type (2^2), some $(3, 1)$.

2.4 Finite Permutation Groups

In this section we introduce the necessary concepts of permutations and permutation groups, which will help us to understand how graph automorphisms (Section 3.2) work. The explanations and notations closely follow Wielandt (1964). Another well written introduction is the "Background Material" chapter of Holt et al. (2005, Chapter 2).

2.4.1 Permutations

A permutation is a function defined on a set Ω that maps an element of Ω to any other element of Ω (Wielandt (1964) speaks of "points" instead of elements). We restrict Ω to be finite and call the number of elements $|\Omega| = n$ *cardinality* (Wielandt (1964) calls it *length*). Let $p : \Omega \to \Omega$ be a permutation. We denote the application of p on α by α^p. Then p must have the following properties:

1. $\forall \alpha \in \Omega : \alpha^p = \beta \in \Omega$ (p maps every element of Ω) and
2. $\nexists \alpha \in \Omega : \beta^p = \gamma^p = \alpha, \beta \neq \gamma$ (the mapping is unique, different elements are not mapped to the same element).

Thus, a permutation is a bijective function and Wielandt (1964, p. 1) simply calls it a “one-to-one mapping”. The above definition also allows for elements to be mapped onto themselves ($\alpha^p = \alpha$) and we call such elements *fixed* by p. A permutation p can be explicitly written down as

$$p = \begin{pmatrix} \alpha & \beta & \gamma & \dots \\ \alpha^p & \beta^p & \gamma^p & \dots \end{pmatrix} \tag{2.50}$$

or in a compressed cycle notation as

$$p = (\alpha\, \alpha^p\, (\alpha^p)^p \, \dots)\, (\beta\, \beta^p\, (\beta^p)^p \, \dots) \dots \tag{2.51}$$

of subsequent mappings. The order of the different cycles (if any) is arbitrary as well as the first element of a cycle. The last element of a cycle maps to the first element. Often the natural ordering (if any) of elements is used when writing permutations.

Example 10. *Let*

$$\Omega = \{1, 2, 3, \dots, 10\} \tag{2.52}$$

be a set of objects. A possible permutation is

$$\begin{aligned} p &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 10 & 8 & 4 & 2 & 3 & 9 & 5 & 7 & 6 \end{pmatrix} \\ &= (1)(2\ 10\ 6\ 3\ 8\ 5)(4)(7\ 9). \end{aligned} \tag{2.53}$$

The two elements $1, 4 \in \Omega$ *are fixed by p.*

The application of more than one permutation on an element $\alpha \in \Omega$ (whether the same or different permutations are used) is written as

$$(\alpha^p)^q = \alpha^{pq}. \tag{2.54}$$

pq is the catenation of the two permutations

$$pq = p \circ q, \tag{2.55}$$

and we keep to Wielandt (1964) by applying the leftmost permutation first (i.e. $\alpha^{pq} = q(p(\alpha))$ in a functional notation). The catenation r of p and q maps each element $\alpha \in \Omega$ directly to α^r, instead of first applying the map $\alpha \mapsto \alpha^p = \beta$ and then applying the map $\beta \mapsto \beta^q = \gamma$. This means $\alpha \mapsto \alpha^{pq} = \gamma$ is equivalent to $\alpha \mapsto \alpha^r$ and, as a consequence, $r = pq$ is itself a permutation. The k-times catenation of a permutation with itself is written as

$$p \circ p \circ \dots = p^k. \tag{2.56}$$

As Ω is finite, of course, also the number of possible different permutations on Ω is finite. A special permutation is the one that maps each element onto itself (fixes each element). It is called *identity permutation* and written as

$$\mathbf{1} = (\alpha)(\beta)(\gamma)\ldots = (). \tag{2.57}$$

Cycles of permutations of length one are normally omitted for brevity of notation, and for the identity permutation we will simply use empty parentheses instead of writing nothing. Note that "**1**" is the *function symbol* we use to denote the identity permutation in general, and "()" is the actual identity permutation written in cycle notation. The *support* of a permutation p is the set of non-fixed elements

$$\operatorname{supp}(p) := \{\alpha \mid \alpha^p \neq \alpha\}. \tag{2.58}$$

Example 11. *We continue Example 10. By omitting the cycles of length one, the permutation* $p = (1)(2\ 10\ 6\ 3\ 8\ 5)(4)(7\ 9)$ *can be abbreviated as* $p = (2\ 10\ 6\ 3\ 8\ 5)(7\ 9)$. *The catenation* $p \circ p = pp = p^2$ *is*

$$\begin{aligned} p^2 &= (2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ (2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \\ &= (2\ 6\ 8)(3\ 5\ 10). \end{aligned} \tag{2.59}$$

For instance, $(2^p)^p = 10^p = 6 = 2^{p^2} = 2^{pp}$ *shows the equivalence of the maps* $2 \overset{p}{\mapsto} 10 \overset{p}{\mapsto} 6$ *and* $2 \overset{p^2}{\mapsto} 6$. *The support of p is* $\operatorname{supp}(p) = \{2, 3, 5, 6, 7, 8, 9, 10\}$.

Every permutation p induces a permutation matrix $\boldsymbol{P} \in \mathbb{B}^{n \times n}$ with the entries

$$p_{ij} = \begin{cases} 1, & i^p = j \\ 0, & \text{else} \end{cases}. \tag{2.60}$$

Given an $n \times n$ matrix $\boldsymbol{M}$, $\boldsymbol{MP}$ permutes the columns, $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{M}$ permutes the rows of $\boldsymbol{M}$. In linear algebra, a permutation matrix determines a base transformation of the coordinate system, thus permutation matrices are orthogonal, i.e. $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{P} = \mathbf{1}$, which induces $\boldsymbol{P}^{\mathrm{T}} = \boldsymbol{P}^{-1}$. Interpreting $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{P}$ as $\boldsymbol{P}^{\mathrm{T}}\mathbf{1}\boldsymbol{P}$ clarifies the relation, as permuting rows and columns of the identity matrix must result in the identity matrix again. Clearly, the identity matrix corresponds to the identity permutation. This is why we use the same symbol "**1**" to denote both concepts. However, which concept is meant in particular is always clear from the context in which we use the symbol.

Example 12. *We continue Examples 10/11. The permutation matrix that corresponds to* $p = (1)(2\ 10\ 6\ 3\ 8\ 5)(4)(7\ 9)$ *is*

$$\boldsymbol{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.61}$$

The row and column sums are all 1*, as each element of* Ω *is mapped onto exactly one other element of* Ω*.*

2.4.2 Permutation Groups

The set of all possible permutations on Ω is called the symmetric group S^{Ω} (Wielandt, 1964, p. 2). Often, if the actual set Ω is not of importance and due to the fact that S^{Ω} is isomorphic to the permutation group that acts on the integers $1, 2, \ldots, n = |\Omega|$, the symmetric group is denoted S_n. Isomorphic means that there exists a bijective function $\phi : \Omega \to \{1, 2, \ldots, n\}$ that maps each element $\alpha \in \Omega$ to an integer between 1 and n. Therefore, ϕ allows every permutation that is defined on Ω to be represented as a permutation defined on $1, \ldots, n$. We use both notations interchangeably. This group naturally has some properties:

1. $\mathbf{1} \in S_n$; the identity permutation is a permutation of Ω.
2. $\forall p, q \in S_n : pq \in S_n$, because any catenation of permutations on Ω must again be a permutation on Ω.
3. $\forall p \in S_n : p^{-1} \in S_n$ and $pp^{-1} = p^{-1}p = \mathbf{1}$, i.e. for every permutation exists an inverse for which $\beta \mapsto \beta^{p^{-1}} = \alpha$ $(\alpha^p = \beta)$ holds.
4. $\forall p, q, r \in S_n : (pq)r = p(qr)$; catenation is associative.

We call permutations that exchange only two elements transpositions. A permutation p that is its own inverse ($p = p^{-1}$) is called involution and every transposition is an involution. We also have $p^{-1} \circ p^1 = \mathbf{1} = p^{-1+1} = p^0$.

Example 13. *We continue Examples 10–12. The inverse of* $p = (2\ 10\ 6\ 3\ 8\ 5)(7\ 9)$ *is*

$$p^{-1} = (2\ 5\ 8\ 3\ 6\ 10)(7\ 9). \tag{2.62}$$

It is formed by simply reversing each cycle. Which element is written first in each cycle is arbitrary, however, for convenience we normally write the smallest element first. It is easy to check that

$$p \circ p^{-1} = p^{-1} \circ p = (2\ 5\ 8\ 3\ 6\ 10)(7\ 9) \circ (2\ 10\ 6\ 3\ 8\ 5)(7\ 9) = () = \mathbf{1} \tag{2.63}$$

as, e.g., $2 \overset{p^{-1}}{\mapsto} 5 \overset{p}{\mapsto} 2$ *and* $2 \overset{p}{\mapsto} 10 \overset{p^{-1}}{\mapsto} 2$. *The inverse of p in the matrix representation* $\boldsymbol{P}$ *is simply* $\boldsymbol{P}^{\mathrm{T}}$. *Again,* $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{P} = \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}} = \mathbf{1}$ *is easily checked.*

The order of the permutation group S_n is the number of its contained permutations $|S_n|$.

Theorem 1 (Order of the symmetric group)**.** *The order of* S_n *is* $n!$.

Proof. Clearly, for $n = 1$ only the identity permutation exists and $|S_1| = 1! = 1$ holds. For $n = 2$, only the transposition that exchanges both elements $(\alpha\ \beta)$ is added. $|S_2| = 2! = 2$ holds. For the case $n = 3$ the identity exists, additionally a transposition for every of the three pairs of elements exists, and, lastly, the permutation $(\alpha\ \beta\ \gamma)$ and its inverse $(\alpha\ \gamma\ \beta)$ exists. Therefore, $|S_3| = 3! = 6$ holds.

Let the symmetric group be S_{n-1} for some n with $|S_{n-1}| = (n-1)!$. Adding a new element ν to Ω will imply $n-1$ new transpositions $(\alpha\ \nu), \forall \alpha \in \Omega \setminus \{\nu\}$. By using the group operator $\circ$, we need to compute every catenation $(\alpha\ \nu) \circ p$ for all $p \in S_{n-1}$. It is sufficient to only compute those catenations because:

1. If α is fixed by p, $(\alpha\ \nu) \circ p = p \circ (\alpha\ \nu)$,
2. and else $(\alpha\ \nu)$ "inserts" ν in the cycle after α: $p = (\dots \alpha\ \beta \dots)$ and $(\alpha\ \nu) \circ (\dots \alpha\ \beta \dots) = (\dots \alpha\ \nu\ \beta \dots)$ because p of course fixes ν. This is the same as $p \circ (\beta\ \nu)$, which inserts γ before β (i.e. we could also compute all $p \circ (\alpha\ \nu)$).
3. Any catenations $(\alpha\ \nu) \circ (\beta\ \nu) = (\alpha\ \beta)$ are already in S_{n-1}.

This gives us the new order as

$$\begin{aligned} |S_{n-1}| + (n-1) \cdot |S_{n-1}| &= |S_{n-1}| \cdot (1 + (n-1)) \\ &= (n-1)! \cdot n \\ &= n! \\ &= |S_n|. \end{aligned} \tag{2.64}$$

□

2.4.2.1 Subgroups and Generators

A subset $G \subseteq S^{\Omega}$ is a subgroup of S^{Ω} if G is a group (i.e. all group properties hold). It is denoted as

$$G \leq S^{\Omega}. \tag{2.65}$$

The proper case $G \subset S^\Omega$ is denoted as

$$G < S^\Omega \tag{2.66}$$

and G is called proper subgroup. The subgroup definition is not limited to subsets of S^Ω, i.e. H is a subgroup of G and a proper subgroup of S^Ω iff $H \leq G < S^\Omega$. However, H is of course also a subgroup of S^Ω and we can state that *any* group that acts on Ω is a subgroup of S^Ω. As for S_n, $|G|$ denotes the order of the group and means the number of permutations in G.

Up to this point, we implicitly assumed that a group G is represented by the explicit enumeration of all the permutations it contains. However, as the group order often grows rapidly, a much more compact representation is given by a set of *generators*. Any subset $S \subset S^\Omega$ can be used to generate a group G. This means $G = S \cup \underline{S}$ and $\underline{S}$ contains all permutations that can be created by catenating all possible combinations of permutations $p, q, \ldots \in S$. A group obviously generates itself and a set S that generates a group G is called *generating set*. The permutations $p \in S$ are called generators. If a set $S \subset S^\Omega$ generates a group G we denote this by $\langle S \rangle = G$. Note that S and G are both subsets of S^Ω, but generally only G (besides S^Ω itself) is a group. An exception is the case where $S = G$, which means S is already a group. A special case is $S = \{\mathbf{1}\}$ and we call it *trivial group* or *identity group*.

The set that generates a group is usually not unique. A set S^* with the smallest necessary number of permutations to generate a group G is called a minimal generating set. S^* is minimal if no generators can be removed from S^* but G can still be generated, formally

$$\forall p \in S^* : \langle S^* \rangle = G > G' = \langle S^* \setminus \{p\} \rangle. \tag{2.67}$$

Equation 2.67 does not imply that a minimal generating set S^* is unique. Therefore, if S^* is minimal there possibly exists another minimal generating set $\tilde{S}^*$ with $|\tilde{S}^*| \neq |S^*|$.

Example 14. *We continue Examples 10–13. We can use the permutation $p = (2\ 10\ 6\ 3\ 8\ 5)(7\ 9)$ as a generator for a permutation group $\langle \{p\} \rangle$. In general, the Schreier-Sims-Algorithm (Sims, 1970) (see also Butler (1991) or Seress (2003, p. 57 ff.)) can be used as the basis for a systematic group generation. However, the generation is very easy for this example, as only one generator exists. To satisfy the group properties, p has to be catenated with itself until all other permutations are computed (Table 2.1). Because of the associativity of the catenation, e.g., $p^7 = p^6 \circ p = \mathbf{1} \circ p = p$, or, more generic, permutations p^k and p^l are equal for $l \equiv k \pmod 6$. From Tables 2.1 and 2.2 we can see that the order is $|\langle \{p\} \rangle| = 6$.*

k	$p \circ p^{k-1}$	p^k	Permutation
$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$\vdots$	$()$	$\mathbf{1}$
1	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ ()$	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9)$	p
2	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ (2\ 10\ 6\ 3\ 8\ 5)(7\ 9)$	$(2\ 6\ 8)(3\ 5\ 10)$	q
3	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ (2\ 6\ 8)(3\ 5\ 10)$	$(2\ 3)(5\ 6)(7\ 9)(8\ 10)$	$r = r^{-1}$
4	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ (2\ 3)(5\ 6)(7\ 9)(8\ 10)$	$(2\ 8\ 6)(3\ 10\ 5)$	q^{-1}
5	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ (2\ 8\ 6)(3\ 10\ 5)$	$(2\ 5\ 8\ 3\ 6\ 10)(7\ 9)$	p^{-1}
6	$(2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \circ (2\ 5\ 8\ 3\ 6\ 10)(7\ 9)$	$()$	$\mathbf{1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2.1: All possible permutations created by catenating p with itself k times.

Table 2.2 shows the so called Cayley table (Cayley and Forsyth, 1889, pp. 123 ff.) of $\langle\{p\}\rangle = \{\mathbf{1}, p, q, r, q^{-1}, p^{-1}\}$ *where the results of all permutation catenations are presented.*

$\circ$	p	q	r	q^{-1}	p^{-1}	$\mathbf{1}$
p	q	r	q^{-1}	p^{-1}	$\mathbf{1}$	p
q	r	q^{-1}	p^{-1}	$\mathbf{1}$	p	q
r	q^{-1}	p^{-1}	$\mathbf{1}$	p	q	r
q^{-1}	p^{-1}	$\mathbf{1}$	p	q	r	q^{-1}
p^{-1}	$\mathbf{1}$	p	q	r	q^{-1}	p^{-1}
$\mathbf{1}$	p	q	r	q^{-1}	p^{-1}	$\mathbf{1}$

Table 2.2: The Cayley table of the group generated by p.

The *cosets* of a group H are defined as $gH := \{gh \mid h \in H\}$ (the left coset) and $Hg := \{hg \mid h \in H\}$ (the right coset), respectively. We use the cosets to define *normal subgroups*. A subgroup $H \leq G$ is called normal iff

$$\forall h \in H, \forall g \in G : ghg^{-1} \in H, \tag{2.68}$$

which is equivalent to

$$\forall g \in G : gH = Hg, \tag{2.69}$$

i.e. the left and right coset are equal. We denote normal subgroups as $H \trianglelefteq G$ ($\triangleleft$ for the proper case) in accordance to most authors (e.g. Wielandt, 1964; Dixon and Mortimer, 1996).

2.4.2.2 Fixed Blocks, Orbits, and the Orbit Stabilizer Theorem

Given subsets $\Delta \subseteq \Omega$ and $G \leq S^{\Omega}$, Wielandt (1964) defines

$$\Delta^G := \{\delta^g \mid \delta \in \Delta, g \in G\} \tag{2.70}$$

and calls Δ a fixed block iff $\Delta = \Delta^G$. For $|\Delta| = 1$ (i.e. $\Delta = \{\alpha\}$) and $G \leq S^\Omega$ we can write

$$\alpha^G := \{\alpha^g \mid g \in G\} \tag{2.71}$$

and call this set the *orbit* of α. $|\alpha^G|$ is the orbit *length*. Every element $\alpha \in \Omega$ must lie on exactly one orbit and the set of all orbits is clearly a partition of Ω. An orbit Δ_i is of course always fixed by G, i.e.

$$\Delta_i = \Delta_i^G. \tag{2.72}$$

Every group has at least two trivial fixed blocks, namely $\varnothing$ and Ω itself. If no other fixed blocks exist, the group is called transitive because

$$\forall \alpha, \beta, \gamma \in \Omega \; \exists g, h \in G : (\alpha^g = \beta) \wedge (\beta^h = \gamma) \iff \alpha^{gh} = \gamma. \tag{2.73}$$

This also means there is only one orbit $\alpha^G = \Omega$. Trivially, the symmetric group is always transitive.

The subgroup of permutations that fix α is defined as (Wielandt, 1964, p. 5)

$$G_\alpha := \{g \in G \mid \alpha^g = \alpha\}. \tag{2.74}$$

G_α is called the pointwise stabilizer and the definition can be extended to more than one point (Wielandt, 1964, p. 5):

$$G_\Delta := \bigcap_{\delta \in \Delta} G_\delta. \tag{2.75}$$

Equations 2.71 and 2.74 can be combined to a theorem that relates the length of an orbit and the order of the stabilizer subgroup.

Theorem 2 (Orbit stabilizer theorem)**.** *The relation*

$$|G| = |G_\alpha| \cdot |\alpha^G| \tag{2.76}$$

holds for $\alpha \in \Omega$ and $G \leq S^\Omega$.

Proof. See Wielandt (1964, Theorem 3.2, p. 5). □

As G_α is a group, Theorem 2 can be applied recursively (if $|G_\alpha| > 1$, otherwise G_α already fixes all elements, i.e. it is the trivial group). Therefore, $|G_\alpha| = |(G_\alpha)_\beta| \cdot |\beta^{G_\alpha}|$ and

$$\begin{aligned} |G| = |G_\alpha| \cdot |\alpha^G| = |(G_\alpha)_\beta| \cdot |\beta^{G_\alpha}| \cdot |\alpha^G| &= |G_{\alpha,\beta}| \cdot |\beta^{G_\alpha}| \cdot |\alpha^G| \\ &= |G_{\alpha,\beta,\gamma}| \cdot |\gamma^{G_{\alpha,\beta}}| \cdot |\beta^{G_\alpha}| \cdot |\alpha^G| \\ &= \ldots. \end{aligned} \tag{2.77}$$

The stabilizer subgroups $G_{\alpha,\beta}$, $G_{\alpha,\beta,\gamma}$, and so forth, are just the stabilizer subgroups G_Δ for $\Delta = \{\alpha, \beta\}$, $\Delta = \{\alpha, \beta, \gamma\}$, and so on.

The relation from Equation 2.77 can also be used to prove Theorem 1 ($|S_n| = n!$):

Proof. Let $G = S^{\Omega}$, then clearly $|\alpha^G| = n$ and $G_\alpha = S^{\Omega \setminus \{\alpha\}}$. This procedure can be repeated recursively; the orbit length of $S^{\Omega \setminus \Theta}$ is $|\Omega| - |\Theta| = n - k$ and therefore

$$|S^{\Omega}| = |\alpha^{S^{\Omega}}| \cdot |S^{\Omega \setminus \{\alpha\}}| \tag{2.78}$$

$$= |\alpha^{S^{\Omega}}| \cdot \left(|\beta^{S^{\Omega \setminus \{\alpha\}}}| \cdot |S^{\Omega \setminus \{\alpha,\beta\}}|\right) \tag{2.79}$$

$$= |\alpha^{S^{\Omega}}| \cdot \left(|\beta^{S^{\Omega \setminus \{\alpha\}}}| \cdot \left(\dots\left(|\gamma^{S^{\gamma}}| \cdot |S^{\{\zeta\}}|\right)\right)\right) \tag{2.80}$$

$$= n \cdot ((n-1) \cdot (\dots (1 \cdot 1))) \tag{2.81}$$

$$= \prod_{k=0}^{n-1} (n-k) \tag{2.82}$$

$$= n! \tag{2.83}$$

Note that α, β, γ certainly lie on the same orbit given S^{Ω} but not after, e.g., fixing α and creating $S^{\Omega \setminus \{\alpha\}}$. □

Example 15. *We continue Examples 10–14. Let $\langle p \rangle = \langle (2\ 10\ 6\ 3\ 8\ 5)(7\ 9) \rangle = G$. The orbit partition of G is $O = \{\{1\}, \{2,3,5,6,8,10\}, \{4\}, \{7,9\}\}$. Therefore, the orbit of 7 is $\{7,9\}$ and the subgroup of G that stabilizes 7 is $G_7 = \{\mathbf{1}, q, q^{-1}\} = \{(), (2\ 6\ 8)(3\ 5\ 10), (2\ 8\ 6)(3\ 10\ 5)\}$. See Table 2.1 to reassure that the other permutations do not fix 7. From the orbit stabilizer theorem follows*

$$|G| = |7^G| \cdot |G_7| = 2 \cdot 3 = 6, \tag{2.84}$$

which is true.

Next, we prove a theorem that will be used in Section 6.4.3.4.

Theorem 3 (Equiprobability of orbit elements)**.** *Given some fixed α with $|\alpha^G| > 1$. Then $\mathbb{P}(\alpha^g = \beta) = \frac{1}{|\alpha^G|}$ holds for $\beta \in \alpha^G$ and $g \in G$ randomly chosen.*

Proof. The probability $\mathbb{P}(\alpha^g = \alpha) = \frac{|G_\alpha|}{|G|} = \frac{1}{|\alpha^G|}$ is a direct consequence of the orbit stabilizer theorem. Let $G_\alpha = \{f_1, \dots, f_k\}$ and $G_{\alpha \to \beta} := \{g \in G \mid \alpha^g = \beta\} = \{g_1, \dots, g_l\}$. As each $f_i \in G_\alpha$ fixes α and each $g_j \in G_{\alpha \to \beta}$ maps α onto β, $\left\{f_i \circ g_j \mid f_i \in G_\alpha, g_j \in G_{\alpha \to \beta}\right\} = G_{\alpha \to \beta}$ must hold.

1. Suppose $\mathbb{P}(\alpha^g = \beta) = \frac{|G_{\alpha \to \beta}|}{|G|} < \frac{|G_\alpha|}{|G|} = \frac{1}{|\alpha^G|}$, i.e. $l < k$. For some fixed $g_{j'} \in G_{\alpha \to \beta}$, the coset $G_\alpha g_{j'}$ must contain k elements and each is a permutation that maps α onto β, as no $i \neq i'$ with $f_i \circ g_{j'} = f_{i'} \circ g_{j'}$ can exist unless $f_i = f_{i'}$. This is a contradiction to $l < k$.

2. The other way round, suppose $\mathbb{P}(\alpha^g = \beta) > \frac{1}{|\alpha^G|}$, i.e. $l > k$. The only possibility that this relation can exist is that there is a $\beta \neq \gamma \in \alpha^G$ for which $\mathbb{P}(\alpha^g = \gamma) < \frac{1}{|\alpha^G|}$ holds. But that is impossible due to case 1.

We conclude that there cannot exist an element on an orbit that is more or less likely to map onto, i.e. $\mathbb{P}(\alpha^g = \beta) = \frac{1}{|\alpha^G|}$ for all $\beta \in \alpha^G$. □

2.4.2.3 Blocks

Another definition (also Wielandt, 1964, §6 on pp. 11 ff.) are blocks. Whereas in a *fixed block* all elements are only fixed within the block. A subset $\Psi \subseteq \Omega$ is a *block* of a permutation group $G \leq S^{\Omega}$ iff

$$\forall g \in G : (\Psi^g = \Psi) \vee (\Psi^g \cap \Psi = \emptyset) \tag{2.85}$$

holds. There are several trivial blocks: Again, the empty set $\emptyset$ and the set Ω itself are blocks, but, additionally, every single element $\{\alpha\} \subset \Omega$ is a block. The intersection of two blocks is again a block (Wielandt, 1964, p. 12) and every Ψ^g with $\Psi \cap \Psi^g = \emptyset$, $g \in G$ is also a block. Every fixed block is of course a block (thus orbits are blocks) and unions of fixed blocks are again fixed blocks (therefore also blocks). A set of disjoint blocks partitions Ω. In contrast to the partition of orbits, a partition of blocks can be finer concerning the definitions in Section 2.3, as an orbit can possibly be further divided into smaller non-trivial blocks.

Example 16. *We continue Examples 10–15. A partition of possible non-trivial blocks of* $\langle p \rangle = \langle (1)(2\ 10\ 6\ 3\ 8\ 5)(4)(7\ 9) \rangle$ *is* $B = \{\{2,6,7,8\},\{3,5,9,10\},\{1,4\}\}$ *and, e.g.,* $\{2,6,7,8\}^q = \{2^q,6^q,7^q,8^q\} = \{8,2,7,6\}$ *(case* $\Psi = \Psi^g$*) or* $\{2,6,7,8\}^p = \{5,10,9,3\}$ *(case* $\Psi \cap \Psi^g = \emptyset$*). The set* $\{1,4\}$ *is the union of two trivial orbits, which form a (fixed) block.*

2.4.3 Products of Permutation Groups

Several definitions of products of permutation groups exist and we present three of them, namely the internal direct product, the semidirect product, and the wreath product. The products can be seen from a constructive point of view, as they allow to compose new permutation groups from given permutation groups, or from an analytic point of view, which allows to describe how a given permutation group can be decomposed.

The *internal direct product* of two non-trivial normal subgroups $H_1 \lhd G$ and $H_2 \lhd G$ $(H_1 \neq H_2)$ is defined as

$$H_1 \times H_2 := \{ab \mid a \in H_1, b \in H_2\} \tag{2.86}$$

where $H_1 \cap H_2 = \{\mathbf{1}\}$. Because of the associativity of permutations, this can be generalized to k normal subgroups H_i, thus

$$G = H_1 \times \ldots \times H_k = \prod_{i=1}^{k} H_i. \tag{2.87}$$

Given a set of generators S, this set can be divided into k disjoint subsets S_i with $\langle S_i \rangle = H_i$. A group that cannot be calculated as inner product under the conditions above is said to be *indecomposable*.

The theorem of Krull-Remak-Schmidt states that the composition of a group G from indecomposable subgroups H_i (if possible) is unique up to isomorphism (proof for finite groups by Remak, 1911, p. 298). An implication of this is that if a decomposition of a group into smaller subgroups is possible, the group acts on different independent "areas"/subsets of the set Ω. Furthermore, $|G| = \prod_{i=1}^{k} |H_i|$, as G combines all possible permutations of each subgroup

H_i. This approach of decomposing a group G into independent subgroups is also described by MacArthur et al. (2008, p. 3526 f.) and they call it the geometric decomposition. They also prove that this decomposition does not depend on the actual choice of generators if they obey certain conditions:

1. No generator is a composition of support disjoint permutations: $\nexists s \in S,\ g, h \in \langle S \rangle : s = gh$ and $\operatorname{supp}(g) \cap \operatorname{supp}(h) = \varnothing$
2. For each subgroup $H_1 \times H_2 = H \leq G$ a subset $S' \subseteq S$ exists for which $S' = S_1 \cup S_2$ holds, $\langle S_i \rangle = H_i$, and $S_1 \cap S_2 = \varnothing$

They call such a generating set *essential* and throughout this thesis all generating sets are assumed to be essential. The authors furthermore state that algorithms like `nauty` (see Section 3.4), which compute a set of generators for the graph automorphism group (see Section 3.2), return essential sets.

Example 17. *We continue Examples 10–16. Taking the permutation*

$$s = (1\ 4) \tag{2.88}$$

in addition to $p = (1)(2\ 10\ 6\ 3\ 8\ 5)(4)(7\ 9)$ *yields a new permutation group* $J = \langle \{p, s\} \rangle$*. It can easily be seen that* $J = \langle \{p\} \rangle \times \langle \{s\} \rangle = G \times H$ *because* $G \lhd J$ *and* $H \lhd J$*. From* $|G| = 6$ *and* $|H| = 2$ *follows* $|J| = |G| \cdot |H| = 12$.

We follow Dixon and Mortimer (1996, pp. 44 ff.) by defining the two other product operations of groups. For two groups G and H, $H \rtimes G$ denotes the so called *semidirect product*. H is a group that acts on some set Ω and G is a group acting on H itself. This means each permutation $g \in G$ is defined to map a permutation $h \in H$ onto another permutation $h' \in H$, i.e. $g : H \to H$ and $h \mapsto h^g$ is the image of g applied on h. The result is a group that contains pairs of permutations and it acts on tuples $(\alpha, h) \in \Omega \times H$. The group is defined as

$$J = H \rtimes G := \{(h,\ g) \mid g \in G, h \in H\} \tag{2.89}$$

together with the catenation defined as

$$(h_1, g_1)(h_2, g_2) := (h_1 h_2^{g_1^{-1}},\ g_1 g_2). \tag{2.90}$$

The catenation of permutations in G and H is defined as in Section 2.4.1. The catenation in Equation 2.90 is well-defined as $g_1 g_2 = g_1 \circ g_2 \in G$ (by definition, G is a group) and $h_2^{g^{-1}} =: h_3 \in H$, therefore, $h_1 h_2^{g^{-1}} = h_1 h_3 = h_1 \circ h_3 \in H$ (by definition, H is a group, too). So the semidirect product of an arbitrary group and a group acting on this group (an automorphism group of the group) is again a group with the group properties defined above.

The second operation is called *wreath product* and it can be used to describe groups that are compositions of “copies” of smaller groups. Let H be a group (the “small” group to be copied) and define $\mathrm{Fun}(\Lambda, H)$ as the set of all functions $f : \Lambda \to H$, where $\Lambda = \{\lambda_1, \ldots, \lambda_k\}$ is a set of k symbols to identify each copy of H. $\mathrm{Fun}(\Lambda, H)$ is itself a group, as

$$(fg)(\lambda) := f(\lambda)g(\lambda) \quad \forall f, g \in \mathrm{Fun}(\Lambda, H),\ \lambda \in \Lambda \tag{2.91}$$

is defined because $f(\lambda), g(\lambda) \in H$ and H is a group. Furthermore, let G be a group that acts on Λ and, therefore, can be interpreted as the "copying" group, which takes the actions of H to each of its copies H_{λ_i}. A copy H_{λ_i} acts on a copy $\Omega_{\lambda_i} = \{\alpha_{\lambda_i}, \beta_{\lambda_i}, \ldots\}$ of the element set $\Omega = \{\alpha, \beta, \ldots\}$. The wreath product is then defined as

$$H \wr_\Lambda G := \mathrm{Fun}(\Lambda, H) \rtimes G \tag{2.92}$$

together with the operation

$$f^g(\lambda) := f(\lambda^{g^{-1}}) \quad g \in G, \lambda \in \Lambda, f \in \mathrm{Fun}(\Lambda, H), \tag{2.93}$$

which represents the action of $g \in G$ on $\mathrm{Fun}(\Lambda, H)$. The wreath product $H \wr_\Lambda G$ acts on the union $\Omega_\Lambda := \bigcup_{\lambda \in \Lambda} \Omega_\lambda$ of the k labeled copies of Ω. An element $(f, g) \in H \wr_\Lambda G$ is applied on an element $\alpha_\lambda \in \Omega_\Lambda$ as

$$\alpha_\lambda^{(f,g)} := \alpha_{\lambda^{g^{-1}}}^{f^g(\lambda)}. \tag{2.94}$$

We can set $\Lambda = G$ if G acts *regularly* on itself (Dixon and Mortimer, 1996, p. 47), i.e. G must be *transitive* ($\forall g, g' \in G\ \exists g^* : gg^* = g'$) and *fixed point free* ($\forall g, g', g^* \in G : gg' = gg^* \iff g' = g^*$). If so, the notation is shortened to $H \wr G$.

Example 18. *Given the group $H = \{(), (1\ 2)\}$, a set $\Lambda = \{a, b\}$ to distinguish two copies of H, and a group $G = \{(), (a\ b)\}$ that acts on Λ. H clearly acts on $\Omega = \{1, 2\}$. The wreath product $H \wr_\Lambda G$ is then given by the semidirect product*

$$\mathrm{Fun}(\Lambda, H) \rtimes G = \{(f_1, ()), (f_1, (a\ b)), (f_2, ()), \ldots, (f_4, (a\ b))\} \tag{2.95}$$

with

$$\mathrm{Fun}(\Lambda, H) = \left\{ f_1 = \begin{cases} a \mapsto () \\ b \mapsto () \end{cases}, f_2 = \begin{cases} a \mapsto (1\ 2) \\ b \mapsto () \end{cases}, f_3 = \begin{cases} a \mapsto () \\ b \mapsto (1\ 2) \end{cases}, f_4 = \begin{cases} a \mapsto (1\ 2) \\ b \mapsto (1\ 2) \end{cases} \right\}. \tag{2.96}$$

The $f_i \in \mathrm{Fun}(\Lambda, H)$ represent all possible functions from Λ to H and $\mathrm{Fun}(\Lambda, H)$ is a group itself. Taking, for instance, $f_2, f_3 \in \mathrm{Fun}(\Lambda, H)$ we can form the catenation $f_2 \circ f_3$ as

$$f_2 \circ f_3 = \begin{cases} a \mapsto (f_2 \circ f_3)(a) \\ b \mapsto (f_2 \circ f_3)(b) \end{cases} = \begin{cases} a \mapsto (1\ 2) \circ () \\ b \mapsto () \circ (1\ 2) \end{cases} = \begin{cases} a \mapsto (1\ 2) \\ b \mapsto (1\ 2) \end{cases} = f_4. \tag{2.97}$$

Furthermore, each f_i *represents a permutation on two copies* $\Omega_a = \{1_a, 2_a\}$ *and* $\Omega_b = \{1_b, 2_b\}$ *of the set* Ω *on which the copies* H_a *and* H_b *of* H *act. The wreath product* $H \wr_\Lambda G = \mathrm{Fun}(\Lambda, H) \rtimes G$ *acts on the set* $\Omega_{\{a,b\}} = \{1_a, 2_a, 1_b, 2_b\}$. *For example,* $(f_2, (a\ b)) \in \mathrm{Fun}(\Lambda, H) \rtimes G$ *acts on* $\Omega_{a,b}$ *as follows:*

$$1_a^{(f_2,(a\ b))} = 1_{a^{(a\ b)^{-1}}}^{f_2(a^{(a\ b)^{-1}})} = 1_{a^{(a\ b)}}^{f_2(a^{(a\ b)})} = 1_b^{f_2(b)} = 1_b^{()} = 1_b \tag{2.98}$$

$$1_b^{(f_2,(a\ b))} = \ldots = 1_a^{f_2(a)} = 1_a^{(1\ 2)} = 2_a \tag{2.99}$$

$$2_a^{(f_2,(a\ b))} = \ldots = 2_b^{f_2(b)} = 2_b^{()} = 2_b \tag{2.100}$$

$$2_b^{(f_2,(a\ b))} = \ldots = 2_a^{f_2(a)} = 2_a^{(1\ 2)} = 1_a \tag{2.101}$$

This means $(f_2, (a\ b))$ *corresponds to the permutation* $(1_a\ 1_b\ 2_a\ 2_b)$*, which is the combination of the transposition* $f_2(a) = (1\ 2)$ *that acts on* Ω_a *and the transposition* $(a\ b)$ *that maps* Ω_a *onto* Ω_b *and vice versa. The group* $H \wr_\Lambda G$ *is isomorphic to* $S_2 \wr S_2$ *as well as to, e.g.,*

$$F = \{(), (1\ 2), (3\ 4), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (1\ 4)(2\ 3), (1\ 4\ 2\ 3), (1\ 3\ 2\ 4)\} \tag{2.102}$$

via the isomorphism (in explicit permutation notation)

$$\phi = \begin{pmatrix} 1_a & 2_a & 1_b & 2_b \\ 1 & 2 & 3 & 4 \end{pmatrix}. \tag{2.103}$$

2.5 Graph Clustering

Graph clustering is an algorithmic approach to create a partition P of the node set V of a graph. The way a clustering partition is formed often follows the idea of grouping nodes that are "more similar" to each other into the same partition cell. Simultaneously, nodes in different cells should be dissimilar. In this context, the cells are called clusters, and this informal definition implies that the number of clusters is not an external parameter but inherently contained in the graph structure. Other methods are not the focus of this work.

Two extensive overview articles are given by Schaeffer (2007) and Fortunato (2010). Both distinguish between local and global definitions of functions that quantify desirable properties for graph clustering and they discuss how they can be of use for determining clusters of graphs. Fortunato (2010, section 3.2.1 on p. 83) states that "[n]o definition is universally accepted". Another review is by Harenberg et al. (2014), who compares several algorithms in terms of partition quality and runtime.

Relatively independent of the definition(s) of what clusters are, it is possible to classify clustering algorithms (or generic approaches) by their control strategies. One classification scheme is to distinguish between hierarchical and non-hierarchical methods. A hierarchical method produces a series of solutions by subsequently merging (agglomerative approach; bottom-up) or dividing (divisive approach; top-down) clusters of a partition. Each series of solutions is a path through the poset of partitions as defined in Section 2.3. Normally, only one solution out of the series is chosen as the result of an algorithm. In contrast, non-hierarchical methods produce exactly one solution (but also out of the set of partitions $\boldsymbol{P}(V)$).

One of the most important of the clustering criterion definitions for the graph clustering problem, which leads to numerous new algorithms (Sections 2.5.2–2.5.4) and on which active research is still focused on, is Newman's modularity (Newman and Girvan, 2004).

2.5.1 Modularity

Newman and Girvan (2004) introduced modularity as a global criterion for graph clustering partition quality. The formula can be written as

$$Q(P,G) = \sum_{i=1}^{|P|} (e_{ii} - a_i^2), \tag{2.104}$$

which is the sum over all clusters of nodes. The term over which is summed consists of

$$e_{ij} = \frac{\sum_{v_x \in C_i, v_y \in C_j} m_{xy}}{2|E|}, \tag{2.105}$$

$$a_i = \sum_{j=1}^{|P|} e_{ij}, \tag{2.106}$$

where, in this context, $\boldsymbol{M} = (m_{xy})$ denotes the adjacency matrix of G (instead if $\boldsymbol{A}$) to avoid possible confusion with the a_i. The e_{ij} also form a symmetric matrix $\boldsymbol{E}$ and, therefore, summing over each row we get the column vector $\boldsymbol{a} = (a_i)$. As each edge is counted twice in total (once for each of the two adjacent nodes), the constant 2 is found in the denominator of the definition of e_{ij}. For e_{ii}, both adjacent nodes are in the same cluster and the constant is canceled out. $\boldsymbol{E}$ can be interpreted as the adjacency matrix of a directed weighted graph with loops, which is a coarsening of G, and $\boldsymbol{a}$ is the vector of node outdegrees of this graph. We want to call this graph a *partition induced graph* $G(P)$ of G given P. As the weights of the directed edges from i to j and vice versa ($i \neq j$) are equal ($e_{ij} = e_{ji}$), this graph is equivalent to an undirected weighted graph with loops that has edge weights $e_{ij} + e_{ji} = 2e_{ij}$ for $i \neq j$. This view on the computation of modularity shows that the applicability of it is not limited to unweighted and loop-free graphs. As $\boldsymbol{E}$ and $\boldsymbol{a}$ capture (aggregated) degree information, modularity is also a graph invariant, as it will be defined in Section 3.3.

The sum from Equation 2.104 can, of course, be written as

$$Q(P,G) = \sum_{i=1}^{|P|} e_{ii} - \sum_{i=1}^{|P|} a_i^2 \tag{2.107}$$

and this brings us an alternative definition of modularity

$$Q(P,G) = \operatorname{trace}(\boldsymbol{E}) - \langle \boldsymbol{a}, \boldsymbol{a} \rangle = \operatorname{trace}(\boldsymbol{E}) - \|\boldsymbol{a}\|_2^2. \tag{2.108}$$

The trace is just the sum of diagonal matrix entries, $\langle \cdot, \cdot \rangle$ is the scalar product of two vectors and $\| \cdot \|_2$ is the Euclidean norm.

Equivalent definitions of e_{ii} and a_i are given by

$$e_{ii} = \frac{m_i}{m}, \tag{2.109}$$

with m_i the number of edges of the subgraph induced by C_i. Furthermore,

$$a_i = e_{ii} + \frac{l_i}{2m} \tag{2.110}$$

with l_i the number of edges that connect C_i to the rest of the graph (the inter-cluster-edges).

Modularity expresses the aggregated differences of the actual number of edges within each cluster and the expected number of edges within each cluster. The interpretation of Equation 2.109 is intuitive, the one of Equation 2.110 needs some explanation: a_i is the fraction of the number of incident edges with cluster C_i. Therefore, it is the relative probability of a random edge to be incident with a cluster that has the same properties—expressed solely by degree information—as C_i. Consequently, a_i^2 is the probability to randomly pick an edge that is incident with two nodes that are expected to have the same properties and thus are in the same cluster. Hence, a large difference $e_{ii} - a_i^2$ indicates a large deviation of the observed from the expected cluster density, which indicates a non random relationship of the nodes within this cluster.

2.5.1.1 Limits

In the following we give several limits of modularity.

Theorem 4 (Modularity of a singleton partition)**.** *For the singleton partition $P_\perp$, $Q(P_\perp, G) < 0$.*

Proof. For each cluster, $e_{ii} = 0$ holds, as no intra-edges exist, but each $a_i > 0$, because all incident edges are counted. Thus

$$Q(P_\perp, G) = \sum_{i=1}^{|P_\perp|} e_{ii} - \sum_{i=1}^{|P_\perp|} a_i^2 = -\sum_{i=1}^{n} a_i^2 < 0. \tag{2.111}$$

□

Theorem 5 (Modularity of a trivial partition)**.** *For the trivial partition $P_\top$, $Q(P_\top, G) = 0$.*

Proof. Only one large cluster exists, which contains all nodes and edges. Thus

$$Q(P_\top, G) = \sum_{i=1}^{|P_\top|} (e_{ii} - a_i^2) = \sum_{i=1}^{1} (e_{ii} - a_i^2) = (e_{11} - a_1^2) = 1 - 1^2 = 0. \tag{2.112}$$

□

Theorem 6 (Minimum Q (Brandes et al., 2008))**.** *The minimum possible modularity value for some partition is* $-\frac{1}{2}$.

Proof. A partition that minimizes Q has to minimize e_{ii} and maximize a_i for all clusters C_i. A cluster that minimizes Q needs to have no intra-edges and as many inter-edges as possible. For the singleton partition $P_\perp$, $e_{ii} = 0$ and $a_i = \frac{\deg(i)}{2m}$ (for all i). The more non-adjacent nodes that can be joined into a larger cluster by leaving $e_{ii} = 0$ and increasing a_i, the better. The best attainable result consists of two clusters with $e_{11} = e_{22} = 0$ and $a_1 = a_2 = \frac{m}{2m}$. The result is then $Q(P, G) = -2 * \left(\frac{m}{2m}\right)^2 = -\frac{1}{2}$, and it can only be achieved if G is bipartite. □

We now rewrite the parts of the modularity formula (Equation 2.104) slightly:

$$e_{ii} = \lambda_i \tag{2.113}$$

and

$$a_i = \lambda_i + \frac{1}{2}\mu_i \tag{2.114}$$

where λ_i denotes the fraction of intra-cluster edges in C_i, μ_i denotes the fraction of inter-cluster edges of C_i to other clusters.

$$\sum_i \left(\lambda_i + \frac{1}{2}\mu_i\right) = 1 \tag{2.115}$$

holds as well as $\lambda_i \in [0, 1]$ and $\mu_i \geq 0$.

Theorem 7 (Supremum of Q)**.** *The supremum of modularity is* 1.

Proof. To maximize Q, now e_{ii} must be maximized and a_i minimized for each cluster individually. As maximizing λ_i for some fixed i would result in the trivial partition (with $Q = 0$ for any graph), maximization is done "in parallel", and w.l.o.g. $\lambda_i := \lambda \ \forall i$ as well as $\mu_i := \mu \ \forall i$. Minimizing a_i means minimizing μ_i. In the limit we get

$$\lim_{\lambda \to \frac{1}{k}, \mu \to 0} k\left[\lambda - (\lambda + \frac{1}{2}\mu)^2\right] = k\left[\frac{1}{k} - \left(\frac{1}{k}\right)^2\right] = 1 - \frac{1}{k} \tag{2.116}$$

with k the number of clusters and $\lambda \to \frac{1}{k}$ because the fraction of intra-edges is equally distributed. Finally,

$$\lim_{k \to \infty} 1 - \frac{1}{k} = 1 \tag{2.117}$$

gives us the claimed result. Unless G is disconnected, no μ_i can actually be zero, and unless G is infinite, k cannot become infinitely large. Therefore, $Q^\top = 1$ is the supremum of $Q(P, G)$. □

Next, we want to look at the per-cluster contributions. By this we mean the term

$$\lambda_i - \left(\lambda_i + \frac{1}{2}\mu_i\right)^2, \tag{2.118}$$

which normally should be positive, when modularity is maximized. By setting Equation 2.118 to zero and solving it for μ_i we get (omitting the indices and for $\lambda_i \geq 0$)

$$\mu(\lambda) = 2\sqrt{\lambda} - 2\lambda, \quad \lambda \geq 0 \tag{2.119}$$

as the function of μ that depends on λ. Equation 2.118 is strictly positive for all $\mu < 2\lambda + 2\sqrt{\lambda}$ if $\lambda \in (0,1)$ and has a maximum at $\lambda = \frac{1}{4}$.

Figure 2.11 shows the plot of Equation 2.119 plus the additional constraint from Equation 2.115. For all points (λ', μ) within the red area $\lambda' - \left(\lambda' + \frac{1}{2}\mu\right)^2 > 0$ holds. The blue area

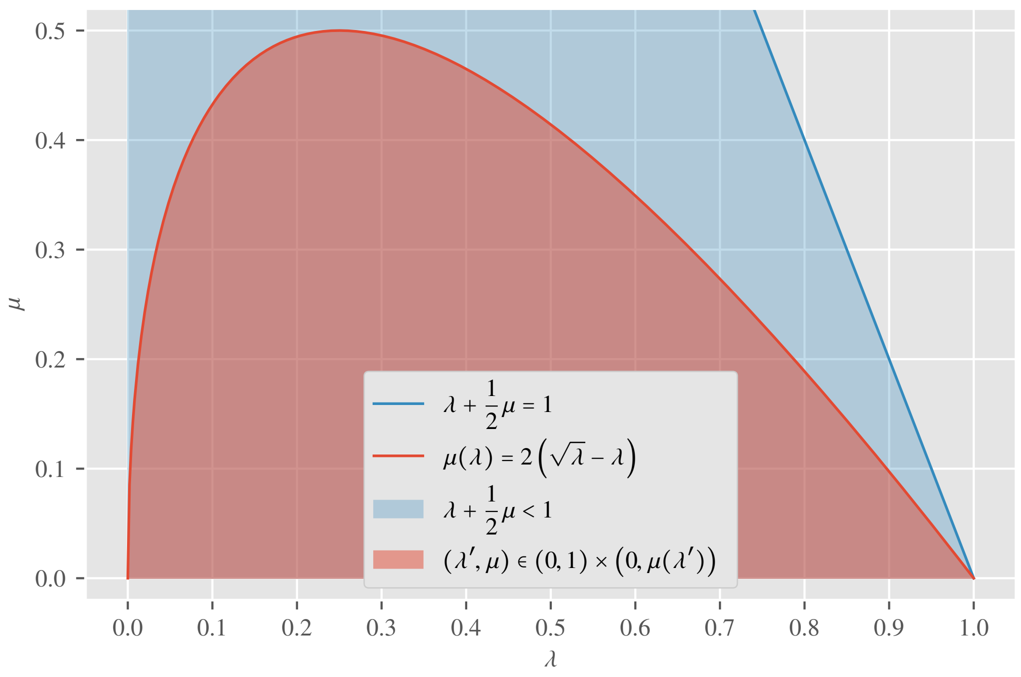


Figure 2.11: Isolated view on a single cluster and its contribution to modularity in the space of intra-cluster-edges and inter-cluster edges. The plot is independent of the actual number of edges but indirectly depends on it, as not every combination is possible. The blue area corresponds to all feasible combinations of λ and μ, the red area corresponds only two those combinations for which Equation 2.118 is strictly positive, and, therefore, the contribution of this cluster to modularity is positive.

(including the border and the two axes) contains all feasible points (λ, μ) if there exists more than only one (trivial) cluster. It is important to keep in mind that each combination strongly depends on all the other clusters of the partition. From Figure 2.11 we can also directly see that the only feasible point for a trivial partition is $(1,0)$ (with $\mu(1) = 0$) and the modularity contribution is zero for every combination $(\lambda, \mu(\lambda))$.

Figure 2.12 extends the examination with a third dimension to show the actual modularity contribution of one cluster, depending on the ratio of λ and μ. There is a maximum at $(\frac{1}{2}, 0)$ that is not feasible because of the constraint from Equation 2.115.

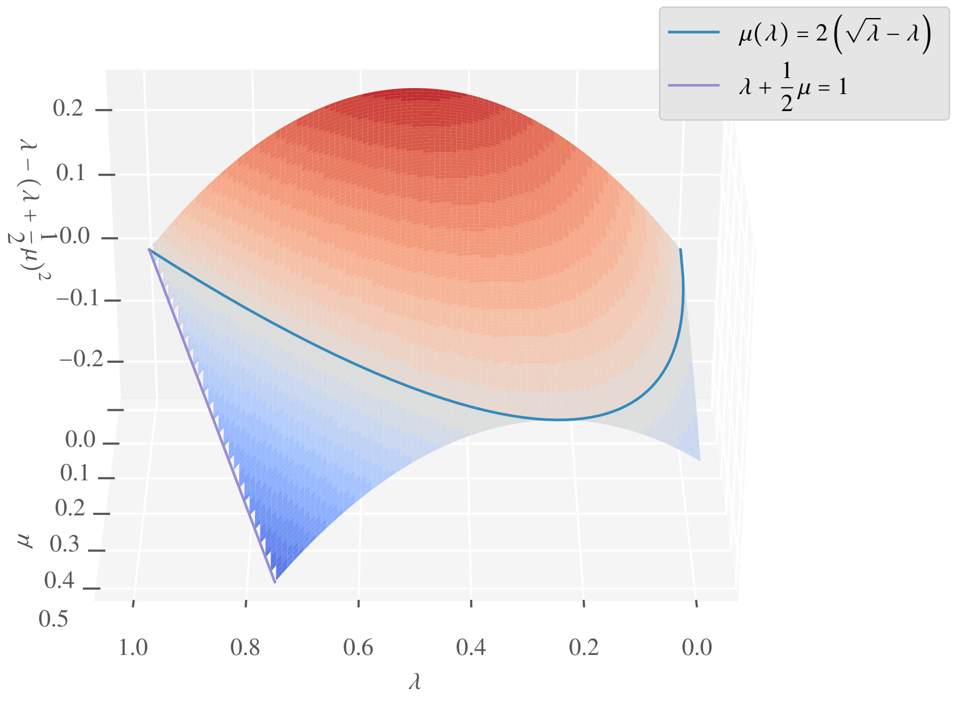


Figure 2.12: The modularity contribution of one cluster depending on the fraction of intra-cluster edges λ and inter-cluster edges μ. Red parts of the plot indicate a positive contribution, blue parts a negative contribution. The color intensity expresses the strength of the contribution. Figure 2.11 shows the cutting plane for $\lambda' - \left(\lambda' + \frac{1}{2}\mu\right)^2 = 0$.

The modularity for a given graph and a partition of it can be considered as the set of points (λ_i, μ_i). Each point is evaluated in the space shown in Figure 2.12 and eventually summed up. The interdependence between all points, given by the constraint in Equation 2.115 and the non-negativity of λ_i/μ_i, makes modularity a global clustering criterion.

Theorem 8 (Modularity of a complete graph K_n)**.** *$Q(P^*, K_n) = 0$ for the modularity optimal partition P^*.*

Proof. Suppose there exists a partition P' of K_n for that $Q(P', K_n) > 0$ holds. From Theorem 5 we know $P' < P_\top$ (P' must be strictly finer than the trivial partition). For each $C_i \in P'$

$$e_{ii} = \frac{\binom{n_i}{2}}{\binom{n}{2}} = \frac{n_i(n_i - 1)}{n(n-1)} \tag{2.120}$$

and

$$a_i = e_{ii} + \frac{n_i(n - n_i)}{2\binom{n}{2}} = \frac{n_i(n_i - 1)}{n(n-1)} + \frac{n_i(n - n_i)}{n(n-1)} = \frac{n_i}{n} \tag{2.121}$$

holds with $n_i := |C_i|$. The modularity contribution $e_{ii} - a_i^2$ of one cluster is

$$\begin{aligned}
\frac{n_i(n_i-1)}{n(n-1)} - \left(\frac{n_i}{n}\right)^2 &= \frac{n(n_i(n_i-1))}{n^2(n-1)} - \frac{(n-1)n_i^2}{n^2(n-1)} \\
&= \frac{nn_i^2 - nn_i - nn_i^2 + n_i^2}{n^2(n-1)} \\
&= \frac{n_i \overbrace{(n_i - n)}^{<0}}{n^2(n-1)} < 0.
\end{aligned} \tag{2.122}$$

The only n_i^* that maximizes the term above is $n_i^* = n$ leading to a value of zero, and, of course, $P^* = P_\top$, which is a contradiction. □

This result is desirable, as it captures the non-modularity of complete graphs.

2.5.1.2 Critique

We only shortly review the most important problem modularity has. Fortunato and Barthélemy (2007) and Lancichinetti and Fortunato (2011) show that the modularity definition comprises an inherent resolution limit. This is a direct consequence of the globality discussed in the previous section. The impact on clustering algorithms that try to maximize modularity is that smaller modular parts in the graph are merged to larger clusters, solely because the modularity increases. On the opposite, large modular parts tend to be split up. Figure 2.13 visualizes this fact quite well. Possible solutions to overcome the problem are given by Traag et al. (2011).

Van Laarhoven and Marchiori (2013) experimentally generalize this issue also to other objective functions than modularity and state that “the resolution bias of the objective function matters most” (van Laarhoven and Marchiori, 2013, part of the title). Furthermore, Kehagias and Pitsoulis (2013) present additional examples by defining families of graphs that consist of natural clusters by construction, which modularity fails to identify.

Another problem are possibly high modularity values in random graphs that, by definition, should not contain a structured modular organization. This issue, which is connected to the resolution limit, is addressed by Reichardt and Bornholdt (2006).

2.5.2 The Randomized Greedy Algorithm and its Extensions

Ovelgönne et al. (2010) present an $\mathcal{O}(m \ln n)$ time randomized greedy algorithm (RG), which is based on the first algorithm that made use of modularity (Newman, 2004, having time complexity $\mathcal{O}((m+n)n)$). It is a heuristic hierarchical agglomerative modularity optimizer, which uses the incremental increase of modularity $\Delta Q(C_i, C_j) = 2(e_{ij} - a_i a_j)$ if C_i and C_j are merged (Newman, 2004, p. 066133-2). The optimization procedure is straight forward:

1. Start with the singleton partition $P_\perp$ of the graph.

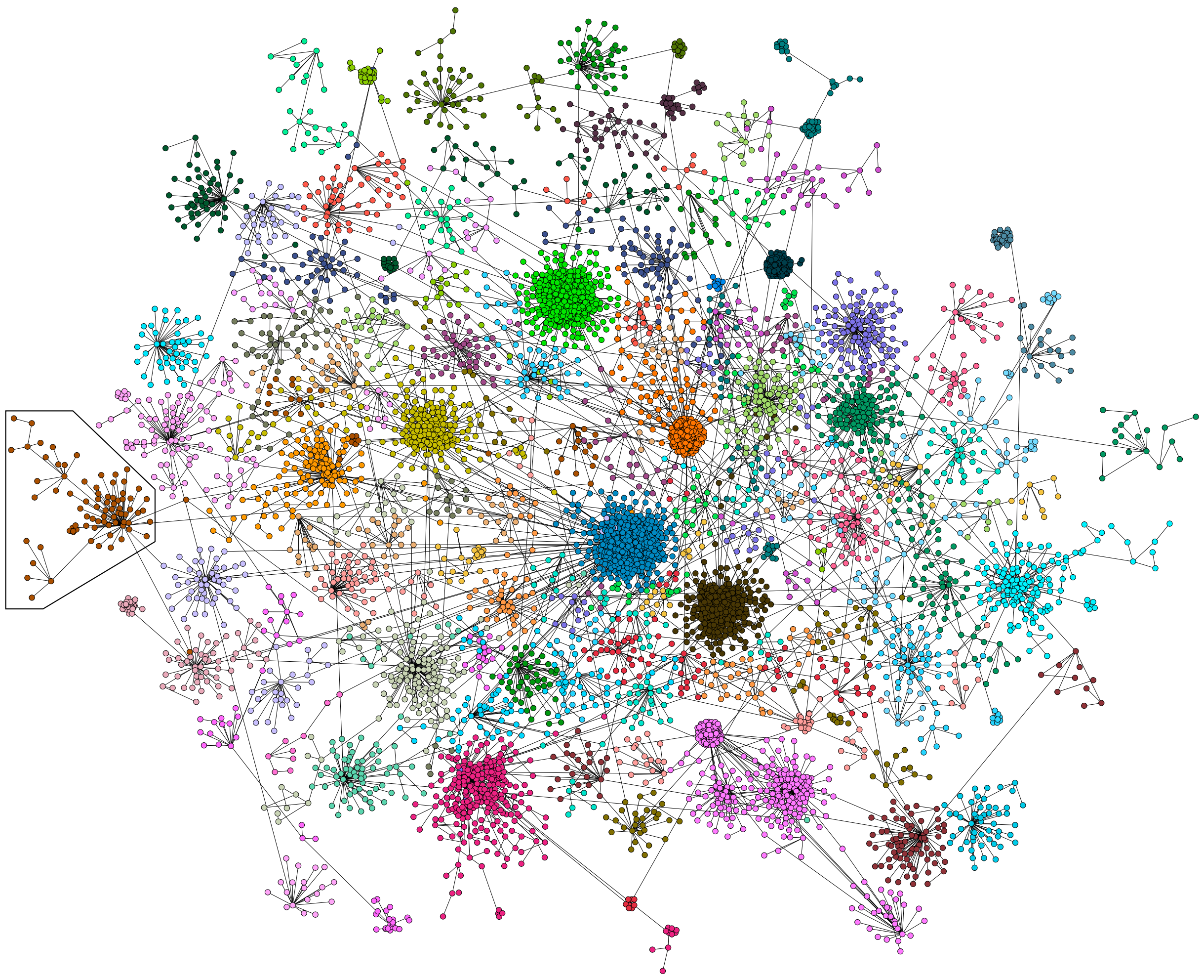


Figure 2.13: The network `rt_obama` from `networkrepository.com`, which is part of the symmetry analysis in Chapter 4. The colors represent the 52 clusters ($Q = 0.932$) that were found by the so called *Louvain algorithm* of Blondel et al. (2008). It is built into the GEPHI tool for graph analysis (Bastian et al., 2009), which was used to draw the graph utilizing the layout algorithm *OpenOrd* (Martin et al., 2011). The network is a good example that shows the resolution limit of modularity, as there can be found many small subgraphs, which obviously form "good" communities, but are all part of a larger cluster: For example, the brown cluster highlighted on the very left side of the drawing seems to consist of three or four natural communities.

2. Randomly pick $k = const$ (very small, often $k = 2$) clusters and compute for each of those clusters and all of its neighbors the gradient ΔQ.

3. The pair of clusters with the highest ΔQ value is chosen for the next join, ties are broken randomly.

4. Repeat steps 2 and 3 $n - 1$ times to obtain a complete path through the partition poset from $P_\perp$ to $P_\top$ (see Section 2.3).

5. Select the partition on the path with the highest value of Q as the final result.

The groundbreaking insight, which lead to such a speedup and increase in quality, is the existence of many equivalent (in terms of ΔQ) joins per iteration. This makes it superfluous to compute the modularity increase for every pair of connected clusters, as it is done in Newman's "plain" greedy algorithm. Additionally, the optimization following the steepest gradient not necessarily yields the global maximum modularity (which is $\mathcal{NP}$-complete in general (Brandes et al., 2008)), but very often only a good local maximum. For pseudocode, an evaluation of the runtime and modularity quality, and an analysis of equivalent joins, see Ovelgönne et al. (2010).

On the one hand, the randomization significantly speeds up the runtime, on the other hand, errors due to misclassifications may tamper the result. As a consequence, the core group graph clustering schema (CGGC) was developed (Ovelgönne and Geyer-Schulz, 2010; Geyer-Schulz and Ovelgönne, 2014) as an adoption of the learning of a "strong classifier" from "several weak classifiers" (Ovelgönne and Geyer-Schulz, 2013, p. 187). For graph clustering, this means to create several (say k) partitions (not necessarily using RG) and combine them to one core group partition using the (commutative and associative) binary operator

$$\wedge\,(P, Q) = P \wedge Q := \{C_P \cap C_Q \mid C_P \cap C_Q \neq \emptyset,\ C_P \in P,\ C_Q \in Q\} \tag{2.123}$$

defined on two partitions P and Q. The core group partition is then

$$P_{CG} = \bigwedge_i P_i = P_1 \wedge \ldots \wedge P_k. \tag{2.124}$$

This partition induces a graph (as described in Section 2.5.1), which is then used as input for the (slightly modified) RG algorithm. The underlying idea is the interpretation of P_{CG} as a saddle point from which several local optima can be reached and, therefore, also an even better local (hopefully global) optimum. Different local optima lie on different paths through the partition poset and the core group partition is a finer partition of all P_i that were used to create P_{CG}.

2.5.3 Label Propagation

Label propagation (LP) is a method to cluster a graph, which is very easy to understand and implement, but yet very efficient (Biemann, 2006). Most articles cite the paper of Raghavan et al. (2007) as seminal work on using label propagation for graph clustering (e.g. Ugander and Backstrom, 2013; Staudt and Meyerhenke, 2016, to name only two). However, Biemann (2006) published his paper with the exact same algorithm (thus not calling it label propagation) about a year earlier. The main idea is based on propagating messages through a network like, e.g., the `ARP`-protocol (Plummer, 1982), where a computer sends messages to all its neighbors on the local network to find out the physical address of the receiver of a payload message. To use this idea in clustering, each node gets assigned a label that is unique at the start of the procedure. Then all nodes start propagating their labels and make an update of their own label, based on the label they received most often. If labels occur equally often, one of them is chosen randomly. As soon as a node has changed its label, it propagates the new label to its neighbors. The overall process soon converges to a stable solution of labels, and, eventually, these are the clusters, i.e. all nodes with the same label lie in the same cluster. Occasionally, there exist cases where the algorithm does not converge to a stable situation and the labeling oscillates between two or more states. Biemann (2006) argues that this issue can be dealt with by setting a maximum number of iterations or by defining another stop criterion like, e.g., stopping if only few label assignments changed in the last iteration. Raghavan et al. (2007) mention this issue too, but only say that "the mathematical convergence is hard to prove" (Raghavan et al., 2007, p. 036106-9).

This "distributed" approach allows to physically distribute label propagation algorithms over several cores and/or computers (e.g. Staudt and Meyerhenke, 2016, in combination with the CGGC scheme).

2.5.4 Other Approaches

There are numerous other approaches than the two mentioned (RG and its derivatives and label propagation). A whole class are spectral clustering methods (e.g. von Luxburg, 2007), which utilize methods from linear algebra on matrices that describe the graph (see Section 2.1.2.1). Another relatively popular and well performing algorithm is the *Louvain method* (Blondel et al., 2008). An overview of additional algorithms can be found in Ovelgönne (2011). A more recent, but rather short, overview of different perspectives in terms of the general approaches is given by Schaub et al. (2017); Zhao (2017) reviews the "theoretical advances of community detection in networks".

2.6 Entropy

2.6.1 General Definition

The term *entropy* goes back to Clausius (1867, p. 357) in the context of thermodynamics. It was then picked up by Gibbs (1873) and Boltzmann (1896, p. 58) who used it in statistical mechanics. In the 1930s, von Neumann (1996, p. 212ff.) adopted the concept for the theory of quantum mechanics.

Some years later, Shannon (1948) formulated his famous entropy definition in terms of information theory:

$$H(X) := -K \sum_{i=1}^{k} \mathbb{P}(X = x_i) \log \mathbb{P}(X = x_i). \tag{2.125}$$

H is the capital Greek letter η (eta). Here, X is some discrete random variable that has k possible outcomes x_i and $K = const > 0$ is some constant ("the constant K merely amounts to a choice of a unit of measure", Shannon, 1948, p. 12), mostly $K = 1$. The $\mathbb{P}(X = x_i) =: p_i$ are the probabilities of the outcomes x_i to occur. The definition has three properties:

1. H is continuous in p_i
2. $H(X) = -K \sum_{i=1}^{k} \frac{1}{k} \log \frac{1}{k} = K \log k$ is monotonously increasing in k
3. $H(X,Y) = H(X) + H(Y \mid X)$

Furthermore, $H(X) \in [0, K \log k]$. The minimum value is reached if $p_i = 1$ for some i and the maximum value is reached if $p_1 = p_2 = \ldots = p_k = \frac{1}{k}$. $H(Y \mid X)$ is the conditional entropy, which is the weighted sum

$$\begin{aligned} H(Y \mid X) &:= \sum_i p_i H(Y \mid X = x_i) \\ &= -K \sum_i p_i \sum_j \mathbb{P}(Y = y_i \mid X = x_i) \log \mathbb{P}\big(Y = y_j \mid X = x_i\big). \end{aligned} \tag{2.126}$$

of the conditional probabilities. Clearly, $H(Y \mid X) = H(Y)$ if X and Y are statistically independent and $H(Y \mid X) = 0$ if Y is determined by X.

One possibility to interpret the entropy is to view it as measure of uncertainty, which increases the more uncertain the actual outcome of X is. Which logarithm base to use mainly depends on the specific scenario. Often $\log_2 =:$ ld (*logarithmus dualis*) is used, especially if the underlying alphabet is binary. This is true, e.g., for a (minimal) binary encoding, which Shannon (1948, p. 19 f.) uses as an example.

2.6.2 Entropy of Graphs

Bianconi (2009) and Anand and Bianconi (2009) utilize the entropy definition on ensembles of graphs and call it "structural entropy". An ensemble of graphs is a set of graphs that all have the same structural properties (e.g. follow the same degree distribution). The authors state that structural entropy allows a quantification of the role of different structural properties on the overall shape of the networks. These ideas are also reviewed by Garlaschelli et al. (2010, section 3.6). In general, the entropy of an ensemble of graphs is defined as in Equation 2.125 ($K = 1$), where $\mathbb{P}(X = x_i)$ represents the probability $\mathbb{P}(G_i)$ that the graph G_i is picked from the ensemble (Garlaschelli et al., 2010, p. 1698).

Kim and Wilhelm (2008) present and compare several complexity measures in order to find an answer to the question: "What is a complex graph?" Two of their measures are entropy-based. For instance, the *spanning tree sensitivity* (Kim and Wilhelm, 2008, p. 2642) is again defined as in Equation 2.125 ($K = 1$), the sum runs over all edges of the graph, and $\mathbb{P}(X = x_i)$ is replaced by

$$a_l := \frac{S^l_{uv}}{\sum_r S^r_{uv}}. \tag{2.127}$$

The term S^l_{uv} is the sensitivity of the edge uv and it quantifies how sensitive the number of all spanning trees reacts to the removal of uv. A spanning tree is a connected subgraph with $n - 1$ edges. They come to the conclusion that the complexity of a graph is high if it consists of many different subgraphs.

Dehmer and Mowshowitz (2011b) give a historical overview of graph entropy measures. They state the earliest defined measures are due to Rashevsky (1955) and Trucco (1956a,b), both take the graph symmetry into account:

$$H_V(G) = -\sum_{C \in O} \frac{|C|}{|V|} \log \frac{|C|}{|V|} \tag{2.128}$$

$$H_E(G) = -\sum_{C_E \in O_E} \frac{|C_E|}{|E|} \log \frac{|C_E|}{|E|} \tag{2.129}$$

where O (O_E) is the node (edge) orbit partition (see Sections 2.4.2 and 3.2). Both measures tend to zero for very symmetric graphs (as the number of orbits decreases) and to the maximum value for asymmetric graphs. In general, the entropy can be computed on *any* graph partition, but, of course, the generation of the partition needs to be clearly defined, otherwise the measure lacks a meaningful interpretation. Practically, every graph invariant (see Section 3.3) that induces a partition of nodes and/or edges could be used to compute a graph entropy (Dehmer and Mowshowitz, 2011a; Lu, 2017). For instance Raychaudhury et al. (1984) test different "topological indices" for their discriminatory power in distinguishing graphs. A topological index is a term from chemistry, which represents an index that is computed on the graph that describes a molecule (e.g. Todeschini and Consonni, 2008). The indices compared by Raychaudhury et al. (1984) are based on information theory and either take the node degrees or distances between nodes into account. The higher the discriminating power of a topological index, the better can molecules be distinguished by simply comparing the index values, which should be different for different molecules.

3 Graph Morphisms

This chapter will give a quick overview of graph morphisms and gets more into detail on isomorphisms (Section 3.1) and automorphisms (Section 3.2). Any morphism is a function that maps one graph to another graph or onto itself. Different mappings are defined differently and have names ending with "-morphism".

We follow Knauer (2011, Definition 1.4.3 on p. 8) with the following definitions. Given two graphs G and G', a function $\varphi : V(G) \to V(G')$ is a homomorphism $G \to G'$ iff

$$uv \in E(G) \implies \varphi(u)\varphi(v) \in E(G'). \tag{3.1}$$

Knauer (2011) further defines an "egamorphism" (or weak homomorphism), which additionally demands $\varphi(u) \neq \varphi(v)$ as condition for the mapping of an edge to be an edge in the image of G. Godsil and Royle (2001, p. 6) state that requiring loop-free graphs already implies this constraint. The definition in Equation 3.1 makes no assertion on what happens with non-edges ($e \in E(\bar{G})$), therefore they can be mapped again onto a non-edge, an edge, or even a node.

Example 19. *Consider $G = (\{a, b, c\}, \{ab, bc\})$ and let $\varphi(a) = \varphi(c) = x$ and $\varphi(b) = y$. Then φ clearly is a homomorphism from G to $G' = (\{x, y\}, \{xy\})$, both edges in G are mapped to the one edge in G', the non-edge $\{a, c\}$ is mapped onto x.*

Example 20. *Any graph G with n nodes can be mapped to the complete graph K_n by a homomorphism.*

Example 21. *For any graph with chromatic number $\chi(G) = k \leq n$ exists a homomorphism to the complete graph K_k. Godsil and Royle (2001, p. 7) prove this.*

The examples show that homomorphisms are not necessarily constructive, i.e. they are not functions that create a new graph from the given graph G because it is not defined what happens with non-edges.

A strong graph homomorphism $\varphi_s : V(G) \to V(G')$ additionally requires "$\Leftarrow$", so

$$uv \in E(G) \iff \varphi_s(u)\varphi_s(v) \in E(G') \tag{3.2}$$

holds. As φ_s needs not to be bijective, $|V(G)| \neq |V(G')|$ may be true.

Example 22. *$G = (\{a, b, c, d\}, \{ab, cd\})$ and $G' = (\{1, 2\}, \{\{1, 2\}\})$ with the associated function*

$$\varphi_s : \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \mapsto \begin{pmatrix} 1 \\ 2 \\ 1 \\ 2 \end{pmatrix} \tag{3.3}$$

is a strong homomorphism.

Example 23. *Let G be a graph given a partition P of its node set. The function $cid(u, P) :=$ $\min \{C \mid u \in C\}$ returns the cell id for each node u of G, which is simply the smallest node id of the cell u is part of. The mapping*

$$\varphi_P : \begin{pmatrix} 1 \\ 2 \\ \vdots \\ n \end{pmatrix} \mapsto \begin{pmatrix} cid(1, P) \\ cid(2, P) \\ \vdots \\ cid(n, P) \end{pmatrix} \tag{3.4}$$

is a strong homomorphism and maps G to the quotient graph given P (see Section 2.3.2).

3.1 Graph Isomorphism

A function φ that is a strong homomorphism *and* bijective is called isomorphism (Knauer, 2011, p. 8). Isomorphic graphs are completely identical in terms of adjacency structure, the only difference is the way of labeling the nodes. Graph isomorphism induces an equivalence relation, which is denoted "$\cong$".

As simple as the definition of this property is, as hard is its recognition (answering the question "Are the graphs G and G' isomorphic?") and computation of the mapping function $\varphi : G \to G'$ (see also Section 3.4). The graph isomorphism problem (GI) tends to be neither in $\mathcal{P}$ (the class of problems solvable in polynomial time) nor in $\mathcal{NPC}$ ($\mathcal{NP}$-complete; the class of "the hardest problems in [$\mathcal{NP}$]"[1]) (e.g. Lubiw, 1981; Babai and Luks, 1983; Fortin, 1996; Hartke and Radcliffe, 2009). For a comprehensive explanation of these problems, see Gary and Johnson (1979). For an introduction to complexity theory and an overview of complexity classes, see Johnson (1990) (who coincidentally uses GI as introductory example), and for an extensive overview of complexity classes, see Aaronson et al. (2017). Instead of delving into complexity theory any further, it is sufficient to mention that problems in $\mathcal{P}$ are seen as not very problematic, because algorithms exist that solve the problem in polynomial time $\mathcal{O}(n^k)$ (n is a variable depending on the input problem size and $k \geq 1$ a positive constant independent of n). $\mathcal{P}$ is a subset of $\mathcal{NP}$, so problems in $\mathcal{NP} \setminus \mathcal{P}$ are those that are not easy to solve in general, but given a possible solution, correctness can be verified in polynomial time. A corresponding

[1] `https://complexityzoo.uwaterloo.ca/Complexity_Zoo:N#npc` as of May 2017; calligraphic font series added

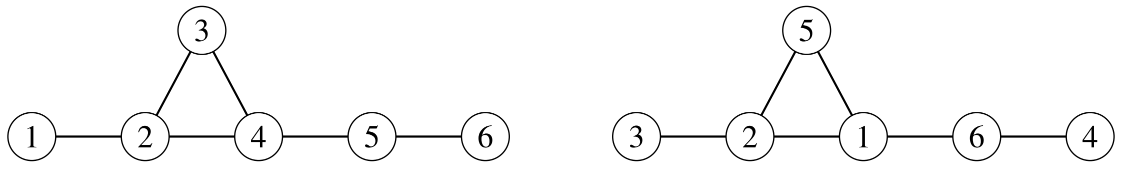
problem is subgraph isomorphism (SGI), which is known to be in $\mathcal{NPC}$. SGI is about solving the problem whether a graph G contains a subgraph that is isomorphic to a graph H (Gary and Johnson, 1979, p. 202).

Applications of graph isomorphism recognition are manifold. One example is given by Rensink (2007) who states that "isomorphism checking can be used as an effective technique for symmetry reduction in graph-based state spaces" (Rensink, 2007, p. 1). Other applications are to determine the equivalence of molecular structures (Balaban, 1985; Li et al., 2008), of biochemical networks (Bonnici et al., 2013, e.g. "protein-protein interaction, metabolic interaction, transcription factor binding, and hormone signaling networks", p. 1), protein-protein interaction network alignment (Elmsallati et al., 2016, an application of SGI), network motif discovery (Grochow and Kellis, 2007, also an application of SGI), pattern recognition/image processing (Conte et al., 2003; Sanfeliu et al., 2002; Sharma et al., 2012), and social network analysis (Fan, 2012).

Finding a mapping function from one graph to another is obviously equivalent to finding functions c, c' that label the nodes of graphs G, G' in a way that $c(G) = c'(G') \iff G \cong G'$. It is called *canonical labeling*, as it relabels the nodes of a graph in a deterministic way, independent of the original labeling, so that eventually all isomorphic graphs have the same labeling. For any "somehow" labeled graph exists a trivial isomorphism to a graph labeled $1, 2, \ldots, n$, which can easily be constructed by mapping the label to its corresponding row/column index of the adjacency matrix. Therefore, it is sufficient to restrict analysis on graphs labeled like that. It follows that each isomorphism is a permutation of the set of nodes.

Two graphs G, G' with adjacency matrices $\boldsymbol{A}_G, \boldsymbol{A}_{G'}$ are isomorphic if there exists a permutation matrix $\boldsymbol{\Phi}$ so that $\boldsymbol{A}_{G'} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{A}_G \boldsymbol{\Phi}$ is true. This relation also clarifies the fact that isomorphic graphs only differ in their labeling of nodes, which is principally arbitrary (we pointed this out already in Section 2.1.2).

Example 24. *The two graphs in Figure 3.1 are isomorphic.*



(a) The graph G (b) The graph G'

Figure 3.1: Two isomorphic graphs with the permutation function $\varphi = (1\,3\,5\,6\,4)$ that maps G to G'. Of course, the inverse $\varphi^{-1} = (1\,4\,6\,5\,3)$ maps G' to G.

The adjacency matrix of G is

$$A_G = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.5}$$

and the permutation matrix corresponding to $\varphi = (1\,3\,5\,6\,4)$ *is*

$$\mathbf{\Phi} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{3.6}$$

Computing $\mathbf{\Phi}^{\mathrm{T}} \boldsymbol{A}_G \mathbf{\Phi}$ *yields (row/column labels are the "old" node labels of G)*

$$(\mathbf{\Phi}^{\mathrm{T}} \boldsymbol{A}_G)\mathbf{\Phi} = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 4 & 0 & 1 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 1 \end{array} \cdot \mathbf{\Phi} = \begin{array}{c|cccccc} & 4 & 2 & 1 & 6 & 3 & 5 \\ \hline 4 & 0 & 1 & 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \\ 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 & 1 & 0 & 0 \end{array} = A_{G'}. \tag{3.7}$$

3.2 Graph Automorphism

A graph automorphism is an isomorphism from a graph to itself. Informally, this means there exist structurally identical nodes so that a relabeling of them preserves adjacency completely. Isomorphisms are permutations, so are automorphisms. But additionally, the set of all automorphisms of a graph forms a permutation group denoted by $Aut(G)$, the automorphism group of G (acting on the set of nodes V). For the group size, $|Aut(G)| \geq 1$ holds. Equality means that only the trivial permutation, which maps each node to itself, is a valid automorphism for the given graph. We will abbreviate the graph automorphism problem with GA.

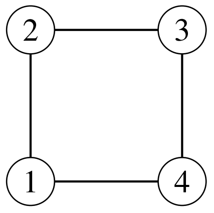
The automorphism group captures any symmetries a graph contains and, analogously to isomorphism, for an induced permutation matrix $\boldsymbol{P}$ that is an automorphism and the adjacency matrix $\boldsymbol{A}_G$, the relation $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{A}_G\boldsymbol{P} = \boldsymbol{A}_G$ holds. $Aut(G)$ is formally defined as

$$Aut(G) := \left\{ g \in S^{V(G)} \mid e \in E \iff e^g \in E \right\}, \tag{3.8}$$

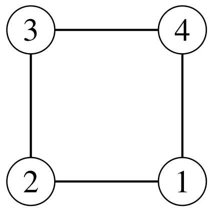
where $S^{V(G)}$ is the symmetric group that acts on the set of nodes. Of course, any other definition for finite permutation groups, which we presented in Section 2.4, is applicable on the automorphism group of a graph. For example, the orbit partition of G arranges the nodes in equivalence classes.

The difference between isomorphisms and automorphisms can informally be explained with graph drawing: When two isomorphic graphs (e.g. those in Figure 3.1) with trivial automorphism group are drawn on a plane with some fixed layout omitting the labels, both drawings are exactly the same and unique. Achieving this is of course not trivial, as it is equivalent to finding a canonical mapping for each graph (which *is* the layout, i.e. coordinates on a plane). For a graph with a non-trivial automorphism group, there exist several different drawings that (again omitting labels) all look the same, but nodes on the same orbit can be permuted in the drawing. However, the different drawings cannot be distinguished without having the labels. It follows that the mapping of two automorphic graph "instances" (which are of course also isomorphic) is not unique.

Example 25. *The two graphs in Figure 3.2 are automorphic.*



(a) A graph G



(b) A graph G'

Figure 3.2: Two automorphic graphs with the permutation function $p = (1\ 2\ 3\ 4)$ that maps G to G'.

The adjacency matrix of G is

$$\boldsymbol{A}_G = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \tag{3.9}$$

and the permutation matrix corresponding to $p = (1\ 2\ 3\ 4)$ *is*

$$\boldsymbol{P} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{3.10}$$

Computing $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{A}_G\boldsymbol{P}$ yields

$$\boldsymbol{P}^{\mathrm{T}}\boldsymbol{A}_G\boldsymbol{P} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \boldsymbol{P} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} = A_{G'} = A_G. \tag{3.11}$$

It is confirmed what already was obvious, both graphs are automorphic. Actually $G = G'$, so they are not two graphs, but the same identical graph. However, contrary to the isomorphism example in Section 3.2, also, e.g., for

$$\boldsymbol{P}^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{3.12}$$

the relation $(\boldsymbol{P}^2)^{\mathrm{T}}\boldsymbol{A}_G\boldsymbol{P}^2 = \boldsymbol{A}_G$ holds.

G and G' are both isomorphic to the cycle graph C_4, which is the only connected graph that is 2-regular. We mentioned this graph before in Section 2.1.4, now it is clear why cycle graphs are unique up to isomorphism.

Erdős and Rényi (1963) proved that graphs tend to be asymmetric with probability 1, the larger (concerning the node set) they become (see also Cameron, 1983, p. 107). This general result is true for the class of all finite graphs. Magner et al. (2014) proved the same for graphs that are created following a preferential attachment model (see Section 2.2.3), where the attachment parameter c controls the number of neighbors a newly added node becomes associated with by an edge. They come to the conclusion that for a parameter $c \in \{1, 2\}$ the resulting graph will contain symmetries with a positive probability. However, for $c \geq 3$, the resulting graph is very likely to be asymmetric.

Contrary to these theoretical results, we will give evidence in Chapter 4 that one cannot automatically conclude that there are not many symmetric graphs (real-world networks), which maybe have an effect on graph clustering. This fact implies that preferential attachment is not sufficient to describe how real-world networks are formed.

We end this section by presenting the theorem of Frucht (1939).

Theorem 9 (Theorem of Frucht)**.** *For every abstract finite group there exist infinitely many finite connected graphs with an automorphism group isomorphic to this group.*

Proof. See Frucht (1939). □

Theorem 9 directly connects the theory of finite groups (especially permutation groups) to the theory of graph symmetry.

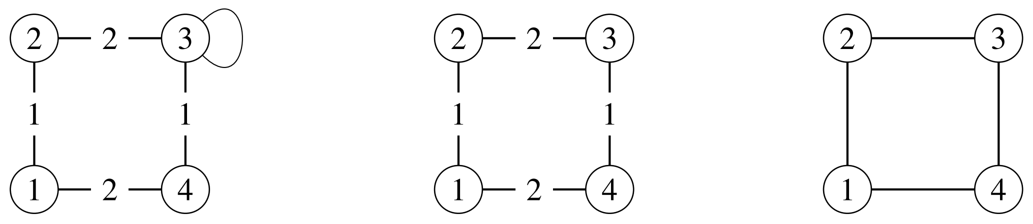
3.2.1 Simplification of Graphs

Automorphisms were only formally defined for simple graphs, but the concept of symmetry can easily be extended for non-simple graphs. To do so, we want to interpret an edge between two nodes as a relation, which can be described by a function $e : V \times V \to \mathbb{B} = \{0, 1\}$. If an edge between the nodes u and v exists, $e(u, v) = 1$. A permutation g is, as defined above, an automorphism of the graph if $e(u, v) = e(u^g, v^g)$ (for all $u, v \in V$) if we follow this interpretation.

Certainly, arbitrary relations could be possible, e.g. $w_n : V \to \mathbb{R}$ or $w_e : V \times V \to \mathbb{R}$, which assign a weight to a node or an edge, or $c : V \to \mathbb{N}$, which assigns a color to a node. Each of those functions describes a certain "property" and the abstract definition of an automorphism is to retain all properties, i.e. that all functions that describe these properties of the given graph are invariant under transformation.

Therefore, each property restricts the "degrees of freedom" of symmetry and, the other way round, removing a property adds a degree of freedom. For example, the graph that has no properties at all is just the empty graph, which has the maximally possible degree of freedom, and—coincidentally—has the symmetric group as automorphism group.

Simplification of a graph means to remove a property by keeping all other properties the same. As a consequence, the automorphism group of a structurally identical graph with additional properties G' is a subgroup of the automorphism group of G: $Aut(G') \leq Aut(G)$. An example of how simplification influences the automorphism group is shown in Figure 3.3.



(a) The weighted graph G_{w+l}, which also has a loop at node 3. It has the trivial automorphism group $Aut(G_{w+l}) = \{\mathbf{1}\}$.

(b) A simplified (loops removed) version G_w of the graph G_{w+l} with the following symmetries: $Aut(G_w) = \{\mathbf{1}, (1\,2)(3\,4), (2\,3)(1\,4), (1\,3)(2\,4)\}$.

(c) The simple (weights removed) graph G, which has the automorphism group $Aut(G) = Aut(G_w) \cup \{(1\,3), (2\,4), (1\,2\,3\,4), (1\,4\,3\,2)\}$.

Figure 3.3: Example of a two-step simplification of the graph G_{w+l} (a), which is asymmetric. G_w (b) is still weighted but has a non-trivial automorphism group, and G (c) is simple with an even larger automorphism group. Each automorphism group of the more complex graph(s) is a subgroup of the simpler graph(s): $Aut(G_{w+l}) \leq Aut(G_w) \leq Aut(G)$.

From a practical point of view, it is necessary to carefully think about the possible consequences of such a simplification. It is particularly important to consider if the simplification, besides the effect on symmetry, is valid for the given application context. For instance in Section 4.3.2, we simplify graphs before we analyze their symmetry and argue that omitting weights and directions of edges is acceptable in a clustering context, as many clustering algorithms do not take them into account anyway.

3.2.2 Further Concepts

For now, we only defined and have looked at graph automorphisms as permutations that preserve adjacency between nodes. However, it is also possible to define the edge automorphism group that is defined as

$$Aut_E(G) = \left\{ g \in S^{V(G)} \mid uv, uw \in E \iff u^g v^g, u^g w^g \in E \right\}. \tag{3.13}$$

Hence, it is the group of permutations that preserve edge incidence.

The transitivity property of a group (only one orbit exists) may also hold for $Aut_E(G)$, but in general, neither node transitivity implies edge transitivity nor vice versa (e.g. Lauri and Scapellato, 2016, p. 10 ff.). That means there exist graphs that are node transitive but not edge transitive, edge transitive but not node transitive, or even node *and* edge transitive. Even "stronger" properties of the automorphism groups exist (like distance transitivity), but we do not want to go into further detail, as these properties are highly unlikely to hold in real-world graphs. Even transitivity (whether on the set of nodes or edges) is implied by a very high level of symmetry, which is usually not found in practical applications (see Chapter 4 and Appendix B). Already, the power-law distribution property of the node degrees (see Section 2.2.1) heavily restricts the automorphisms that are theoretically possible, as nodes on the same orbit must have the same node degree.

3.2.3 Symmetry Bounds

It is obvious that the complete graph K_n has an automorphism group isomorphic to S_n. A question that could be posed is: How many edges need to be removed from K_n so that the resulting connected graph G has $|Aut(G)| = 1$? Quintas (1967, p. 64) gives lower (upper) bounds for the number of edges m_n (M_n) an asymmetric graph with n nodes can have. Next, we want to present these bounds and give an example. The definitions given by the author are for $n = 1$ or $n > 5$, because there exist no graphs with $n = 2, 3, 4, 5$ nodes that are asymmetric (e.g. Erdős and Rényi, 1963). Remember, we are only interested in connected graphs.

The lower bound m_n is given by

$$m_n = \begin{cases} 0 & , n = 1 \\ 6 & , n = 6 \\ n - 1 & , n > 6 \end{cases} \tag{3.14}$$

and the upper bound M_n by

$$M_n = \begin{cases} 0 & , n = 1 \\ 9 & , n = 6 \\ 15 & , n = 7 \\ \frac{n(n-3)}{2} + \sum_{i=1}^{N} a_i + w & , n > 7 \end{cases} \tag{3.15}$$

where a_i is "the number of asymmetric trees having n [nodes]" (Quintas, 1967, p. 58). This number is derived by Harary and Prins (1959) and can also be found in OEIS Foundation Inc. (2017, sequence A000220). The quantity N can be determined by the inequality

$$\sum_{i=1}^{N} a_i \cdot i \leq n < \sum_{i=1}^{N+1} a_i \cdot i, \tag{3.16}$$

and w must obey the constraints

$$n = \sum_{i=1}^{N} a_i \cdot i + w(N+1) + r, \tag{3.17}$$

$$0 \leq w < a_{N+1}, \text{ and} \tag{3.18}$$

$$0 \leq r < N + 1. \tag{3.19}$$

These bounds induce the existence of connected asymmetric graphs with $n = 1$ or $n > 5$ nodes that have at least m_n or at most M_n edges. Note that a graph with $n > 6$ nodes and less than m_n edges is not connected anymore and the n nodes can always be connected in a way that this disconnected graph is symmetric. Graphs with $n > 6$ nodes and more than M_n edges are always symmetric and graphs whose number of edges lies between the lower and upper bound are either symmetric or asymmetric.

Example 26. *Let $n = 8$. The lower bound can be directly calculated from Equation 3.14, which yields $m_8 = 7$. For the upper bound, we first need to calculate N by using the values from*

n	1	2	3	4	5	6	7	8	9	10	...
a_n	1	0	0	0	0	0	1	1	3	6	...

Table 3.1: The first ten values for the number of asymmetric trees having n nodes

Table 3.1 and inequality 3.16:

$$\begin{aligned} \sum_{i=1}^{N} a_i \cdot i &= 1 \cdot 1 + 0 \cdot 2 + \ldots + 0 \cdot 6 + 1 \cdot 7 \\ &= 8 \leq 8 < 16 \\ &= 1 \cdot 1 + 0 \cdot 2 + \ldots + 0 \cdot 6 + 1 \cdot 7 + 1 \cdot 8 = \sum_{i=1}^{N+1} a_i \cdot i \end{aligned} \tag{3.20}$$

and we find $N = 7$. Second, we find w by

$$8 = \sum_{i=1}^{7} a_i \cdot i + w(7+1) + r = 8 + 8w + r \tag{3.21}$$

$$\iff \quad 0 = 8w + r \tag{3.22}$$

$$\iff \quad 0 = w = r \qquad (\textit{because } w, r \geq 0) \tag{3.23}$$

This finally lets us calculate $M_8 = \frac{8(8-3)}{2} + (1+1) + 0 = 22$. Because $|E(K_8)| = \binom{8}{2} = 28$ we can conclude that any connected graph with $n = 8$ nodes and more than 22 edges must be symmetric. The graph shown in Figure 3.4 has exactly $m_8 = 7$ nodes and is an example for an asymmetric graph.

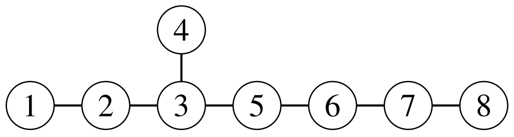


Figure 3.4: An example of a connected asymmetric graph that has the minimum number of edges $m_8 = 7$ for which there must exist such an asymmetric graph.

3.3 Graph Invariants

We have seen in the two preceding sections that isomorphisms (and automorphisms) preserve graph structure. This implies that a lot of other properties of graphs are also preserved, but are probably far less expensive (in terms of complexity) to be computed. Formally,

$$G \cong G' \implies \text{inv}(G) = \text{inv}(G') \tag{3.24}$$

for some invariant inv. The opposite implication does not hold. If an invariant could be found that holds in both directions, GI would be solved if the invariant can be obtained with reasonable complexity (Read and Corneil, 1977, p. 344). Still, invariants are useful for pruning, fast checks, and pre-classification of, e.g., nodes prior to using an algorithm. For instance, only graphs with the same number of nodes and edges can be isomorphic by definition, both values are available (mostly) “for free”. Other invariants are based on spectral properties (an overview is given by Read and Corneil, 1977, p. 345) or distances between nodes (e.g. Corneil and Kirkpatrick (1980, p. 287) and Khalifeh et al. (2009)). Also Dehmer et al. (2013) provide a good overview of different invariants in their introduction as well as a method that combines several invariants for a better discriminating power.

In chemistry, graph invariants are often used to distinguish different molecules (e.g. Basak et al., 1991). De Melo et al. (2013) propose a polynomial time invariant to test for graph isomorphism. It is based on the quadratic assignment problem, which is an optimization of assigning n entities of one set to the n entities of another disjoint set. A graph that is represented by a quadratic adjacency matrix can be understood as the instance of either a flow matrix or a distance matrix (de Melo et al., 2013, pp. 165 f.) that are both the input of a quadratic assignment problem (each matrix represents one set of n entities). They state that the problem solution for two isomorphic graphs G and H is a permutation that maps one graph onto the other. However, the invariant is not a solution of the assignment problem itself—which is $\mathcal{NP}$-hard according to the authors—but to calculate the variances for the three assignment problems (G, G), (G, H), and (H, H) for the two graphs G and H. The computational complexity of the variances is of polynomial order and if $G \cong H$, these variances are equal.

Another noticeable invariant is a special partition of the node set, called an equitable partition. The invariant differs from the ones above, as it is an invariant on the node level. It is used by algorithms like `nauty` (see Section 3.4) as a starting point for the search of generators of the automorphism group. A graph partition is called *equitable* if every node $u \in V$ has the same degree to all other cells. Let

$$\deg(u, C) := |\{uv \in E \mid v \in C, C \subseteq V\}| \tag{3.25}$$

be the constrained node degree given some subset (partition cell) C. An equitable partition P^e is defined as

$$P^e := \left\{C_1^e, \ldots, C_k^e \mid \forall u, v \in C_i^e : \deg(u, C_j^e) = \deg(v, C_j^e),\ i, j = 1, \ldots, k\right\}, \tag{3.26}$$

and, as a consequence, all nodes in an equitable cell C_i^e have the same node degree. The idea behind this definition is simple and it is based on two natural properties:

1. Isomorphic nodes must have the same node degree, and
2. the neighbors of isomorphic nodes must have the same node degrees.

All nodes contained in the same cell of an equitable partition can possibly be isomorphic, nodes contained in different cells can not be isomorphic.

3.4 Algorithms

In this section an extensive overview of algorithms that solve the graph isomorphism and/or automorphism problem is given. We will go into more detail for those algorithms that are capable of computing a set of generators for the automorphism group, as this is the application we will need for our analyses in Chapters 4 and 7. To classify different algorithms, we first distinguish three different manifestations of the same problem in general and explicitly for GI and GA in Table 3.2.

Mathon (1979) additionally gives the problem manifestations, which he calls *AGEN(G)* (find the generators of $Aut(G)$) and *APART(G)* (find the orbit partition of $Aut(G)$), and proves some equivalence results of the issues.

	Abstract problem	GI	GA
Decision	Is there any solution to it?	Are the graphs G and H isomorphic?	Is there a non-trivial automorphism that maps G onto itself?
Counting	How many solutions exist?	How many isomorphic mappings from G to H exist?	How many such mappings exist, i.e. what is $\|Aut(G)\|$?
Enumeration	Which are the solutions?	Explicitly get all isomorphic mappings from G to H.	Which are the $g \in Aut(G)$?

Table 3.2: The three abstract problem manifestations and what they mean for graph isomorphism (GI) and graph automorphism (GA).

When comparing GI and GA, the counting and enumeration problems for both are equivalent: Consider $G \cong H$ and $|Aut(H)| > 1$. Counting the isomorphisms $\phi_i : G \to H$ is obviously the same as finding one $\phi : G \to H$ and $|Aut(H)|$, as for all $g \in Aut(H)$ the function $\phi \circ g$ is defined as $\phi \circ g : G \overset{\phi}{\to} H \overset{g}{\to} H$. This means each $\phi \circ g$ is an isomorphism from G to H and there exist exactly $|Aut(H)|$ of them. Moreover, also $Aut(G) = Aut_{\phi^{-1}}(H)$ holds with $Aut_{\phi^{-1}}(H) = \{g \circ \phi^{-1} \mid g \in Aut(H)\}$. From the latter relation, the equivalence of the enumeration problems becomes clear as well. The decision problem, however, is not equivalent for GI and GA: Asking if G is isomorphic to G is trivially always true, as at least the identity $G^{\mathbf{1}} = G$ exists. In contrast to that, deciding if G and $H \neq G$ are isomorphic has no non-trivial solution.

Booth and Colbourn (1979) provide proofs for many problems to be *isomorphism complete*, i.e. they can be reduced to GI in polynomial time. As a consequence, these problems are as complex as graph isomorphism. Examples of isomorphism complete problems involve many special graphs (e.g. directed graphs, bipartite graphs), lattices, and combinatorial designs. The authors do not provide any pseudocode or algorithms, however, the contribution shows that there cannot exist any method that solves one of all these equivalent problems in, say, linear time while all others are still in $\mathcal{NP}$.

Most algorithms that are presented in the following are intended to solve GI by either producing a concrete mapping between the two graphs or by determining a canonical labeling of the nodes, which, as we have seen above, must be the same for isomorphic graphs. Our survey of algorithms begins in the mid of the 1970s; an overview of older methods is, e.g., given by Read and Corneil (1977). We loosely adhere to the following order: First, algorithms for special classes of graphs are reviewed and then we come to algorithms that solve the general case. In each subsection, we always try to retain the order of appearance of the different articles.

3.4.1 Efficient Algorithms for Special Graphs

Planar Graphs One of the first notable efficient algorithms for handling GI of planar graphs is by Hopcroft and Wong (1974). The word "efficient" is used a bit arbitrary in this context and only means "better than brute-force", i.e. better than complete enumeration of all possible solutions. They present an $\mathcal{O}(n)$ algorithm that determines isomorphism and—after modifying the procedure—returns an isomorphic mapping if one exists. Although pseudocode is provided, the authors emphasize that their results are rather theoretical than practical. Kukluk et al. (2004) evaluate their own method, which is based on results by Hopcroft and Wong (1974), and compare it with other algorithms that follow a general approach. Surprisingly, their method is outperformed in terms of runtime by the general algorithms (presented in Section 3.4.3). This is admitted by the authors in their conclusion.

Trees In their book on the design and analysis of algorithms, Aho et al. (1974, pp. 84/85) present a linear time algorithm to determine tree isomorphism. It successively labels the nodes, starting at the leaves, and assigns to all inner nodes on each level of the tree a sorted tuple of labels of its children. Each distinct tuple per level gets assigned its own label. The algorithm stops if the sequences of tuples on one level are different for the two trees that are compared. As a consequence, if the root node is reached, the tuples of labels in the root node for both trees must be equal, too. This algorithm is an example of a problem solver that does not yield an isomorphic mapping between the two graphs but only decides GI. A quite didactic approach to motivate tree isomorphism is by Campbell and Radford (1991), who build upon the results of Aho et al. (1974). A polynomial time algorithm to solve the automorphism group counting problem for trees is given by Zhang et al. (2012).

Interval Graphs An $\mathcal{O}(n+m)$ algorithm is presented by Lueker and Booth (1979) for interval graphs, which are graphs constructed from intervals in $\mathbb{R}$. Given a set of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ with $I_i = [a_i, b_i]$, $a_i \leq b_i$, each interval represents a node of the graph and two nodes I_i, I_j $(i \neq j)$ are adjacent iff $I_i \cap I_j \neq \varnothing$ (i.e. either $a_j \in I_i$ or $b_j \in I_i$). For example, $\mathcal{I} = \{I_1, I_2, I_3\} = \{[4,6],[5,7],[7,8]\}$ implies $G_{\mathcal{I}} = (\{I_1, I_2, I_3\}, \{\{I_1, I_2\}, \{I_2, I_3\}\})$. The method first constructs a special tree (“PQ-tree”, where P and Q stand for different types of tree nodes) from the graph and then creates a labeled version of it. The authors prove that both sub-algorithms have complexity $\mathcal{O}(n+m)$ with n the number of nodes and m the number of edges. Two such graphs are isomorphic if the trees are isomorphic, which can be tested by a modified version of the algorithm of Aho et al. (1974).

Graphs of Bounded Genus The genus of a graph is defined as the minimal number γ so that the graph can be embedded in an orientable surface S_γ (Gross and Tucker, 2001, pp. 24 ff.), i.e. it can be drawn without crossing edges. Informally, the genus is the number of holes in the surface. The surface S_0 is a sphere and all planar graphs, which can be drawn on a plane, can also be drawn on a sphere. Therefore, planar graphs have genus 0. For example, the complete graph K_5 is not planar, so its genus must be positive. In fact, K_5 has genus $\gamma = 1$ and that means it can be embedded into S_1, which is, e.g., a torus (the surface of a “doughnut”, i.e. a surface with a hole in it; see Figure 3.5 for an embedding of K_5 in S_1).

A polynomial time algorithm for isomorphism testing of graphs of bounded genus is given by Miller (1980). His method computes a *succinct code*, which is the same as a canonical

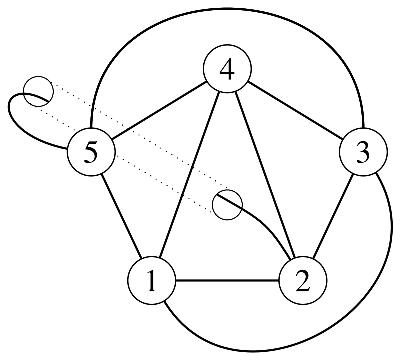


Figure 3.5: An embedding of the complete graph K_5 onto the surface S_1. All edges but $\{2,5\}$ can be drawn on a plane without crossings. However, $\{2,5\}$ can be drawn through the hole that exists for S_1, which circumvents the crossing of at least $\{1,3\}$.

labeling defined above. Besides providing pseudocode, the author proves his algorithm to have a complexity of $n^{\mathcal{O}(\gamma)}$.

The results of Miller (1980) coincide with those of Hopcroft and Wong (1974), as planar graphs of genus 0 have complexity $n^{\mathcal{O}(0)} = n^{\mathcal{O}(1)} = \mathcal{O}(n^c) \approx \mathcal{O}(n)$ for small c. It is, however, questionable if the result is still correct, as Myrvold and Kocay (2011) have recently published results that the algorithm for the embedding of graphs, on which the method is based on (and other algorithms as well), is incorrect and actually has exponential complexity. They argue in their conclusion that there are newer attempts to actually provide a polynomial or even linear time algorithm (e.g. Mohar, 2006), but it is not clear if the proposed methods can be correctly implemented.

Permutation Graphs A permutation graph is defined for a permutation p on $V = \{1, \ldots, n\}$ as $G_p = (V, \{ij \mid \forall i, j \in V : (i-j)(p(i) - p(j)) < 0\})$. This means there exists an edge for every two nodes where p inverts the ordering of the nodes under the permutation.

A polynomial time algorithm for this special class of graphs is given by Colbourn (1981). He provides a method that tests GI by computing a canonical labeling of the graph. The algorithm's complexity is of order $\mathcal{O}(n^3)$.

Graphs of Bounded Valence Luks (1982) presents a method to test isomorphism of graphs that have bounded valence (i.e. degree). However, the algorithm does not directly solve the isomorphism problem, but the polynomial-time reducible, and thus equivalent, color automorphism problem. The rough idea of the method is to join the two possibly isomorphic graphs G and H and then find—given a set of generators for a permutation group that acts on a set of colors—those generators that generate the subgroup of permutations that preserve the colors. The details are relatively complicated. Luks (1982) uses many group theoretic tricks to reduce the problem's complexity. Furthermore, he states in the notes at the end of his article that the complexity of $\mathcal{O}(n^5)$ of the algorithm could be further reduced by several authors.

Soon after, promising results were published by Babai and Luks (1983) who reduced the worst case complexity for general graphs to $\exp(n^{\frac{1}{2}+\mathcal{O}(1)})$, which was a long standing upper bound for GI until the year 2015 (see below).

Compact and Non-isomorphic Graphs A graph G is said to be *compact* if the convex hull of its automorphism group equals the set of doubly stochastic matrices that commute with the adjacency matrix of G (Brualdi, 1988). The graph automorphisms are represented as permutation matrices as defined in Section 3.2, and the convex hull of the group is the set of all linear combinations

$$\overline{Aut(G)} := \left\{ \sum_i \lambda_i \boldsymbol{P_i} \mid \sum_i \lambda_i = 1,\ \boldsymbol{P_i} \in Aut(G) \right\}. \tag{3.27}$$

A matrix is called doubly stochastic if all rows and columns sum up to one. Clearly, each permutation matrix is doubly stochastic and so are all the matrices in the convex hull $\overline{Aut(G)}$. Moreover, a matrix $\boldsymbol{X}$ commutes with another matrix $\boldsymbol{A}$ if $\boldsymbol{AX} = \boldsymbol{XA}$ holds.

Tinhofer (1991) gives a polynomial time algorithm for GI that yields the correct result for any pair of non-isomorphic graphs or if one of them is compact. For non-compact isomorphic graphs, the correctness depends on the graphs' structure. Brualdi (1988) also proves that trees are compact graphs, i.e. the result of Tinhofer (1991) coincides with the complexity of tree isomorphism. The algorithm's result is an actual isomorphic mapping between the two graphs if it exists. Interestingly, the approach of the algorithm is to successively refine partitions of the node set that are unaffected by $Aut(G)$. This general procedure is applied by most of the algorithms for GA that we discuss in Section 3.4.3.

Graphs of Bounded Average Genus The average genus of a graph is defined as

$$\gamma_{avg} = \frac{\sum_i i \cdot g_i}{\sum_i g_i} \tag{3.28}$$

with g_i being the number of embeddings into the surface S_i.

Chen (1992) presents a linear time algorithm to check GI for graphs of bounded average genus. The basic idea of the method is to combine "topological invariants with combinatorial analysis" (Chen, 1992, p. 104) that allows to compute a finite number of "frames" for the graphs of bounded average genus. A "frame" of a graph G is derived from it by the deletion of several edges in G. As a consequence, the frames must be isomorphic iff the two compared graphs are isomorphic and the distribution of a subset of edges is equal.

Circulant Graphs A graph is considered *circulant* if it is isomorphic to a graph for which the permutation $(1\ 2\ \dots\ n)$ is an automorphism. This includes of course the cycle graph C_n as well as the complete graph K_n.

The second to last efficient algorithm is by Muzychuk (2004) and applicable for this type of graphs. We note here that the author only gives an idea how an actual polynomial time algorithm (with complexity $\mathcal{O}(n^2)$) could look like. Therefore, it is not clear how difficult it would be to implement the method. A different approach, which comes to similar results, is presented by Evdokimov and Ponomarenko (2004).

Graphs with Excluded Minors A graph minor is a subgraph of a graph that is obtained by the contraction of edges, i.e. connected nodes are merged into a single node by keeping the edges to all other nodes. Grohe (2010) proves that isomorphism decidability for graphs with excluded minors is in polynomial time.

3.4.2 General Algorithms

Corneil and Gotlieb (1970) published a procedure that is based on the first author's PhD thesis, and it consists of several parts. It is based on refining the set of nodes into cells of equivalent nodes in terms of their degrees. In fact, these partitions are equitable as defined in Section 2.3.2, although the authors do not name them as such. Recall that all nodes in a cell have exactly the same number of edges to all other cells and that this partition is invariant under automorphism. From the computed equitable partition the so called *directed quotient graph* is formed. It has the partition's cells as nodes and directed weighted edges that represent the number of edges each node in a cell has to another cell. In a second step, each cell is further refined by assigning a new label to all its contained nodes and then the equitability property is restored. This procedure results in several finer partitions as well as their corresponding quotient graphs. These quotient graphs are isomorphic iff they are equal. The refinement is repeated until no cell was refined anymore.

The final result of the second step is the so called *terminal quotient graph*, which is conjectured to be the *automorphism partitioning* of the graph, i.e. all nodes in a cell are "somehow" mapped onto each other. This definition corresponds to the orbit partition of the graph, however, this is not mentioned by the authors. Indeed, Corneil and Gotlieb (1970, Figure 3) give a counterexample of two non-isomorphic graphs that have the same terminal quotient graphs and this facts leads them to the definition of the *representative graph*, which is derived from the terminal quotient graph. The authors state that the equality of the derived representative graphs from the two graphs that are tested for isomorphism is a necessary condition and it allows to determine non-isomorphism if this condition does not hold.

In the last step, the *reordered graph* is derived to be a sufficient condition for graph isomorphism. As the name tells, a reordering (of the node labels) is equivalent to a canonical labeling. However, on the basis of the authors' conjecture, only equality of the two derived reordered graphs implies isomorphism. Non-equality would be a counterexample to the conjecture, which could be resolved by "a deterministic nonefficient heuristic procedure" (Corneil and Gotlieb, 1970, p. 52). The overall complexity depends on whether there exists a "transitive h-strongly regular" subgraph and it is $\mathcal{O}(n^5)$ if no such subgraph exists, $\mathcal{O}(n^{5+h})$ otherwise. The two flaws of the algorithm are (i) that it relies on the conjecture that the terminal quotient graph is the automorphism partitioning and (ii) that h is only bounded by n, which could result in an $\mathcal{O}(n^n)$ worst-case complexity. Nonetheless, the method involves many ideas on which later algorithms are based, like, e.g., the refinement of partitions.

In his article on Monte-Carlo algorithms in graph isomorphism testing, Babai (1979) is the first who coins the term "Las Vegas" algorithm. Both general types of algorithms are probabilistic, which means they do not yield deterministic results (Babai, 1979, pp. 2/3):

Monte-Carlo algorithm Given some input x and the expected result $y = f(x)$, there is a certain error smaller than $\frac{1}{3}$ that $f(x) \neq y$.

Las Vegas algorithm Given some input x and the expected result $y = f(x)$, there is a probability less than $\frac{1}{2}$ that $f(x)$ can not be computed and the result is "?".

In other words, the former method always gives a result but with a certain error, the latter always gives a correct result but probably no result at all. The contribution of the publication is an $\mathcal{O}(n^4 \log n)$ Las Vegas algorithm for colored graphs with bounded class size that tests if there is a color-preserving mapping between two given graphs. Furthermore, the author utilizes his procedure to obtain better theoretical upper bounds for GI for two special graph classes, one of them are graphs with bounded valence. The author also highlights the shortcomings of his method: There are heuristics that should be considered first, as $\mathcal{O}(n^4 \log n)$ is indeed polynomial, but too slow in practice, and the algorithm does not provide a canonical labeling of the nodes.

Babai and Luks (1983) present an algebraic approach (i.e. based on properties of permutation groups) for the canonical labeling of general graphs—and therefore for GI—in time $\exp(n^{\frac{1}{2}+\mathcal{O}(1)})$. This complexity result was the best standing upper bound for over 30 years. The authors furthermore prove (new) upper bounds for, e.g., graphs with bounded valence and tournament graphs. The latter are complete graphs, where each of the $\binom{n}{2}$ edges has an explicit direction. Pseudocode for a recursive algorithm that produces a "canonical placement" of a string of length n given some alphabet of symbols is given by the authors. Canonical placement is just another term for canonical labeling. The idea of the algorithm is to find a lexicographically minimal string that stabilizes subsets of the node set for some group. For the symmetric group, simply the lexicographically smallest string would be the correct placement. The first string that acts as input for the algorithm is the one that can be constructed from the degree partition. It is important to mention that stabilizing a (sub-)set of nodes is of course related to finding automorphisms of the graph. Besides the theoretical relevance of the article, there does not seem to exist an actual implementation of the method.

Only recently, Babai (2016a,b) came up with a new proof that is claimed to reduce the worst-case time complexity of GI to $\exp((\log n)^{\mathcal{O}(1)})$, which is called quasipolynomial complexity. The ideas of Luks (1982) and Babai and Luks (1983) are picked up and used to provide pseudocode for a conceptual algorithm that realizes the new upper bound. This especially involves the subroutines of the algorithm to solve the string isomorphism problem rather than GI directly. Babai (2016b, p. 684) states that his method builds on the divide-and-conquer algorithm by Luks (1982) and additionally creates "local certificates" to test for local symmetry. Moreover, Babai (2016b, section 5.5) argues that his results are not particularly helpful in practice but give further evidence that GI is not $\mathcal{NP}$-complete (Babai, 2016b, section 5.2).

3.4.3 Practical Algorithms

In this section we present "practical" algorithms to solve GI and/or GA, i.e. algorithms for which actual implementations exist and that are efficient. Of course, the theoretical upper bounds of the problems also hold for these algorithms, however, their expected performance on problems is in practice much better. The following five algorithms are described:

nauty (McKay, 1981; McKay and Piperno, 2014)

saucy (Darga et al., 2004, 2008; Katebi et al., 2012)

bliss (Junttila and Kaski, 2007)

Traces (Piperno, 2008; McKay and Piperno, 2014)

conauto (López-Presa and Anta, 2009; López-Presa et al., 2014)

The first four methods have many concepts in common, as they all are backtracking algorithms, which build and traverse a tree data structure to create a set of generators for the automorphism group of the graph. `nauty`, `bliss`, and `Traces` are also able to produce a canonical labeling of the input graph, which allows to test isomorphism. The last, `conauto`, only tests for isomorphism by solving the decision problem. Especially `nauty` and `saucy` have undergone some improvements over the years; implementations of `nauty` and `Traces` are available as one package and are described by a joint article of McKay and Piperno (2014).

Of course, there are other algorithms that try to solve GI and/or GA (e.g. Tener, 2009; Stoichev, 2010), but they have not gained as much attention as the five algorithms selected. Therefore, we will not describe them further.

Before describing the different algorithms separately, we give an idea of the generic procedure of the tree traversal. The explanation is loosely based on the article of McKay and Piperno (2014, section 2), but much more informal, as we believe it is most important to grasp the general idea of these algorithms in this thesis' setting. Each tree-node represents an ordered equitable partition of the node set $V(G)$ and each cell of a partition has a different color. An ordered partition has the usual properties of a partition as defined in Section 2.3, but the cells obey some ordering (see also Hartke and Radcliffe, 2009). Ordering can be achieved by replacing the partition definition as set of sets by the definition as ordered set of sets. An ordered set is represented by a sequence of elements. This means $\{\ldots, C, D, \ldots\}$ (if C or D is noted first is arbitrary) is replaced by $(\ldots, C, D, \ldots)$ (C noted before D is determined by the chosen ordering). Therefore, also equality of two ordered partitions depends on the ordering. For example, $\{\{1\},\{2\}\} = \{\{2\},\{1\}\}$ but $(\{1\},\{2\}) \neq (\{2\},\{1\})$. Additionally, the explicit ordering allows to determine the position of a cell so that, e.g., C_i is at position i, and if $i < j$, C_j comes after C_i.

The partition that represents the root node is created by either making the degree partition or some other initially colored partition equitable. To "make equitable" simply means to refine a given partition by dividing its cells until the resulting partition is the coarsest equitable one. The definitions for *coarser* and *finer* are the ones described in Section 2.3, but they also must obey the ordering, i.e. split cells of the finer partition must keep their position relative to their ordering in the coarser partition. For example, $(\{1,3\},\{2,4\}) \geq (\{1,3\},\{4\},\{2\})$ but $(\{1,3\},\{2,4\}) \ngeq (\{2\},\{1,3\},\{4\})$ as $\{2\}$ suddenly appears before $\{1,3\}$, which destroys the ordering of the coarser partition (Hartke and Radcliffe, 2009, p. 5).

By definition, the partition represented by the root node is invariant under the automorphism group, which means that all nodes affected by $Aut(G)$ (yet unknown) are only moved within the cells. A singleton partition in this context is called *discrete* and induces (due to the ordering) a permutation of the node set $V(G)$. If the root node is already discrete, $|Aut(G)| = 1$, as each node is fixed by the automorphism group. This is because it is the coarsest equitable refinement of the initial partition. Otherwise, the full tree eventually consists of leaf nodes, which all represent discrete ordered partitions, and each of the partitions induces a permutation (see Figure 3.7 in the following Example 27). The question now is, what do these permutations express and what are the inner nodes of the tree?

We begin by explaining the second part of the question. Each node of the tree represents an equitable partition and, more specifically, each child node represents a coarsest equitable refinement of the partition of its parent node. To derive the children for the root node or some inner node, an arbitrary cell of the represented equitable partition is selected by a so called *target cell selector*. For each node in the selected cell, a new partition is created by splitting off the node from the cell and refining the resulting partition to make it equitable again. This procedure is called individualization (of the specific node v). Recall the interpretation of the partition represented by the root node: It is invariant under $Aut(G)$, which means only nodes within those cells may be affected by automorphisms, if they exist. Therefore, the tree is recursively built by selecting a target cell of the partition representing the current node, individualizing all nodes of it, and refining the result. This is repeated until discrete partitions are reached, those represent the leaf nodes of the tree.

Let us now come to the first part of the above question (What do the permutations express?). Due to the ordering of the partitions, the coarsest refinements differ for different individualized nodes. As a consequence, all leaf nodes represent different permutations with respect to the initial partition. The equitability property assures that only those permutations are created that have a possibility to be an automorphism of the graph. Let L denote the set of partitions represented by leaf nodes. As a result, these permutations generate some group $\langle L \rangle = H$ and $Aut(G) \leq H \leq S_n$ holds. Therefore, the goal is to keep only those permutations so that $Aut(G) = \langle L \rangle$ and, possibly, that L is a minimal generating set.

The individualization of a node u and the following partition refinement corresponds to a fixation of u. Fixation means that all following inner nodes (the subtree rooted at the corresponding tree-node) represent partitions in which u is assumed not to be affected by $Aut(G)$. As a consequence, the permutations represented by the leaf nodes of this subtree, which actually *are* automorphisms of G, generate the stabilizer subgroup $Aut(G)_u \leq Aut(G)$ (see Equation 2.74). This means the repeated individualization of (necessarily different) nodes successively reduces the possible automorphisms, as more and more nodes are fixed.

Example 27. *Let B be the "butterfly" graph in Figure 3.6. The degree partition is $P_{\text{deg}} = \{\{1,2,4,5\},\{3\}\}$ and it is equitable. The ordered representation of P_{deg} is $(\{1,2,4,5\},\{3\})$, which we abbreviate by $(1,2,4,5|3)$. The actual ordering is arbitrary, as long as it is well defined and used throughout the whole procedure. Here, lexicographical ordering is chosen.*

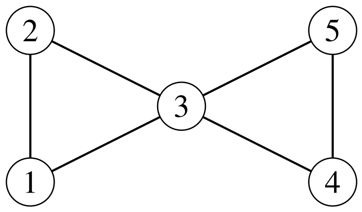


Figure 3.6: The butterfly graph B, which is clearly symmetric. $|Aut(B)| = 8$ and the globally minimum number of generators needed to generate the group is two, as there are two "separate" symmetric mappings: E.g. switch 1 and 2 or "mirror" $\{1,2\}$ and $\{4,5\}$ along the (thought) vertical axis through node 3. The permutation group obtained at the end of Example 18 in Section 2.4.3 by computing the wreath product of two symmetric groups S_2 is isomorphic to $Aut(B)$.

Figure 3.7 shows the complete search tree concerning B. The root node represents the initial partition, which is the coarsest refinement of the degree partition (it is just coincidence that they are equal). Without any knowledge about $Aut(B)$, the search is already reduced to $Aut(B) \leq (S^{V(B)})_3$, as node 3 must be fixed by $Aut(B)$. The individualization of, say, node 1 requires also to separate node 2 from nodes 4 and 5 because $(1|2,4,5|3)$ is not equitable (not every node of cell $\{2,4,5\}$ has an edge to cell $\{1\}$). None of the discrete partitions are direct automorphisms of B, because the ordering relation of the partitions is a permutation itself. Therefore, given two permutations induced by leaf nodes τ_i and τ_j, one must calculate the catenation $\tau_i^{-1} \circ \tau_j$. One partition, e.g. the one induced by τ_i, is the reference to which all other τ_j must refer. For example, the two permutation under the first subtree $(1|2|4|5|3)$ and $(1|2|5|4|3)$ represent

$$\tau_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 5 & 3 \end{pmatrix} \tag{3.29}$$

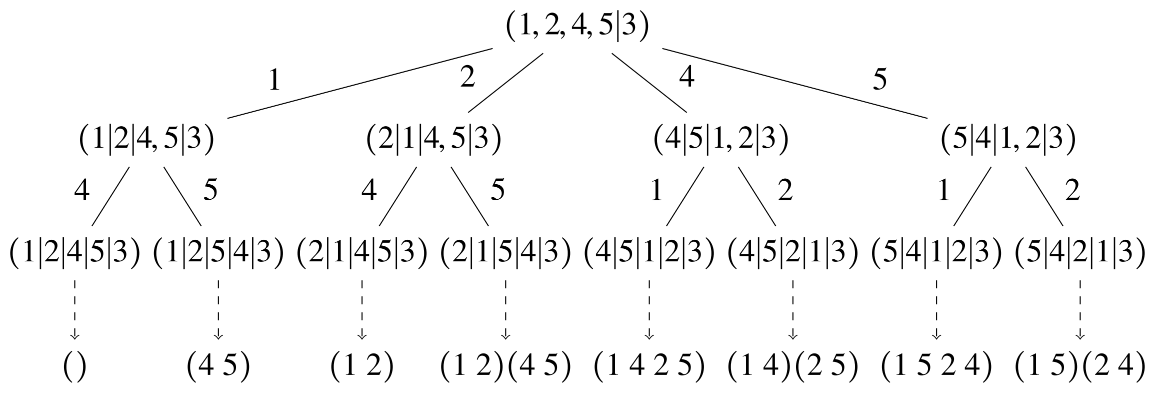


Figure 3.7: The complete search tree for B. The edge labels determine which node of the target cell is individualized. As there is exactly one non-singleton cell for the root node and each inner node, this cell is selected. Below the leaf nodes of the search tree, the permutations of $V(B)$, which are derived from the discrete partitions represented by the leaf nodes, are shown. The dashed arrows pointing towards them and the permutations themselves are not part of the search tree and are only shown for didactic reasons.

and

$$\tau_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 5 & 4 & 3 \end{pmatrix} \tag{3.30}$$

so that

$$\tau_1^{-1} \circ \tau_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 5 & 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 5 & 4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 5 & 4 \end{pmatrix} = (4\ 5). \tag{3.31}$$

Recall that fixing node 1 *means that the subtree rooted at* $(1|2|4,5|3)$ *represents the stabilizer subgroup that fixes* 1 *(and of course also* 3*). With the above result and by looking at B, one can assert that* $(4\ 5)$ *is the only permutation that is an automorphism of B if* 1 *is fixed.*

When looking at the induced permutations below the leaf nodes in Figure 3.7 (the leftmost partition is used as reference, which corresponds to **1***) it can be seen that all these permutations are actually automorphisms of B. This is, however, coincidence and can not be generalized.*

Until now, we have only described how the search tree looks like and that it—by construction—produces permutations of a subgroup of S_n; $Aut(G)$ is itself a subgroup of this permutation group. To find the actual automorphism group, one has to test each found permutation if it is an automorphism of the graph. This is already better than brute-force search, but the complexity is still exponential in the number of children of the tree in each level. If the initial partition, represented by the root node, is the trivial partition, individualizing would result in exactly n children on the first level. Each of those nodes could have a maximal number of $n-1$ children, and so on. Therefore, the tree could have a maximum of $n \cdot (n-1) \cdots = n!$ leaf nodes, which corresponds to all permutations of S_n.

To circumvent this issue, the tree traversal is performed in a more clever way. When the first discrete partition is derived (i.e. the first path from the root to a leaf node is explored), the tree traversal continues from the parent tree-node of the discrete partition with the next node to individualize. This results in a second discrete partition and, by using the procedure described in Example 27, a permutation that is a possible automorphism is derived.

If the permutation *is* an automorphism of G (this must be checked), then it is saved as part of the generating set of the group. Additionally, the orbit partition is gradually built by the revealed information, as all nodes that lie on the same cycle of a given permutation must lie on the same orbit. Therefore, at each level in the tree traversal it is sufficient to individualize only one of the nodes that are known to lie on the same orbit. If, on the other hand, the derived permutation is not an automorphism of the graph, the procedure continues.

Example 28. *We continue Example 27. Figure 3.8 shows an example of how the tree actually could be built and traversed for the butterfly graph B:*

(a) The first node of the target cell of the root node is individualized with subsequent refinement.

(b) Individualization (node 4) and refinement of the finer partition; the first discrete partition τ_1 is found.

(c) Another discrete partition τ_2 is found after individualizing node 5. The permutation $\tau_1^{-1}\tau_2 = (4\ 5)$ is an automorphism of B and it is, therefore, added to the generating set. Also, nodes 4 and 5 lie on the same orbit.

(d) As no other individualizations are possible, the procedure continues at the root node. Node 2 is individualized.

(e) Again, a discrete partition τ_3 is derived, $\tau_1^{-1}\tau_3 = (1\ 2)$ is an automorphism of B. The orbit partition is updated.

(f) Node 5 is not individualized, as nodes 4 and 5 are already known to be equivalent. The subtree is pruned, i.e. not generated and traversed further.

(g) Individualization again continues from the root node.

(h) τ_5 is found and $\tau_1^{-1}\tau_5 = (1\ 4)(2\ 5)$ is an automorphism of B. With this additional information, we know that all four nodes must lie on the same orbit.

(i) As nodes 1 and 2 are equivalent, the subtree is pruned.

(j) The last node of the target cell is also not individualized, as we already know that all nodes in the target cell lie on the same orbit.

By using the orbit information, which is successively exploited by building and traversing the tree, the effort can be decreased: Only 7 out of 12 individualization/refinement steps are needed, and only 4 out of 8 discrete partitions are created. However, the generating set $L = \{(4\ 5), (1\ 2), (1\ 4)(2\ 5)\}$ is not minimal, as one out of the two transpositions would suffice to generate $Aut(B)$. It is possible to compute $|Aut(B)|$ by the definition of the search tree utilizing

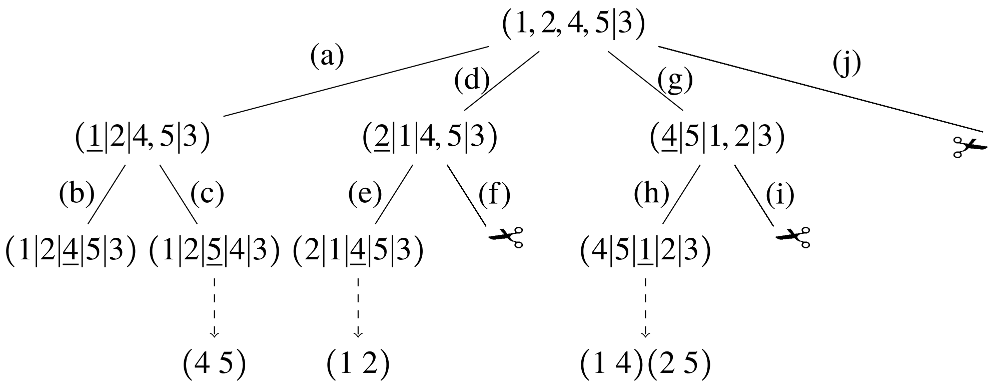


Figure 3.8: A possible search tree traversal where subtrees are pruned (emphasized by the scissors that cut off the subtrees) by orbit information. The steps (a) to (j) are described in the text; underlined nodes are the ones that were individualized.

Theorem 2 (orbit stabilizer theorem). The leftmost subtree yields all generators that fix node 1*, so the subgroup* $Aut(B)_1$ *contains two permutations (identity and the transposition* $(4\ 5)$*). Also, the orbit* $\{1, 2, 4, 5\}$ *of node* 1 *is known, therefore* $|Aut(B)| = |Aut(B)_1| \cdot |1^{Aut(B)}| = 2 \cdot 4 = 8$.

A last "ingredient" of these types of search algorithms are *node invariants*, which are used to compare tree-nodes. For details, see McKay and Piperno (2014, sections 2.4/3.3). A node invariant is a function that depends on the graph and initial partition and that returns some value for a given tree-node. If the values for two different tree-nodes are unequal, there must not exist an automorphism that maps the partitions represented by those nodes onto each other (McKay and Piperno, 2014, p. 98)). The purpose of such invariants is to further reduce computational complexity by comparing tree-nodes (not only leaves!) to possibly allow additional pruning.

We have omitted certain details, like the target cell selection and the node invariants and we have not proved the correctness of the procedure. For these details (which are out of scope of this thesis) we refer to the articles we are going to cite in the next paragraphs and the references that can be found therein. How to compute a canonical form of the graphs has also not been discussed yet.

3.4.3.1 nauty

The first popular algorithm that employs the described search strategy is `nauty` (McKay, 1981). McKay thoroughly describes the procedure with all its subroutines (e.g. refinement, tree generation) and proves the correctness of the method. The results go back to the author's master and PhD theses from 1976 and 1980 (McKay, 1981, p. 45, references 12 and 14). In section 3 of this article, McKay also describes the used data structures. For graphs, an adjacency matrix representation is chosen, which is quite expensive in terms of storage ($\mathcal{O}(n^2)$ space). The author also discusses the time and storage complexity. Refinement of a partition is in $\mathcal{O}(n^2 \log n)$ and the complexity of the main routine is (at the first sight) polynomial in n, but it also includes several other variables like the number of tree nodes that follows a certain condition for which no upper bound is known (McKay, 1981, section 3.7). The results of Miyazaki (1997) show that examples can be found that force nauty to have exponential time complexity. The first version of nauty was implemented in Fortran and parts even in direct machine language (assembler) (McKay, 1981, section 3.8). Pruning of the search tree is mainly done by using the found automorphisms, which are used to update the orbit partition (as we have described above).

To obtain a canonical labeling, McKay (1981, p. 57) generally defines an *indicator function* (another name for a node invariant) whose value is invariant under *any* permutation for a given graph and partition. The value of such a function is an ordered set, the actual ordering is arbitrary. In the end, the canonical labeling is the labeling implied by a partition that represents a permutation (a discrete partition) for which the value of the indicator function is maximal. The crucial part is that the indicator function employs a relabeling that is independent of the actual labeling of the graph. As two isomorphic graphs have the same automorphism group, the traversal of the search tree is identical (although the node labels differ) and, therefore, the values of the indicator function (which must be of course fixed) are the same. As a consequence, the maximal values must be the same iff the graphs are isomorphic.

The latest version of nauty[2] (McKay and Piperno, 2014) is written in the C language and contains several improvements as, e.g., the use of a randomized method (Seress, 2003, randomized Schreier method, pp. 70 ff.) to compute the stabilizer subgroup at a certain point in the search, which sometimes allows additional pruning. It is also possible for a user to provide custom node invariants that are applicable for the input graph to speed up the computations.

3.4.3.2 saucy

Over 20 years nauty was the only algorithm that utilized the backtracking strategy, until Darga et al. (2004) introduced saucy, whose background is purely application-driven: The automated design of electric circuits can be transformed into a Boolean satisfiability (SAT) problem, which needs to be solved efficiently. These satisfiability problems can be transformed into a conjunctive normal form (CNF), so that the derived problem only consists of several clauses that must all be true. Each clause C_i contains a number of Boolean variables v_j (literals; possibly negated) that are connected by OR operators; the clauses are combined by AND operators:

$$\bigwedge_i \bigvee_{v_j \in C_i} v_j. \tag{3.32}$$

An example is given by Darga et al. (2004). They also show how an undirected bipartite graph (literals and clauses are colored differently) can be derived from a CNF formula. The bipartite graph construction is an example of a different initial partition for the search tree generation. Two SAT problems are equivalent if their induced graphs are isomorphic. The example of the authors is

$$\underbrace{(\neg a \vee b \vee c)}_{C_1} \wedge \underbrace{(a \vee \neg b \vee \neg c)}_{C_2} \wedge \underbrace{(\neg b \vee c)}_{C_3} \tag{3.33}$$

[2] Version 2.6r10 from October 2017, see http://pallini.di.uniroma1.it/ as of March 2018

and the permutation $(a\ \neg a)(b\ \neg c)(\neg b\ c)(C_1\ C_2)$ is an automorphism of the derived graph, i.e. clauses C_1 and C_2 are equivalent after some relabeling. As a consequence of the equivalence, the solution space of the SAT problem is reduced from 2^3 to 2^2 different literal assignments. Darga et al. (2004, section 4) argue that graphs induced by CNF formulas are usually very sparse and have a small average node degree, which leads to a major improvement compared to `nauty`: Only connected cells (in terms of a partition represented by a node in the search tree) are selected for the individualization-refinement procedure. In the authors' evaluation, `saucy` has a significant speed advantage over `nauty` for the used class of benchmark graphs. To underline the role of sparsity, the authors use the respective complement graphs (which are very dense but have the same automorphism group) and show that `nauty`'s performance is unaffected by this transformation. As expected, `saucy`'s advantage for sparse graphs turns into a disadvantage for these dense graphs. Note, however, that `saucy` can not create a canonical labeling of the graph.

Like `nauty`, also `saucy` was improved several times[3]:

- (Darga et al., 2008): Instead of detecting symmetry only in leaf nodes of the search tree, `saucy` was improved to "short-circuit" paths from inner nodes to leaf nodes if certain conditions hold. Again, these conditions hold very often for sparse graphs and decrease the number of tree-nodes from quadratic to linear in terms of the number of generators. Another change is to eliminate the strict target cell selection that always selects the same cell per level in the tree. Instead, an arbitrary non-singleton cell is selected.

- (Katebi et al., 2012): The authors extend the definition of the search tree. Each node represents an ordered partition pair (OPP), which consists of the so called top and bottom partitions. These two partitions are refined simultaneously and correspond to the partitions that result from the individualization of the first and last node of a given target cell. Both partitions must always be isomorphic and if a conflict is found (i.e. the partitions are *not* isomorphic), the subtree can be pruned. The isomorphism test is performed after each step of the refinement procedure, as this invariant must hold at all time. The authors call this isomorphism invariant for non-discrete partitions the representation of "partial permutations" (Katebi et al., 2012, p. 256).

The improvements from `saucy` 2 (Darga et al., 2008) to `saucy` 3 (Katebi et al., 2012) are, however, only relevant for the special class of Miyazaki graphs (Miyazaki, 1997), which was created as a benchmark for `nauty`. Therefore, we believe that especially the improvements to reduce the number of nodes of the search tree are another main competitive advantage for sparse graphs. This advantage on sparse graphs can be carried over to the analysis of large real-world graphs. Therefore, `saucy` is used for the analyses presented in Chapters 4 and 7.

[3] Latest version 3 from "Spring 2012", see `http://vlsicad.eecs.umich.edu/BK/SAUCY/` as of March 2018

3.4.3.3 bliss

Junttila and Kaski (2007) present `bliss`, which is—as `saucy`—a descendant of `nauty`. The main advantage over `nauty` and `saucy` (note the date of publication, only the first version of `saucy` was available) is the introduction of "incremental leaf certificates", which are basically node invariants that are not only computed for leaves, but already partially for inner nodes of the tree. As soon as two partial certificates differ, the complete subtree can be pruned as—by the invariance definition—no automorphisms of the graph can be found. Furthermore, the authors emphasize their more efficient data structures that are used, which, e.g., allow a less expensive target cell selection method. Yet another difference is that `bliss` keeps track of two orbit partitions what sometimes allows additional pruning during the search. The authors compare the algorithm with `nauty` and `saucy` (the available versions at that time) and find that `bliss` outperforms both in many cases. Like `nauty`, `bliss` is also able to extract a canonical labeling of the input graph.

In a later publication, Junttila and Kaski (2011) further improve their idea of using sequences of node invariants (i.e. the partial leaf certificates) by additionally saving hash values of the certificates of nodes that are not isomorphic to the first path that was traversed (i.e. the first path from the root node to a leaf node). This allows, again, additional pruning of subtrees that cannot result in automorphisms of the graph. A second improvement consists of saving another colored partition for every node of the search tree, which makes it possible to prune subtrees if two inner nodes are found to be isomorphic (a similar procedure that was introduced for an improved version of `saucy`).

3.4.3.4 Traces

Besides small differences and improvements in detail, all three algorithms above follow the same basic search procedure and act on the same abstract data structure. We have not yet explicitly noted that the search strategy follows the *depth-first search* (DFS) pattern, i.e. the tree traversal goes from the root to the leaves by exploring complete partition refinement paths one by one.

Piperno (2008) presents a quite different approach to tackle the canonical coloring problem. Before going into algorithmic details, the author points out a major problem that results from the DFS strategy: The actual runtime complexity for finding a canonical labeling depends on the labeling of a graph, i.e. it may take longer for one graph to find the canonical labeling than for another graph, although they are isomorphic. To deal with this issue, `Traces`' traversal strategy is breadth-first search. So instead of exploring a partition refinement path until a leaf node is reached, each level of the search tree is created first. However, to retain the ability to prune subtrees by already found automorphisms, one single refinement path (the "experimental path") is traversed for each tree node per level. By definition of the tree, all discrete partitions represented by those leaf nodes can be used to find automorphisms of the graph. The overall strategy tries to incorporate early pruning at each tree level and pruning by already found automorphisms. As a further improvement of the latter procedure, `Traces` utilizes the Schreier-

Sims algorithm (e.g. Seress, 2003), which computes a strong generating set from a given set of group generators, that then eases several helpful computations (e.g. “computing the orbits of point stabilizers” Piperno, 2008, p. 11). Other implemented improvements are:

- The use of a 2-dimensional partition refinement (the 1-dimensional one is the “standard” refinement, which, e.g., `nauty` implements) that—despite being more costly to compute—may decrease the depth of the tree. The author argues that this invariant is applicable for all classes of graphs, which, contrary to `nauty`, removes the need of the user’s background knowledge to choose specialized invariants. This invariant, which is referred to as the *2-dimensional Weisfeiler-Lehman refinement* (Weisfeiler, 1976), considers not only how the neighbors of cells are colored, but also employs a coloring to the edges (treated as directed). The edges are colored by the number of triangles in which they participate in (Piperno, 2008, p. 12). As a result, this 2-dimensional refinement is finer than the 1-dimensional one.

- The comparison of inner tree nodes by partial leaf certificates like the ones `bliss` implements. However, each certificate is a vector of vectors called the “trace”. This is where the algorithm’s name originates from (McKay and Piperno, 2014, p. 102).

The authors of `nauty` and `Traces` have joined forces (McKay and Piperno, 2014) and provide explanations of the latest versions[4] of both algorithms. Moreover, performance tests are shown and the authors conclude that `nauty` is the preferable choice for small graphs, `Traces` is the leading option for “difficult” classes of graphs (McKay and Piperno, 2014, section 5).

3.4.3.5 conauto

A quite different method is proposed by López-Presa and Anta (2009), who published an algorithm, which they call “direct”, as it neither computes a canonical labeling nor does it find generators for the automorphism group. Instead, it computes a mapping between two given graphs (if it exists). At first, the authors argue that direct methods normally have a problem with symmetric graphs, as they are not able to efficiently compute the automorphism group of either of the two possibly isomorphic graphs. They claim to overcome this problem by partially taking existing automorphisms into account, but only up to the point an isomorphic mapping is found or can be ruled out (i.e. invariants, which must always be true for both graphs, fail). For this purpose `conauto` stores and updates a “semiorbit” partition, which contains equivalence information of nodes due to found automorphisms (but without keeping the actual permutation). The name “semiorbit” comes from the fact that not necessarily the complete orbit of the automorphism group is found during the procedure. The crucial part of the algorithm is performed recursively and involves a backtracking strategy, just as all the other methods we have described above. The general functionality of `conauto` can probably be best thought of as the traversal of a path of the search tree of one of the graphs and then to find a matching path for the other graph.

[4] `Traces` version 2.1, see `http://pallini.di.uniroma1.it/` as of March 2018

All the details can be found in the PhD thesis of the first author (López-Presa, 2009). The gentle reader will also find the origin of the algorithm's name, which combines the two words `con` (Spanish "with") and `auto` (short for automorphism(s)) so `conauto` simply means "with automorphism(s)".

With the second version of `conauto` (López-Presa et al., 2014), the algorithm is also capable to compute a generating set of the automorphism group. Their approach involves several techniques that make the implementation quite efficient compared to others:

EAD The *early automorphism detection* allows to uncover automorphisms in inner tree-nodes, not only in leaves.

BJ *Backjumping* is an improved backtracking method (if a conflict is found), which makes it possible to directly jump back to a certain ancestor that is on a higher level in the search tree instead of backtracking to the parent node only.

DCS A novel *dynamic cell selector* is presented, which is necessary for the two preceding techniques. The authors argue that the target cell selector is—due to its dynamism—very good for diverse classes of graphs. The idea of DCS is that it favors those partitions as refinement results that consist of many more cells than the input partition. This means that DCS selects target cells in a way that the discriminating power is maximized and, as a consequence, the search tree depth is reduced.

CDR Similar to the pruning procedure that was published with the latest version of `bliss`, *conflict detection and recording* stores a hash value for tree-nodes at which a conflict occurs. However, according to the authors' observations—their method is superior to the one of `bliss`.

López-Presa et al. (2014, pp. 8 ff.) evaluate the described techniques by testing against the latest versions (at that time) of the four algorithms discussed above and also the older version of `conauto` itself. They find a significant decrease in the number of explored tree-nodes for many special classes of graphs compared to the competitors. The authors conclude with the remark that their four improvements are general and can be integrated in any algorithm that follows the node individualization and refinement approach.

3.4.4 Different Approaches and Algorithms for Related Problems

We only briefly want to review some related problems to GI/GA. A survey is given by Conte et al. (2004).

One of those problems is subgraph isomorphism (SGI), which we already mentioned when theoretical complexity was discussed in Section 3.1. An often cited approach is presented by Ullmann (1976) and a more recent approach is by Cordella et al. (2004). The latter algorithm is a direct method, as described above, and capable to decide GI as well as SGI by returning an actual mapping.

Sorlin and Solnon (2004) introduce a transformation of GI to a constraint satisfaction problem, which also can be extended to be used for directed graphs or to solve SGI.

A method other than the backtracking algorithms described above is given by Mansour (2017) and it is based on message passing. First, the possibly isomorphic graphs are converted to a *canonical form* and then the message passing algorithm is applied on both graphs resulting in an output signature that must be equal if the graphs are isomorphic. However, a comparison with `nauty` yields worse results of the described algorithm. An inexact method to solve GI based on neural networks is proposed by Jain and Wysotzki (2005).

The article by Rudinger et al. (2013) could be used as an entry-point to approaches that tackle GI by using quantum computing. The authors do not provide a new algorithm but compare previously published ones.

Foggia et al. (2001) compare the performance of five different GI solvers, namely, `nauty` (McKay, 1981; McKay and Piperno, 2014), the one of Ullmann (1976), VF (Cordella et al., 1999), VF2 (Cordella et al., 2001), and a method presented by Schmidt and Druffel (1976).

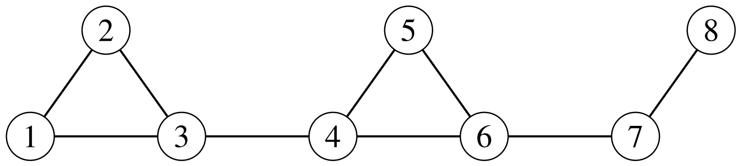
3.5 Other Graph Symmetry Concepts

So far, only exact mappings of nodes onto nodes were discussed. In this section we review some related and sometimes "weaker" concepts. Weaker is meant in the sense that exact symmetry implies the weaker form of symmetry but not the other way round.

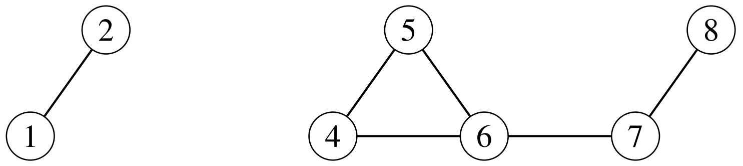
Pseudo-similarity Two nodes u, v of a graph are equivalent/similar if they lie on the same orbit, i.e. there exists an automorphic mapping between those nodes (see Sections 2.4 and 3.2). Let $G_u = (V(G) \setminus u, E_u)$ and $G_v = (V(G) \setminus v, E_v)$ be the induced subgraphs that result from the removal of the nodes u and v ($u \neq v$). As u and v lie on the same orbit, these two induced subgraphs must be isomorphic, i.e. $G_u \cong G_v$ holds.

However, there may exist nodes u', v' ($u' \neq v'$) that do not lie on the same orbit, but their induced subgraphs $G_{u'}$ and $G_{v'}$ are nonetheless isomorphic. Therefore, u' and v' are called *pseudo-similar* if $G_{u'} \cong G_{v'}$ holds (Harary and Palmer, 1966; Kimble et al., 1981). An example is shown in Figure 3.9.

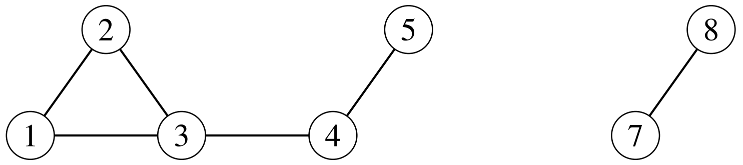
Hidden Symmetry Liu (1997) presents a method to decompose a graph into a disconnected graph that is isospectral, i.e. it has the same spectrum (eigenvalues) of the adjacency matrix $\boldsymbol{A}$. The resulting disconnected subgraphs normally have a higher degree of symmetry in terms of exact automorphisms than the "original" graph, which the author calls "hidden symmetry". The method allows the construction of a group that is a subgroup of the Hamiltonian group of a graph and that contains all matrices $\boldsymbol{T}$ for which $\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T} = \boldsymbol{A}$ holds. The automorphism group is a subgroup of this group, where the $\boldsymbol{T}$ are permutation matrices as defined in Section 3.1.



(a) A graph G, which is nearly asymmetric ($(1\ 2)$ is a symmetry).



(b) The induced subgraph G_3 that results from removing node 3.



(c) The induced subgraph G_6 that results from removing node 6.

Figure 3.9: Example from Kimble et al. (1981, figure 1). G is nearly asymmetric but, additionally, the nodes 3 and 6 are pseudo-similar, as $G_3 \cong G_6$.

Near Automorphisms In the article of Emerson and Trefler (1999) a relaxation of exact automorphisms to "near automorphisms" is presented. They apply their idea on an example of two computing processes—one is a reader, the other a writer—which depend on each other if they want to access the same shared resource. The individual states of both processes are the same ("Non-Trying" (N), "Trying" (T), "Critical Section" (C)) and the state flow is $\curvearrowright N \rightarrow T \rightarrow C \curvearrowright$. This means a process that is in state N (it is not trying to access the shared resource) can transition into state T (it is now trying to access the resource) and, eventually, can transition into the state C (the critical phase in which the resource is actually accessed). After accessing the resource, the process state changes back to N. Their dependency is described as a directed state transition graph (Emerson and Trefler, 1999, figure 1), which contains every possible pair of states as nodes and connects those combinations that are possible transitions. For example, the combined state is (N, N) (neither process wants to access the resource) and the possible transitions are into (T, N) or (N, T) (one of the two processes tries to access the resource). The state transition graph is asymmetric because of one edge breaking the symmetry, due to the fact that one process is a writer and has a higher priority than the reader for accessing data: The only possible state transition from (T, T) is into (C, T) if the first process is the writer. A transition into (T, C) is not possible. Because of this fact, the state transition graph is asymmetric, although both processes follow the exact same state transition flow. However, the presented relaxed definition of symmetry allows to exchange both processes and one can see the similarity of them.

Stochastic Symmetry Garlaschelli et al. (2010) state that exact symmetry—as captured by the automorphism group—is very unlikely to exist in complex networks. Therefore, the authors define "stochastic symmetry" of a graph as the assignment of this graph to a graph ensemble with a high probability. An ensemble is simply a set of graphs that all share similar properties and—considering a constructive model—have a certain probability to be generated by this model. These probabilities sum up to one. They nicely motivate this idea with a simple circle, which is perfectly symmetric in terms of rotations around its center. However, in reality it is very unlikely that perfect circles actually occur, rather than objects that are approximately circles but with small disturbances. Nonetheless, different objects that are all approximate circles with the same radius form an ensemble. Furthermore, the authors also review "scale invariance" as presented in Section 2.2.1.

In a second article, Ruzzenenti et al. (2010) apply the stochastic symmetry definition to the world trade web, which is described as a directed network showing trade relations between countries (the weights expressing actual monetary values are omitted). In fact, the authors analyze an ensemble of such graphs that represent the global trade networks in the years between 1948 and 2000. They especially emphasize the role of the degree sequence and state that countries with the same degree (i.e. the same amount of trading relationships) are equivalent. Furthermore, the authors find a strong relation between the GDP (Gross Domestic Product) and trading activity (Ruzzenenti et al., 2010, p. 1736) but no dependence of spatial embeddings of the network (i.e., for example, direct neighborship of countries or absolute distances given by the road network).

Just recently, Schieber et al. (2017) introduced a dissimilarity measure for graphs that combines global and local quantities (node distances, centrality). Isomorphic graphs have dissimilarity 0 (they *are* similar), but the authors also state that they cannot guarantee that two non-isomorphic graphs exist that also have a measure of zero. A positive value of the measure, however, indicates non-isomorphism of two graphs. Schieber et al. (2017) evaluate their measure by comparing several random (using models as described in Section 2.2) and real-world networks.

4 Symmetry Analysis of Real-world Graphs

We have introduced symmetry of graphs in Section 1.2 and presented formal definitions in Section 3.2. So far, we have seen that literature on graph symmetry either refers to the formal mathematical description and analysis, but independent of the actual relevance in graph data analysis, or that a graph representation is used as an intermediate model to solve a symmetry problem of another application domain (e.g. logical satisfiability problems (Darga et al., 2004)). There is a lack of analyses and understanding if and when real-world graphs are symmetric. We presented the most well-known graph models (e.g. Erdős-Rényi and preferential attachment) in Section 2.2 and none of them directly involves symmetry considerations. Only models that are based on the copying of nodes and their edges indirectly involve graph symmetry, as we briefly discussed in Section 2.2.4. Furthermore, we found that the ER model (Erdős and Rényi, 1963) and the preferential attachment model (Magner et al., 2014) produce asymmetric graphs in the limit, but only a few publications investigate "arbitrary" graphs from several application domains for their symmetry properties:

- MacArthur et al. (2008): A *symmetric decomposition* (see Section 2.4.3) of the automorphism group of graphs is presented by the authors and they exemplarily apply this decomposition to 20 complex biological, technological, and social networks. They also identify several small symmetric network motifs that often occur in the diverse graphs.
- Darga et al. (2008): The authors analyze graphs for symmetry as part of a benchmark of their own method of computing a set of generators (a modified version of `saucy`).
- Xiao et al. (2008a): They define and compute "network quotients", which are simplified versions of a network that group structural equivalence (i.e. symmetry) in equivalence classes. They apply their definitions on 11 real-world networks.
- Wang et al. (2009): These authors explore symmetry in the world trade network and find many local symmetries. We define local symmetry in Section 5.2 in the same context as Wang et al. (2009) but more thoroughly.
- Wang et al. (2012): In their article, the authors present an algorithm for the "symmetry compression" of graphs that allows a faster discovery of network motifs (Milo et al., 2002). They use 12 symmetric protein-protein-interaction networks to benchmark their method.

To underline the importance of taking symmetry into account when analyzing network data, we analyzed a large number of real world networks. The results are published in (Ball and Geyer-Schulz, 2018a).

In total, over 1500 graph datasets have been obtained and analyzed from the meta-repository `networkrepository.com` and over 70 % of them are symmetric. To quantify symmetry, the measure "normalized network redundancy" is derived from results of MacArthur et al. (2008). It is notable that symmetries are found in all types of graphs, independent of their size and modularity.

4.1 Measures of Graph Symmetry

In this section some measures to quantify the symmetry of graphs are presented. They are either based on the order of the automorphism group or on the number of orbits of the group. The absolute size of the automorphism group $|Aut(G)|$ itself (which grows faster than exponential in n) is not a good measure, as the following example shows.

Example 29. *Let G be a graph that has $n = 100$ nodes and 15 of them can be mapped onto each other in any way, while all others are fixed. The group of this graph is isomorphic to the symmetric group S_{15} with a size of $|S_{15}| = 15! \approx 1.3077 \times 10^{12}$. Compare this to a much larger graph with, say, $n = 1000$ nodes and 15 of them form the cyclic group C_{15}, the size of this automorphism group is only $|C_{15}| = 15$. As a consequence, the group order is completely misleading because (i) the size of the graph is not taken into account and (ii) the group order cannot be interpreted intuitively.*

4.1.1 Measures Based on the Group Size

As was shown in Section 2.4, the maximal number of permutations on n elements (realized by the symmetric group) is $n!$. Therefore, one measure of (relative) symmetry, which is used, e.g., by Zenil et al. (2014), is

$$symm_1(G) := \frac{|Aut(G)|}{n!}. \tag{4.1}$$

It is the relation of actually existing permutations of the automorphism group of the graph compared to the maximum number of permutations of its n nodes. Both values tend to become very large, which makes the numeric computation of the measure difficult. The factorial of n can be approximated by Stirling's formula (e.g. Mortici, 2009)

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{4.2}$$

and the order of the group is normally given as result of an algorithm that computes generators for the group (like `nauty` or `saucy`). Besides numerical issues, even for very symmetric real-world graphs, the order of the group is much smaller than $n!$, which would result in very small values of the measure.

Another—quite similar but more specific—measure is given by Zhang et al. (2012, definition 3), which they call "normalized symmetry":

$$symm_2(G) := \frac{|Aut(G)|}{\prod_{k \in \{\deg(u) | u \in V\}} |\{u \in V \mid \deg(u) = k\}|!}. \tag{4.3}$$

Instead of relating the group order to the order of S_n, the measure only takes into account actually possible symmetries, which are constrained by the node degrees of the graph. Therefore, the inequality

$$symm_1(G) \leq symm_2(G) \tag{4.4}$$

holds and equality is obtained for regular graphs.

Based on $symm_2$, one can also think of a measure that is not only based on the degree partition but on the coarsest possible equitable partition P^e (see Section 3.3):

$$symm_3(G) := \frac{|Aut(G)|}{\prod_{C \in P^e} |C|!} \tag{4.5}$$

Again,

$$symm_2(G) \leq symm_3(G) \tag{4.6}$$

holds, as the coarsest equitable partition possibly divides the degree partition into finer cells.

4.1.2 Network Redundancy

MacArthur et al. (2008) take a detailed look at 20 real world networks from the biological, technical, and social domain. They decompose the automorphism group of each graph G into products of subgroups (see Section 2.4) and compute the index "network redundancy" (MacArthur et al., 2008, p. 3530). It is defined as

$$r_G = \frac{|O_G| - 1}{n}, \tag{4.7}$$

where O_G is the orbit partition of G and $n = |V(G)|$. As $|O_G| \in [1, n]$, $r_G \in \left[0, \frac{n-1}{n}\right]$ holds, and the lower the value, the higher the symmetry of G. Using the orbit partition as source of redundancy information is a good idea, as the orbits contain only the information *that* nodes are mapped onto each other by $Aut(G)$ but not *how*.

However, MacArthur et al. (2008) argue that nodes on non-trivial orbits *increase* the redundancy of a network in terms of robustness; therefore, a higher value should indicate a greater symmetry. We believe a better definition is

$$r'_G = 1 - \frac{|O_G| - 1}{n - 1} \quad \in [0, 1], \tag{4.8}$$

because correcting the denominator by -1 means having $r'_G = 0$ for asymmetric graphs instead of some value near zero. The only restriction that arises is that r'_G is not defined for the unique graph with $|V| = 1$, but this is no practical restriction. We call the measure in Equation 4.8 *normalized network redundancy*. This normalization allows better comparability between several graphs and there is no bias for small graphs (in terms of n).

Example 30. *Let G_a be an asymmetric graph with $|V(G_a)| = 10$ and G_s be a symmetric graph with $|V(G_s)| = 100$ and $|O_{G_s}| = 91$. Using the redundancy definition of MacArthur et al. (2008) yields*

$$r_{G_a} = \frac{10-1}{10} = \frac{9}{10} = \frac{90}{100} = \frac{91-1}{100} = r_{G_s}. \tag{4.9}$$

If the definition of r'_G is used instead,

$$r'_{G_a} = 1 - \frac{10-1}{10-1} = 0 \tag{4.10}$$

and

$$r_{G_s} = 1 - \frac{91-1}{100-1} = 1 - \frac{90}{99} \approx 0.091 \tag{4.11}$$

hold.

A flaw that both measures have is that they do not quantify how many nodes are actually affected by the automorphism group. This is because the quantity *number of orbits* does not carry any information on the distribution of nodes relating to the number of trivial and non-trivial orbits. For example, consider a graph with $r'_G = 0.1 = 1 - \frac{|O_G|-1}{n-1}$. Resolving for $|O_G|$ yields $|O_G| \approx 0.9n$ and the two extreme cases are:

- There is exactly one long non-trivial orbit, all others are trivial.
- There are many short non-trivial orbits (length of two), the rest are trivial.

For the first case, nearly 90 % of the nodes are on trivial orbits, thus not affected by the automorphism group. In the second case, there are only about 80 % nodes on trivial orbits, the remaining 20 % are spread over the many short orbits. This means that between 10 and 20 % of the nodes are actually affected. This observation is formalized in Appendix A.

4.2 Description of the Analysis Procedure

The complete analysis procedure is divided into the following steps:

1. Retrieval of network metadata (e.g. number of nodes/edges, file size, download link of the dataset)[1].
2. Downloading of the actual datasets, which are provided as compressed zip-archives. Datasets are pre-selected by their file size, as the largest graphs have several gigabytes.

[1] `http://networkrepository.com/networks.php` as of 10.02.2017

3. Performing the main analysis procedure by loading, selecting, and transforming the graph data and computing relevant properties, like modularity, automorphism group size, and number of orbits.

4. Calculating relevant statistics (e.g. the normalized network redundancy) using the network properties computed in step 3.

Each step is realized by an independent Python script. The complete details are given in Appendix B and the scripts are available online[2]. An overview and an interpretation of the results are given in the following sections.

4.2.1 Network Data

The network data was obtained from `networkrepository.com` (Rossi and Ahmed, 2015). The repository website claims to be "[t]he first interactive data and network repository with real-time analytics"[3]. Furthermore, they state to be "not only the first interactive repository, but also the largest network and graph data repository with over 500+ donations"[3]. As of February 2017, the website lists about 3900 network datasets, categorized by different types (see Table 4.1). Users can contribute datasets and have a glance at individual networks by several indices (such as number of nodes/edges, maximum/average node degree, average local/ global clustering coefficient), by visualization, by (cumulative) distributions of indices and by scatterplots between pairs of indices. Information on graph symmetry is not captured. The repository includes networks that can be found in many other repositories (e.g. the one of GEPHI[4] (Bastian et al., 2009), SNAP[5] (Leskovec and Krevl, 2014), or the DIMACS challenges[6] (Bader et al., 2014, 10th DIMACS challenge)). Using this kind of data repository allows to access a large variety of datasets at one site and without the need to combine multiple sources.

4.2.2 Downloading the Datasets

A total of 3015 datasets were downloaded, each with a (compressed) size of about 70 megabytes or less. The largest network in the sample in terms of nodes consists of $n = 11{,}950{,}757$ nodes and $m = 12{,}711{,}603$ edges. Looking at the edges, the largest graph has $n = 8{,}388{,}608$ nodes and $m = 25{,}165{,}784$ edges. Limiting the datasets' size was a matter of the algorithmic complexity of obtaining the automorphism group of a graph rather than a lack of disk-space. The complexity increases at least linearly with the size of the graph. All in all, datasets with an overall compressed size of 11.5 gigabytes were downloaded.

The graph data is encoded in either one of two plain-text data formats, namely:

- The Matrix Market exchange format (Boisvert et al., 1996) or

[2] `https://github.com/KIT-IISM-EM/networkrepository_analysis`
[3] `http://networkrepository.com/` as of February 2017
[4] `https://github.com/gephi/gephi/wiki/Datasets` as of February 2017
[5] `http://snap.stanford.edu/data/` as of February 2017
[6] `http://dimacs.rutgers.edu/Challenges/` as of February 2017

Type Name	Short Type Name	Count
BHOSLIB	`bhoslib`	36
Biological Networks	`bio`	9
Brain Networks	`bn`	36
Cheminformatics	`chem`	600
Collaboration Networks	`ca`	16
DIMACS	`dimacs`	78
DIMACS10	`dimacs10`	84
Dynamic Networks	`dynamic`	22
Ecology Networks	`eco`	6
Facebook Networks	`socfb`	114
Infrastructure Networks	`inf`	8
Interaction Networks	`ia`	19
Massive Network Data	`massive`	9
Miscellaneous Networks	`misc`	2652
Recommendation Networks	`rec`	13
Retweet Networks	`rt/retweet_graphs`	62
Scientific Computing	`sc`	11
Social Networks	`soc`	57
Technological Networks	`tech`	10
Temporal Reachability Networks	`tscc`	38
Web Graphs	`web`	27
	$\sum$	3907

Table 4.1: Number of datasets for different network types on `networkrepository.com`. The “Type Name” is directly taken from the navigational menu `REPOSITORY`[3]; the data grouped by “Short Type Name” (column “Type” on the website) can be found in the overview table of all network data[1]. For retweet networks, only the short type name `rt` will be used in the following.

- an edge list format (Kunegis, 2014, Chapter 9).

Both formats basically hold information for one edge per line and have additional meta-information in the first lines of the file.

4.2.3 Analysis Procedure

The procedure is described in more detail in Appendix B. Here, we will give only a short impression on what was done. The goal was to analyze a huge amount of network data for their symmetry properties. Table 4.1 lists the different network categories and their unique (besides “Retweet Networks”) short name. Most of the networks are uniquely assigned to one category, however, some are contained in more than one category.

An important issue that had to be decided about is how to deal with non-simple graphs, i.e. graphs that are weighted, disconnected, have loops or multiple edges, and/or are directed. A helping factor for this decision is the fact that `saucy`, which is utilized for the analysis, can not handle weights. The first idea was to simply exclude all non simple graphs from the analysis. Other authors (e.g. MacArthur et al., 2008) often simplify the graphs and analyze only the underlying structure, which is given by the undirected and unweighted edges. But as described in Section 3.2.1, simplifying graphs by removing all “additional properties” like weights, loops, and multiple edges, is likely to “add” symmetry that is not existing in the original non-simple graph. This is why we finally decided to split the main analysis procedure into two parts: One that uses only the simple graphs and one that uses the remaining non-simple graphs and simplifies them. Disconnected graphs are excluded.

For each graph (whether simple or simplified), we obtained the number of nodes n, the number of edges m, and the density ρ. Furthermore, the randomized greedy algorithm (RG) for graph clustering (Section 2.5.2) was used to compute a modularity optimal partition and `saucy` was applied to compute the automorphism group of the graph. Modularity, the size of the group $|Aut(G)|$, the number of generators $|S|$, and the number of orbits $|O|$ is saved. `saucy` itself does not return the orbit partition, but it can be efficiently computed from the generators.

Yet another issue one must be aware of is the existence of duplicates. Sometimes the same graph appears in more than one category (already described above), sometimes a graph is duplicated within the same category. For the latter case, the datasets have different but similar names, which often already indicates a duplication. However, in some cases the identification of duplicates is not that easy. To deal with this issue, we used a semi-automatic approach that groups the analyzed datasets by several attributes (n, m, $|O|$, $|S|$, $|Aut(G)|$) that all have to be the same for duplicated (thus isomorphic) graphs. The actual deletion of the duplicates is done after a manual inspection by hand. This procedure is fail-safe, as false positives or false negatives are very unlikely to occur.

From the 3015 downloaded datasets, a total of 1805 graphs are non-simple and another 277 graphs are disconnected. From the remaining 933 graphs, 31 were identified as duplicates. Most of them were part of the “DIMACS10” class, which contains datasets from a graph clustering and partitioning challenge held in 2012[7]. All graphs of this special class are also contained in one of the other classes.

For the second part of the analysis, the 1805 non-simple graphs were simplified. After simplification, 383 were disconnected and another 420 datasets were invalid, which here means that the provided adjacency matrix is not symmetric and it is not clear how to interpret this fact. One possible explanation could be that they are bipartite: One node set is described by the columns, the second node set by the rows. In the end, from the remaining 1002 graphs, only 797 were unique. This large number of duplicates is a result of the existence of many structurally identical graphs, which, e.g., have different edge weights.

4.3 Analysis Results

In this section, we describe the most interesting results that were obtained from the analyses of simple graphs (Section 4.3.1) and simplified graphs (Section 4.3.2). Each analysis consists of statistics of the complete sample and of the data grouped by their categories. Duplicates are eliminated for the first type of statistics, but kept for the second type to prevent a bias within the

[7] `http://www.cc.gatech.edu/dimacs10/`, as of January 2018

classes. All details can be found in Appendix B. The number of orbits is used to compute the normalized network redundancy r'_G (see Equation 4.8).

4.3.1 Simple Graphs

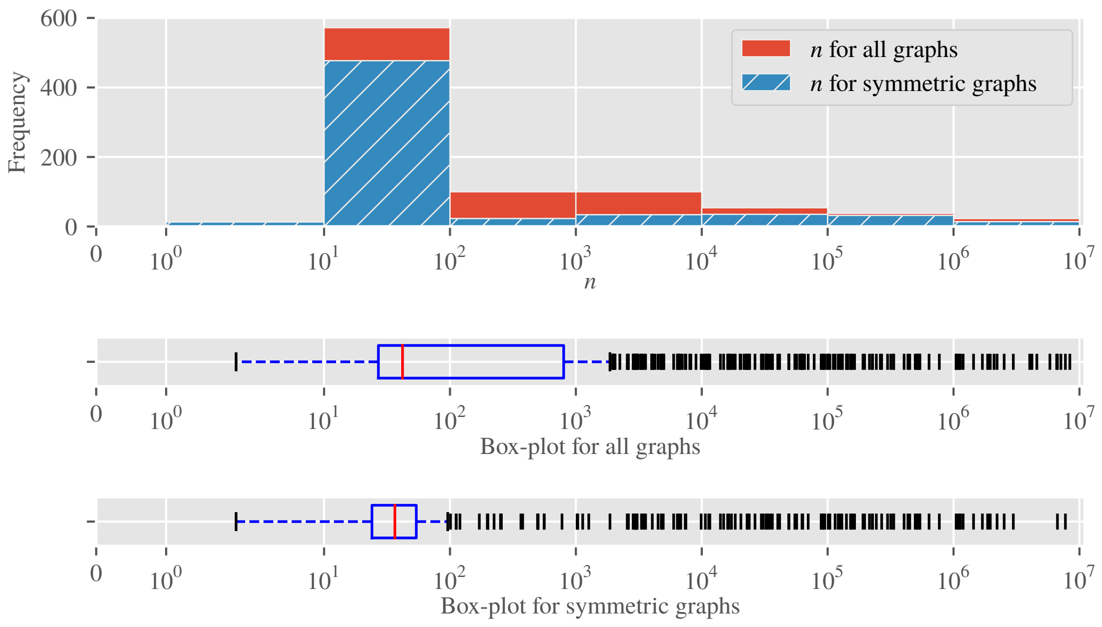
In Table 4.2, basic statistics for the different properties of the simple graphs can be found, which gives a first insight of how the data looks. The median values for n and m show that most graphs are quite small. Nonetheless, also relatively large graphs exist. The columns for density (ρ) and modularity (Q) underline that the graphs represent real-world networks: The density is low and the modularity high. Symmetries exist, however, at least 75 % of the graphs have a small automorphism group. Normalized network redundancy is about 0.18 or less for the same percentage of datasets. In contrast, also very symmetric networks exist as the maximum values for r'_G and $|Aut(G)|$ show. We identified only 272 out of the 902 networks to be completely asymmetric ($|Aut(G)| = 1$), which means about 70 % of the analyzed networks contain non-trivial automorphisms. Additionally, Figures 4.1–4.4 show histograms and box-plots for n, m, ρ, Q and r'_G and thus give more specific information.

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	902	902	902	902	902	902
mean	1.2796×10^5	4.1845×10^5	0.1981	0.5257	0.1514	$6.3659 \times 10^{2,517,003}$
std	8.6096×10^5	1.8654×10^6	0.2420	0.2469	0.2345	$1.9119 \times 10^{2,517,005}$
min	2	1	1.7801×10^{-7}	0	0	1
25 %	27	51	0.0654	0.4156	0	1
50 %	42	84	0.1176	0.5793	0.0625	4
75 %	800	24,489	0.2053	0.6601	0.1818	24
max	1.1951×10^7	2.5166×10^7	1	0.9987	1	$5.7420 \times 10^{2,517,006}$

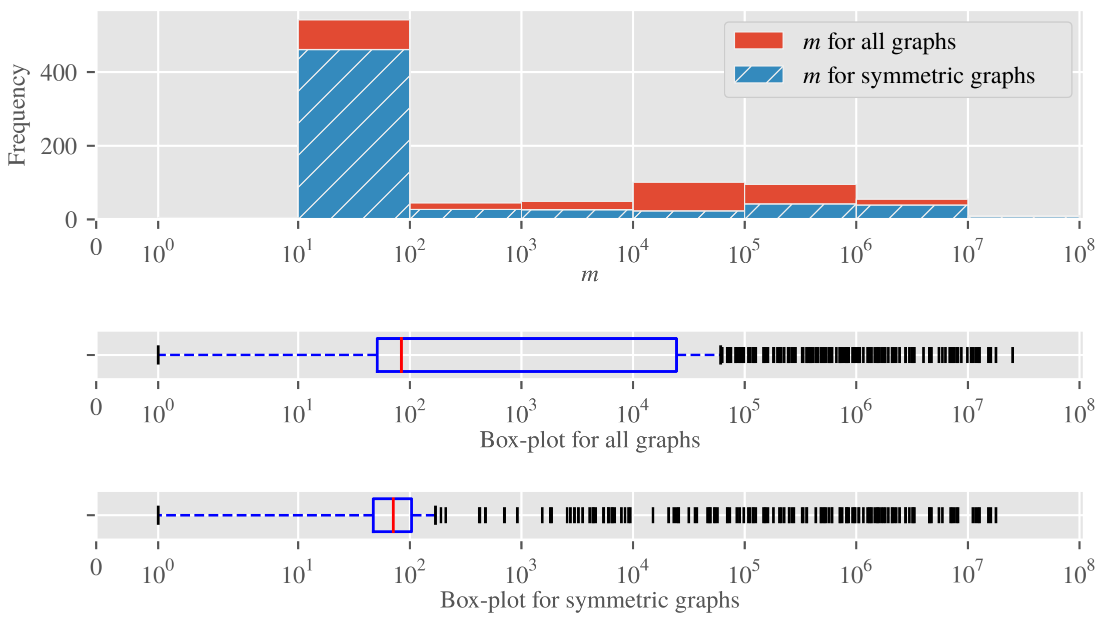
Table 4.2: Statistics for the `networkrepository.com` datasets: Only 272 of the 902 graphs are asymmetric. *count* is the number of datasets on which the statistics per column were computed on, *mean* (*std*) is the respective mean value (standard deviation). *min* and *max* are the minimal/ maximal values observed and the three remaining rows in between denote the two quartiles (25 % and 75 %) and the median (50 %). The columns n (m) yield statistics on the number of nodes (edges) of the graphs, Q is the modularity, which measures (clustering) partition quality. The last two columns contain information on the symmetries, r'_G is the normalized redundancy (see Equation 4.8) and $|Aut(G)|$ the size of the automorphism group.

The distributions for n and m in Figure 4.1 show a bias of the graph sizes. Most datasets consist of 10 to 100 nodes and/or edges. Looking into the data clarifies this fact: Over 570 datasets are part of the `chem` class (described below), which predominantly consists of graphs having less than 100 nodes/edges. There is also an underrepresentation of symmetric medium-sized graphs ($2 \leq \log_{10} n < 4$ and/or $4 \leq \log_{10} m < 6$), which will be discussed below.

How the density is distributed can be seen in Figure 4.2. The high amount of quite sparse graphs (i.e. with low density) is not surprising as its value normally decreases when the number of nodes and edges are increased by some fixed constant factors. This is due to the non-linear proportionality $\rho \propto \frac{m}{n^2}$. Moreover, real-world networks are normally highly modular and, as a matter of definition, only dense graphs can be highly modular. There even exists



(a)



(b)

Figure 4.1: Histogram and box-plots for the number of nodes n (a) and the number of edges m (b). Each figure distinguishes the symmetric graphs from all graphs. Note the log-scaled x-axes and increasing class sizes. Most of the networks have between 10 and 100 nodes/edges. There are significantly less symmetric graphs with $2 \leq \log_{10} n < 4$ ($4 \leq \log_{10} m < 6$) nodes (edges). This fact is explained in the text. However, there are symmetric graphs at all sizes.

the relation $\rho \leq 1 - Q$, which means that density must decrease with increasing modularity. This relation is proved in Appendix C. Furthermore, we have presented the relation between density and symmetry in Section 3.2.3. The conclusion was that there exists an upper bound for the number of edges depending on n for a graph to be asymmetric. This bound is $M_n =$

$\frac{n(n-3)}{2} + o(n)$, thus graphs with $m > M_n$ must be symmetric. For the density, this means $\rho = (\frac{n(n-3)}{2} + o(n))/(\frac{n(n-1)}{2}) = \frac{n-3}{n-1} + o(1/n)$. Therefore, high density graphs are likely to be symmetric.

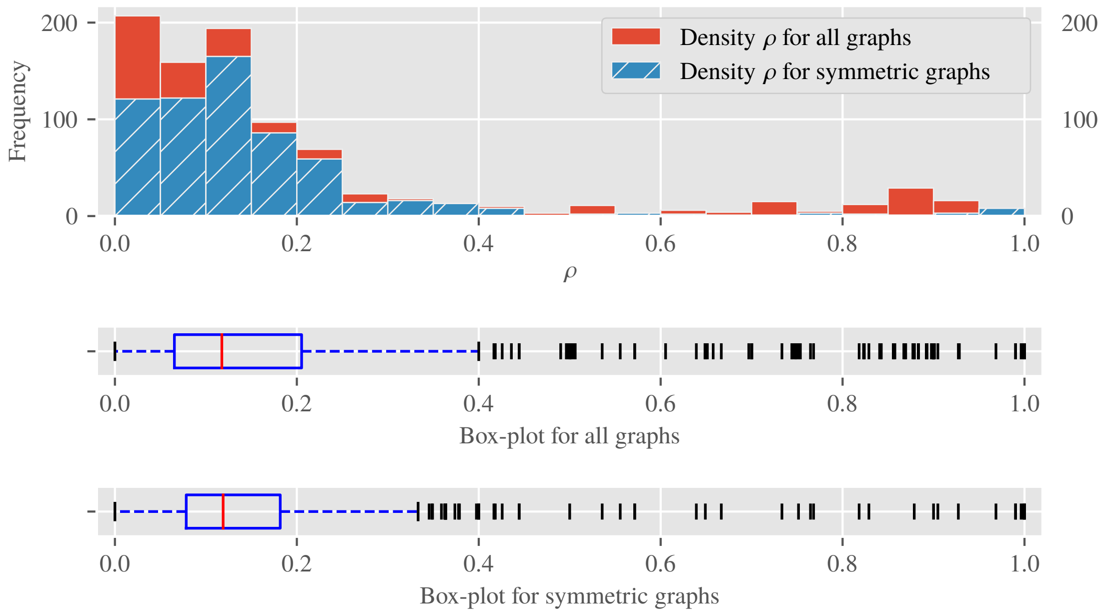


Figure 4.2: Histogram and box-plots for the density ρ, distinguished by all graphs and symmetric graphs only. Dense networks are likely to be asymmetric, except very dense networks ($\rho > 0.95$) are symmetric (which agrees with the symmetry bounds from Section 3.2.3).

As often pointed out (see Section 2.2.1), one large part of the characterization of real-world networks is their expected high modularity. Figure 4.3 meets these expectations. There is no valid limit for all graphs that distinguishes non-modular and modular graphs. Due to a matter of design, higher values of modularity are desirable (e.g. $Q > 0.8$) for large graphs whereas lower values for smaller graphs (e.g. $Q > 0.3$) are sufficient. The well-known Karate network (Zachary, 1977) is quite small ($n = 34$, $m = 78$) and has a maximal modularity of nearly 0.42 (Ovelgönne and Geyer-Schulz, 2010).

It is a bit surprising to have such a large amount of asymmetric graphs with low modularity ($Q < 0.05$), as can be seen in Figure 4.3. As discussed above, graphs with low modularity are very likely to have a high density. This makes it impossible to find a partition of the nodes that meets the requirement of having many more edges within the clusters as between them. By investigating these graphs in the leftmost bin of the histogram, we find them to have indeed a fairly high average density ($\overline{\rho}_{Q<0.05} = 0.788$, std 0.172). Most of the datasets come from the `dimacs` and `bhoslib` categories. The first one contains networks that were used for the second DIMACS challenge, which was about “Maximum Clique, Graph Coloring, and Satisfiability”[8]. Those datasets are generated for benchmarking reasons (e.g. to contain cliques of a certain size) and, therefore, are no proper real-world networks. All graphs in the latter category are

[8] `http://dimacs.rutgers.edu/Challenges`, as of January 2018

asymmetric, which will be discussed and described below. Another group of graphs come from the `misc` category and have the names `G1`, `G2`, and so on. Helmberg and Rendl (2000) describe how these datasets are generated, most of them emerge from a random model, and thus are no real-world networks, too. This explains the high amount of asymmetry and low modularity.

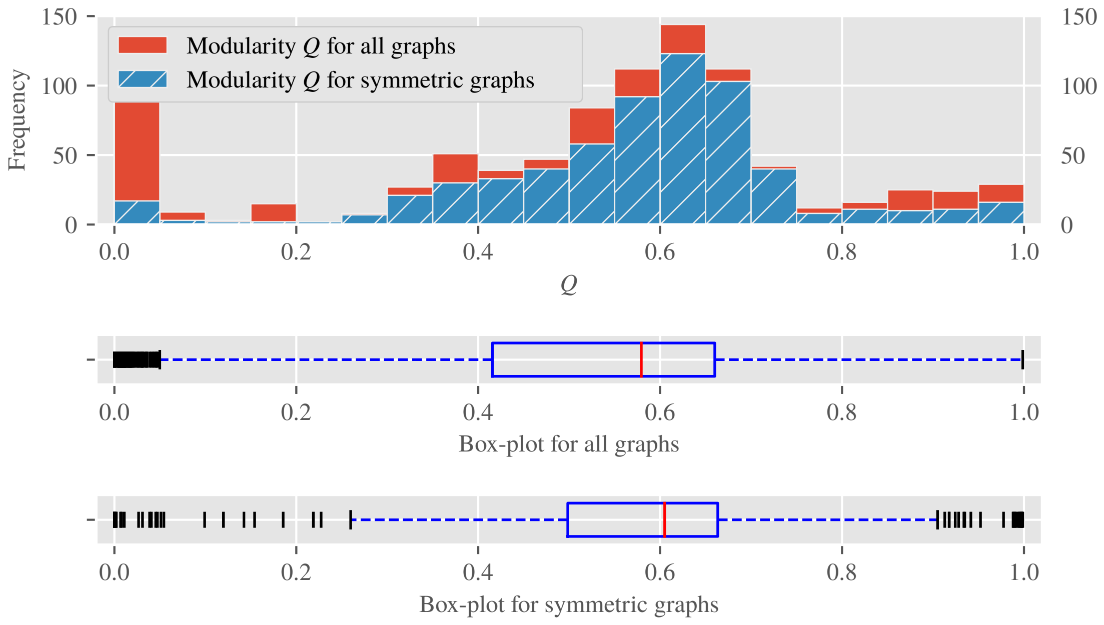


Figure 4.3: Histogram and box-plots for the modularity Q, distinguished by all graphs and symmetric graphs only. The bins all have equal width 0.05. Very non-modular networks seem to be notably asymmetric, which is a contradiction that is discussed and investigated in the text.

Up to this point, the only distinction made between graphs was symmetric versus asymmetric. Figure 4.4 shows the histogram of the normalized network redundancy. As already stated, most of the graphs have a relatively low normalized redundancy (over 80 % of them with $r'_G < 0.2$) and 272 of the datasets are asymmetric. On the other hand, 13 graphs are transitive. However, we remind the fact discussed in Section 4.1.2 and Appendix A, which is about the actual number of affected nodes that is not entirely reflected by the (normalized) network redundancy.

Going a step further, networks are now grouped by their type/category, which is shown in Figure 4.5. The bias of very unequal group sizes results from the unbalanced classification of networks (see Table 4.1) as well as from different properties many graphs of certain categories have (e.g. retweet networks are often disconnected or ecological networks are all weighted). Because of this, four categories (“Brain Networks”, “Ecology Networks”, “Massive Network Data”, “Dynamic Networks”) are not present in this analysis, the rest of the datasets is spread over 17 categories.

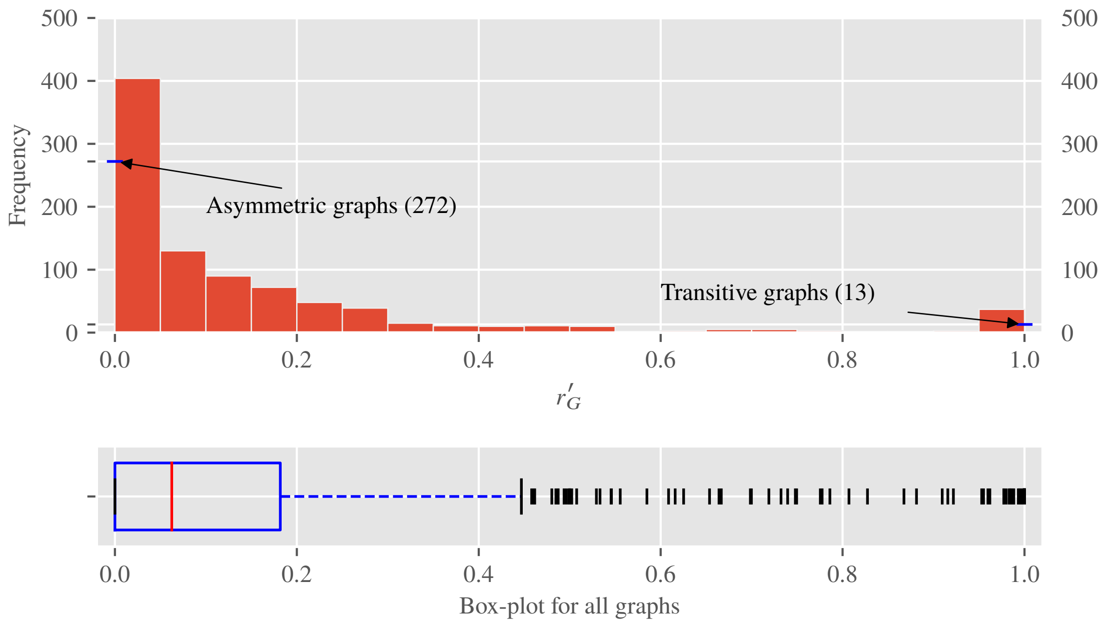


Figure 4.4: Histogram and box-plot for the normalized network redundancy r'_G. The numbers of asymmetric and transitive (only one orbit) graphs are emphasized, as they are contained in the two buckets $[0, 0.05)$ and $[0.95, 1]$. The distribution is right skewed, which is underlined by a relationship that often holds: The median (0.0625) is smaller than the mean (0.1514).

Next, we give additional information for the six largest categories:

- `bhoslib`[9]: All these graphs are asymmetric. They all are generated from the "Model RB" (Xu and Li, 2006), a derived random graph model that is able to generate a random constraint satisfaction problem (CSP) that can be used as benchmark for algorithms solving this problem. We do not want to elaborate on CSPs, but remind the fact that random graphs are very likely to be asymmetric (Erdős and Rényi, 1963).
- `chem`: This largest class contains cheminformatics data, which has overall a large degree of symmetry. The origin of all those networks is described by Borgwardt et al. (2005): They extracted proteins from an online enzyme information system and transformed them into undirected graphs. The nodes are labeled and we used this additional information by providing a partition of colors as parameter for the `saucy` call (see the details in Appendix B). `saucy` supports initially labeled nodes as starting point for the search procedure.
- `dimacs`[10]: These datasets come from the second DIMACS challenge. We already have seen that many of these graphs are asymmetric, but there is also a large amount of very

[9] "**B**enchmarks with **H**idden **O**ptimum **S**olutions for Graph Problems", `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`, as of 4 September 2017

[10] `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/volume/instruct.tex` as of March 2017

symmetric graphs (e.g. Hamming graphs, which are distance transitive, see Brouwer and Haemers (2012)).

- `dimacs10`: These are networks that were used in the tenth DIMACS challenge, which was about efficient graph clustering. Therefore, these graphs are highly modular (average modularity of nearly 0.9) and many of them contain symmetries as well. It is well worth to mention the Randomized Greedy algorithm and its iterated core group version (Section 2.5.2) has won the challenge in several categories (Bader et al., 2014).
- `misc`: This category contains datasets that seem not to fit in any of the other classes. Therefore, it is hard to make any valid assertions.
- `socfb`: Although these networks seem to be mostly asymmetric, only four of the 29 actually are (see also Table B.15 in the Appendix). All these graphs are quite large—more than 15,000 nodes on average—as they represent parts of the Facebook social network.

The 11 other classes contain nearly only symmetric graphs (54 out of 55).

The underrepresentation of symmetric medium sized graphs comes from the fact that all graphs from the `bhoslib` category, most graphs from the `dimacs` category, and many other asymmetric random graphs fall into the observed ranges of $2 \le \log_{10} n < 4$ for nodes and $4 \le \log_{10} m < 6$ for edges.

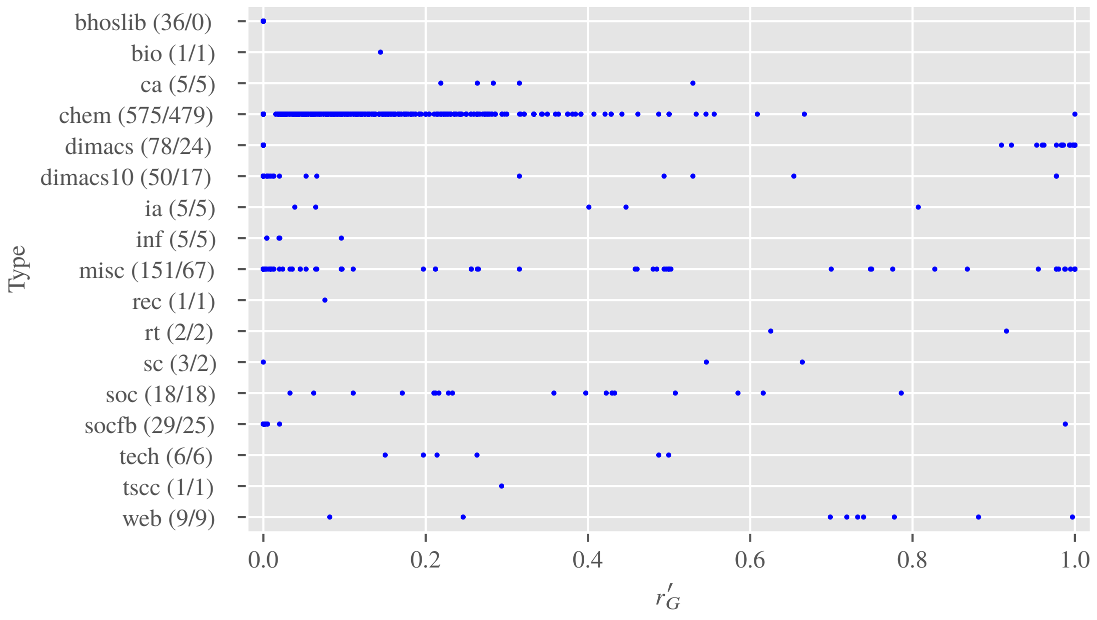


Figure 4.5: Overview of the normalized network redundancy grouped by type/category of the graphs. Duplicates are included to prevent a bias between classes. There is a total of 975 datasets, which are spread over 17 different categories. The numbers in parentheses are the count of networks of each category as well es the count of symmetric networks in the category. Full names of the types are given in Table 4.1.

4.3.2 Simplified Graphs

The results for the 797 simplified and unique graphs are presented in the same manner as in Section 4.3.1. By comparing Tables 4.2 and 4.3, we can see a number of differences: The set of analyzed simplified graphs contains more large graphs, as half of them have at least 10,605 nodes (89,927 edges) in contrast to 42 nodes (84 edges) for the simple graphs. Moreover, they are even sparser, have a higher average modularity (0.7345 compared to 0.5257), and are more symmetric (average r'_G of 0.4275 compared to 0.1514). The last fact is also underlined by less asymmetric graphs (163 of 797), which means that nearly 80 % of the simplified graphs are symmetric.

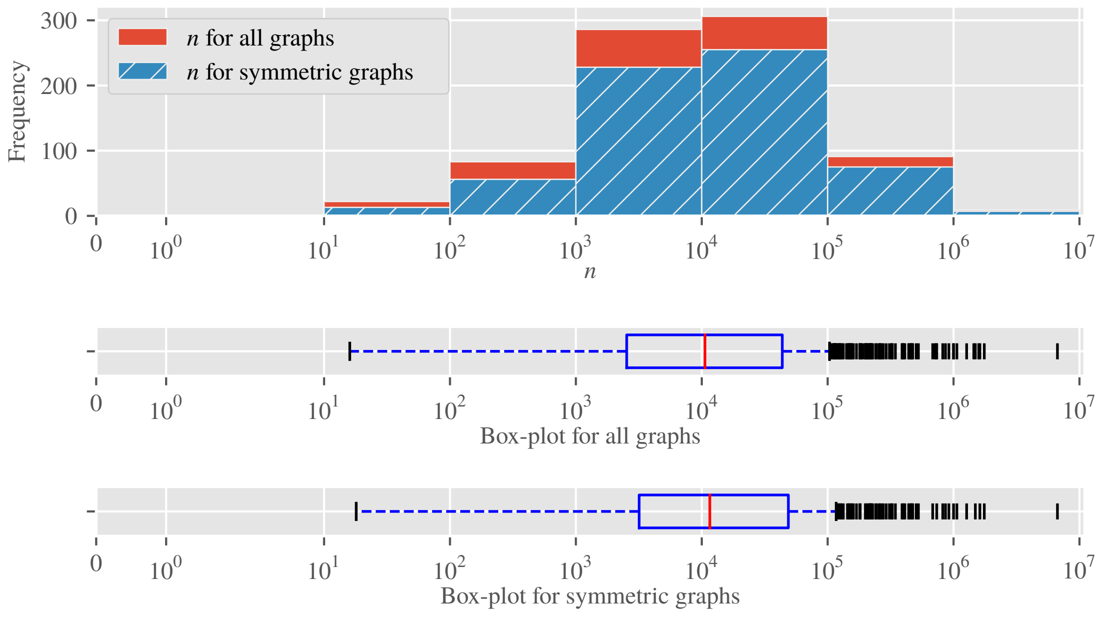
Once again, let us point out that the simplification of the graphs—by taking only the structural part into account—probably does not reflect the reality. We elaborated on this topic in Section 3.2.1. However, depending on the purpose of the analysis, simplifying a graph can be an appropriate instrument to reduce the complexity of the problem. Here, we are interested in existing graph symmetry as potential cause of “irregularities” in graph clustering. Most graph clustering algorithms only take the graph’s structural connections into account, a few others can deal with edge weights, too. Directed edges are often turned into undirected ones. All in all, we think it is appropriate to simplify the non-simple graphs as (i) this is the way to go in many applications, (ii) we can compare the results to those of the simple graphs, and (iii) every graph that emerges from a practical situation is already a simplification of reality, as normally not all properties that distinguish the entities represented by nodes are considered for the creation of the graph.

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	797	797	797	797	797	797
mean	70,041	7.0852×10^5	0.0169	0.7345	0.4275	$2.3286 \times 10^{7,237,014}$
std	2.9308×10^5	1.6665×10^6	0.0735	0.2005	0.3682	$6.5740 \times 10^{7,237,015}$
min	16	46	3.1376×10^{-7}	0	0	1
25 %	2534	11,173	0.0003	0.6518	0.0041	2
50 %	10,605	89,927	0.0015	0.7955	0.4925	3.3692×10^{32}
75 %	43,618	5.5493×10^5	0.0072	0.8788	0.7642	2.9131×10^{2486}
max	6.6865×10^6	1.7233×10^7	1	0.9977	1	$1.8559 \times 10^{7,237,017}$

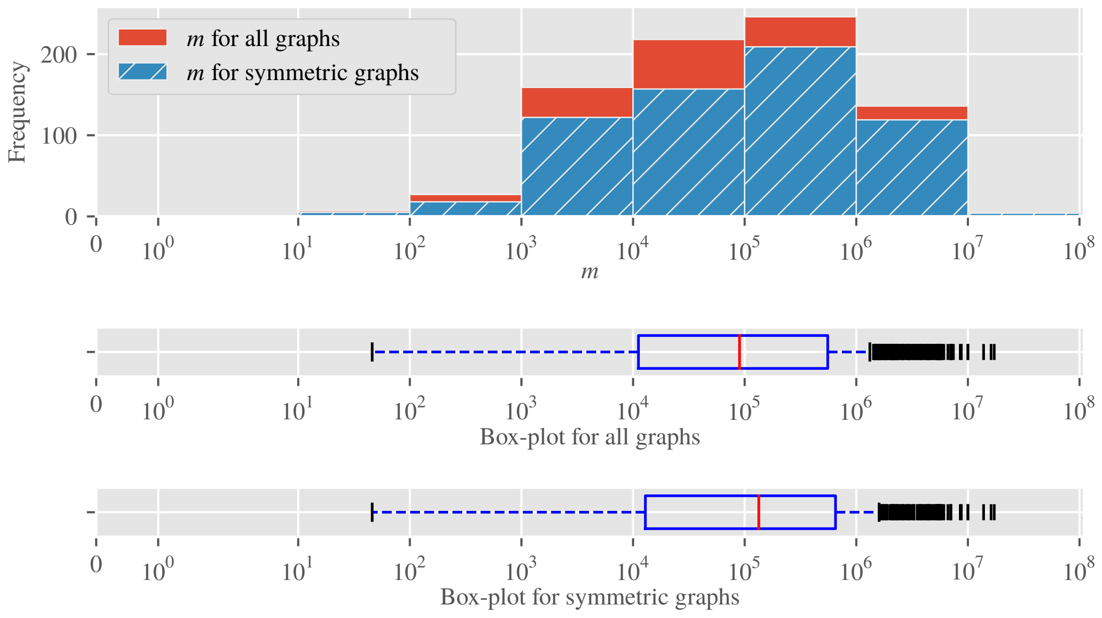
Table 4.3: Analysis statistics for `networkrepository.com` datasets: 163 of the 797 graphs are asymmetric

The two histograms and the corresponding box-plots in Figure 4.6 show that most graphs consist of 10^3 to 10^5 nodes, a strong bias of the network sizes, as it could be seen for the simple graphs, does not exist. The distribution of asymmetric graphs over the different sizes is very homogeneous, which can be concluded by comparing the box-plots of all graphs to those of the symmetric ones only: Median, quartiles, and extremes are close together, both, for nodes and edges.

The very low density is completely reflected by Figure 4.7. As the simplified networks are quite large, this “effect” is not unusual. Most growth models for real-world-like graphs add



(a)



(b)

Figure 4.6: Histogram and box-plots for the number of nodes n (a) and number of edges m (b). Symmetric and asymmetric simplified graphs are distinguished. Note the log-scaled x-axes and increasing class sizes.

nodes to an existing graph by adding a constant number of connecting edges. However, we have seen that the density decreases when there is a constant increase in the number of nodes and edges. Another explanation for this phenomenon could be that the simplified graphs represent other relationships compared to the simple graphs.

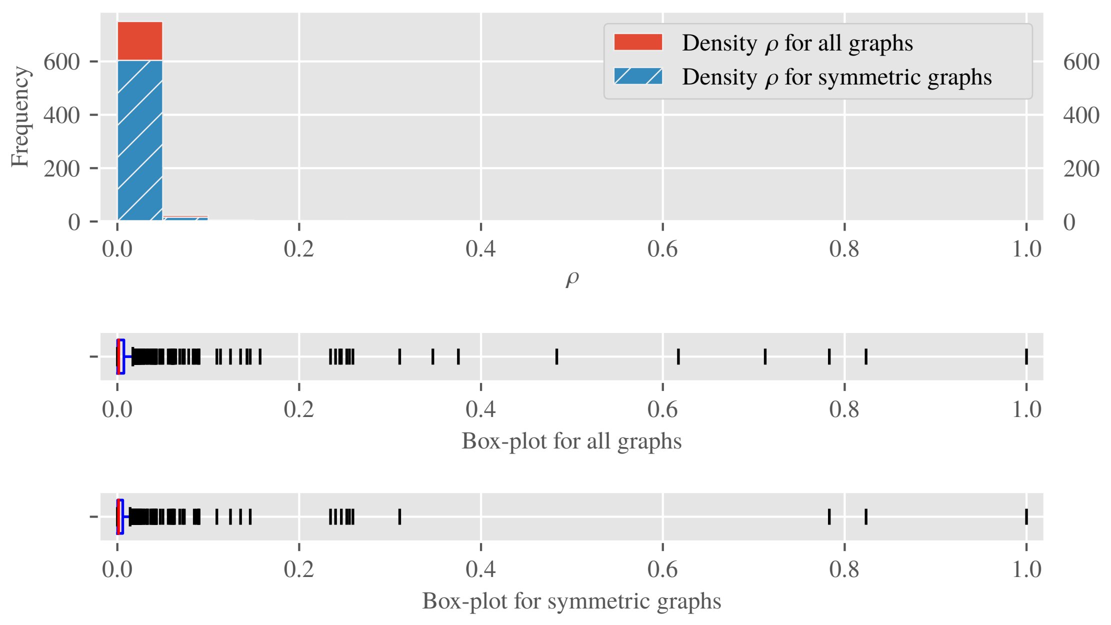


Figure 4.7: Histogram and box-plots for the density ρ, distinguished by all simplified graphs and symmetric graphs only.

The quite high average modularity also fits well with the overall large graph sizes. Again, the asymmetric networks are distributed relatively homogeneous among the different modularity values (see Figure 4.8).

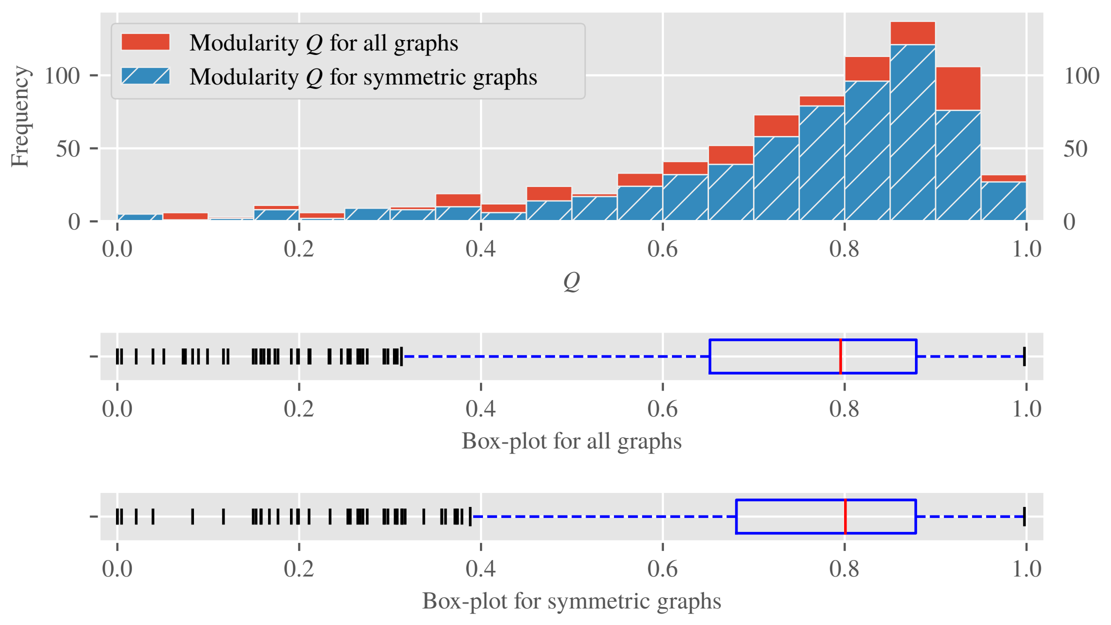


Figure 4.8: Histogram and box-plots for the modularity Q, distinguished by all simplified graphs and symmetric graphs only.

The distribution of the normalized network redundancy in Figure 4.9 looks quite different compared to Figure 4.4 for the simple graphs. There exists a large number of datasets with a relatively low r'_G (< 0.1). Only very few graphs have values of r'_G between 0.1 and 0.45. Then there is an erratic increase in networks with a normalized network redundancy larger than 0.45 up to 1. Together with the median value of nearly 0.5 one can conclude that the simplified networks are either nearly asymmetric or have a large degree of symmetry.

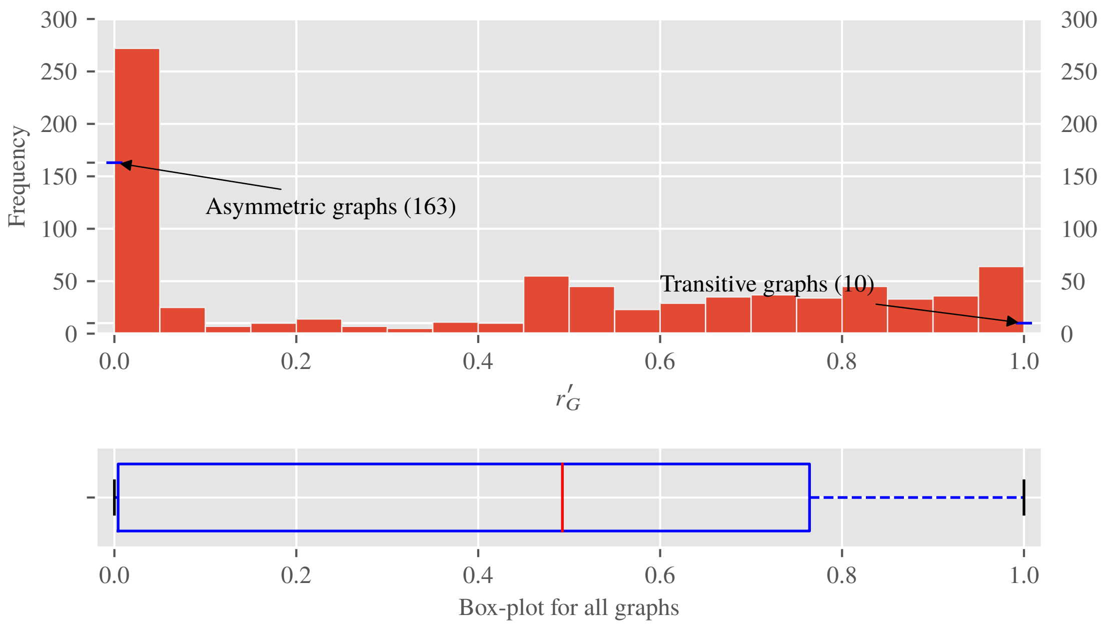


Figure 4.9: Histogram and box-plot for the normalized network redundancy r'_G. The numbers of transitive (only one orbit) and asymmetric simplified graphs are emphasized, as, again, the leftmost and rightmost buckets contain these special graphs.

In contrast to the category-wise analysis of the simple graphs, there are plenty of differences in Figure 4.10. The two largest categories are `misc` and `rt`, however the first one is by far dominant in terms of the number of graphs. Some categories that are present in Figure 4.5 now vanish (`bhoslib`, `ca`, `chem`, `dimacs`, `sc`, `socfb`, and `tscc`), others appear (`bn`, `dynamic`, and `eco`).

Dynamic networks all contain timestamps and a positive (1) or negative (-1) indicator as additional edge attributes. These carry the information at which point in time an edge was added or removed from the graph. As mentioned above, ecological networks are all weighted. This is why these datasets are not contained in the previous analysis of simple graphs. Again, for the 11 small categories, 27 of 33 graphs contain symmetries, which is a relatively large amount. And, as before, we have a more detailed look into the two largest categories:

- `misc`: The normalized network redundancy among the networks in the miscellaneous category is distributed very heterogeneously and—as in the simple case—there are no large subgroups of datasets that stand out. There are datasets that obviously belong together, identified simply by looking at their naming scheme. However, we are not interested in investigating the details in the context of this thesis.

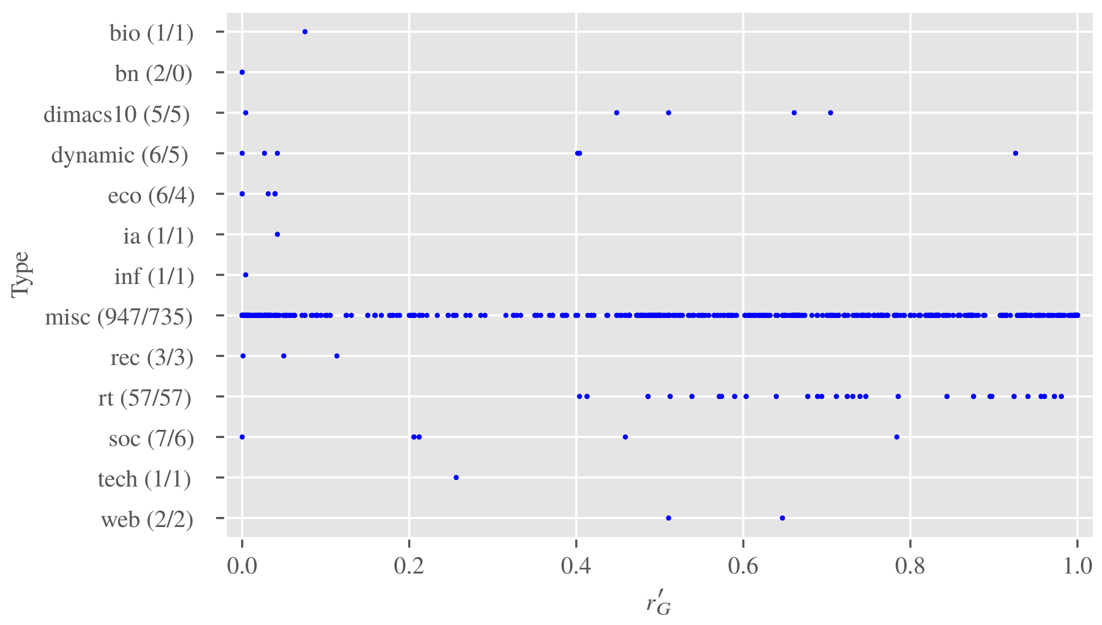


Figure 4.10: Overview of the normalized network redundancy grouped by type/category of the simplified graphs. Between-category duplicates are included to prevent a bias between classes, within-category duplicates are removed. There is a total of 816 datasets, which are spread over 13 different categories. The numbers in parentheses are the count of networks of each category as well es the count of symmetric networks in the category. Full names of the types are given in Table 4.1.

- `rt`: Retweet networks are tightly connected to the famous Twitter[11] social network, which allows registered users to publish short messages ("tweets"). These may contain additional content like images and so called hashtags, which allow to mark a message with unique tags. A core feature of Twitter is that users can share messages of others ("retweeting"), hence the name of the category. A user sees messages of other users (original or retweeted) he or she follows. These information can be used to create a graph of users that are connected to other users, e.g., if they retweet one or more messages of them. For instance, Cherepnalkoski and Mozetič (2016) present a method on how such a network can be created and analyzed. Normally, independent of the actual method of construction, retweet graphs represent only a small section of the full online social network. And additionally, these type of networks often contain a rather small amount of densely connected hubs (e.g. Newman, 2010, pp. 245/246), which, e.g., are very influential users like persons of interest from politics or entertainment, or institutional users like a government or news agencies. Although every user can publish messages that may be shared within the network, it is clear that the probability that messages of influential users are shared is much higher. Therefore, retweet networks normally consist of a few densely connected areas (communities) and many tree-like appendices around these areas. Figure 2.13 in Section 2.5.1 shows a visualization of the retweet network `rt_obama` from

[11] `https://twitter.com/`, as of January 2018

the `rt` category. The visualization shows the existence of some very dense areas on the one hand, and many tree-like, sparse appendices. As a consequence, the degree of symmetry of the retweet networks is relatively high, as can be seen in Figure 4.10.

4.3.3 A Comparison of the Symmetries in Simple and Simplified Graphs

Let us briefly review the key similarities and differences in the findings between simple and simplified graphs. The simple graphs are dominated by the datasets from the category of chem-informatics networks, whereas the simplified graphs are dominated by miscellaneous networks. There is also a major difference in the average graph sizes (in terms of the number of nodes and edges) that also directly influences the density, the modularity, and the average amount of symmetry. The main reason for the fact that the analyzed simple graphs are smaller on average, are the many small graphs from the `chem` category.

Also, there are a lot of datasets that emerge from some random graph model in the set of the simple graphs and they are far less symmetric or even asymmetric. This results in a less homogeneous distribution of the graphs among the bins of the histograms, especially modularity is distributed quite differently (Figures 4.3 and 4.8). Furthermore, simplified graphs are overall more symmetric, independent of their size. This fact is reflected by the distributions of the normalized network redundancy in Figures 4.4 and 4.9, which is a relative symmetry measure.

Besides some individual differences between simple and simplified graphs that emerge from divergent graph sizes, the two datasets have in common to contain numerous symmetric graphs in all sizes. Therefore, we conjecture that a certain degree of symmetry is another property that characterizes real-world networks, besides the properties presented in Section 2.2.1 (power-law distribution of node degrees, a high clustering coefficient, small-world effect).

4.3.4 Discussion

The purpose of this study was to find evidence of symmetry in real-world graphs. And, indeed, we have found strong evidence for the frequent existence of symmetry in real-world graphs. The two analyses with a total sample size of about 1700 network datasets uncover the frequent occurrence of graphs (1264 of 1699, i.e. over 74 %) that have a non-trivial automorphism group. The selection of the datasets follows rather a convenience sample, but we act on the assumption that a large number of networks that are often used in a scientific environment (e.g. for performance benchmarks in graph clustering) are covered by our analyses. It can be seen that there exist differences between different categories of networks, which gives room for further inspections. We tried to highlight at least some details of the larger categories, especially when there are interesting or unexpected results.

5 Graph Symmetries

The results of the previous chapter show that symmetries in real-world graphs exist. In this chapter we give some ideas and facts what possible reasons for these symmetries can be (Section 5.1). Furthermore, precise definitions for *local* and *global* symmetry are given in Section 5.2 and a weak symmetry definition in terms of a node partition is presented in Section 5.3.

5.1 Reasons for Symmetry in Graphs

Only few references of actual reasons for symmetry in graphs could be found, but, as is noted in the introductory Chapter 1, existence of symmetry in physical systems is apparent. As graph representations are a general "modeling tool" for diverse circumstances, any transformation into a graph of a system/problem/phenomenon/... containing symmetry is a candidate to be symmetric itself. Before we discuss examples from the literature, different possible sources of symmetry in graphs are listed. These sources may also be the reason for any kind of weak symmetry. However, weak symmetry is not deepened further.

By (natural) law If a graph is constructed from a structure that follows certain (natural) laws that involve symmetry, the graph itself will be symmetric. For instance, Feynman et al. (1963, Chapter 52, heading) describe "Symmetry in Physical Laws". But also much "simpler" laws can lead to symmetry like, e.g., rules of a game, economical "rules", or the search path to the optimal solution of a mathematical problem (e.g. graph clustering).

By construction Symmetry may also emerge if it is necessary or preferred by construction. For example, redundancy (by design) of technical systems or data networks (e.g. the internet) can cause symmetry.

By simplification/abstraction A graph, due to its simplest mathematical definition, has no attributes that describe, e.g., nodes any further. Technically, a graph even does not have node labels, as we have seen in Section 2.1.1. Nonetheless, in accordance to graph simplification described in Section 3.2.1, one can easily make up an extended definition of graphs that have additional attributes. However, graphs are mostly simple (or maybe weighted, directed, ...), which means they already represent merely the structure of a much more complex real-world system. Van Fraassen (1989, pp. 233 ff.) describes this fact in a philosophical way by stating that "[p]roblems which are essentially the same must receive essentially the same solution" (van Fraassen, 1989, p. 236). In practice, two problems are essentially the same if the transformed problems that result from only keeping the essential information (here: the structural relations that define a graph) are the same.

By aggregation This point is quite similar to the preceding one. However, in detail it differs: Instead of omitting unimportant attributes from an already given network structure, the network itself is built from a finer network by aggregating (e.g. clustering) subgraphs. In the resulting coarser network, these aggregates are simply represented by nodes and the relations between the aggregated subgraphs by edges. The exact relations between pairs of finer subgraphs are omitted. A possible construction of such graphs based on aggregation is defined in Section 5.3.

By error Measurement of any real-world data variables comes at the cost of possible errors. There exist diverse causes and they are not within the scope of this thesis. If, e.g., an entity that is represented as a node in the network and its relations are erroneously measured more than once, there exist exact copies of this entity, which results in "accidental" symmetry.

By chance The last possible cause for symmetry in networks is simply coincidence. It is perfectly possible by (possibly very small) chance that two or more nodes have exactly the same relations to all other nodes of the network. Although Erdős and Rényi (1963) and Magner et al. (2014) proved for several graph models that the probability of asymmetry tends to 1, this does not mean there cannot exist symmetry at all. Furthermore, many graphs are neither described by the ER random graph model nor by the preferential attachment model.

It is clear that also a combination of several causes can result in symmetric graphs.

The other way round, a graph may be asymmetric if the symmetry of the underlying structure was broken. Physicists distinguish explicitly and spontaneously broken symmetries (e.g. Brading et al., 2003, Section 4). Remember, for instance, the concept of "near automorphisms" (Emerson and Trefler, 1999) that was shortly discussed in Section 3.5: Two processes that may access and modify the same data source with identical states (non-trying, trying, critical section) are coupled together. One of them is a reader process; the other is a writer process that has a precedence over the former. The graph of the combined possible state transitions (i.e. all pairs of states for both processes) is asymmetric due to the precedence of the writer. Therefore, this precedence breaks the symmetry.

Let us now come to some examples in the literature for most of the above reasons. Chung et al. (2003) present a *node duplication model*, which describes the formation and growth of biological networks better than others, like, e.g., the preferential attachment model (see Section 2.2.3). The basic idea of the graph model is to duplicate existing nodes and connect them to the rest of the graph in the same way as the copied node (full duplication) or only partially the same (partial duplication). Nodes to copy are chosen at random with probability proportional to the orbit lengths. They justify their model with two facts: (i) Preferential attachment models can produce only graphs whose degree distribution follows a power law with exponent $\alpha > 2$, while in biological networks $1 < \alpha < 2$ is likely to be true, and (ii) biological processes involve a high degree of duplication of information (e.g. duplication of genome segments (Chung et al., 2003, p. 684)). The graph model itself, as well as the justification of it, show that—especially in biological networks—a certain amount of symmetry "by law" can be expected: Full duplication of a node directly results in exact symmetry of the graph, partial duplication at least involves a certain degree of weak symmetry, as the duplicated node is nearly identical to the one copied. The number of biological networks analyzed in Chapter 4 is unfortunately very small (one simple, one simplified graph) so that we cannot underline the elaboration of Chung et al. (2003). However, both networks are actually symmetric. Wang et al. (2012) give further evidence that symmetry in biological networks is not unusual.

Expander graphs describe highly connected networks that are at the same time very sparse (Hoory et al., 2006). Their construction involves a high degree of symmetry (see, e.g., Hoory et al. (2006, section 11) or Chee and Ling (2002)) and they have several applications, like constructing telephone networks or virtual circuits (Chee and Ling, 2002, p. 294). Therefore, systems that are designed on the basis of the properties of expander graphs involve a high degree of symmetry and are an example for graph symmetry by construction. Another example is the anonymization method for social networks of Wu et al. (2010). They define *k-symmetry anonymity* as the property of a graph to have orbit lengths of at least k. If a graph is not k-symmetric, orbits are copied and the new nodes are connected to the rest of the graph so that the orbit lengths increase. The details of the whole anonymization procedure is not of importance here, but it is clear that such a k-symmetric graph with $k > 1$ is symmetric *by design*.

An example for symmetry by aggregation comes again from the biological domain, where protein-protein-interaction (PPI) networks are used to model interdependencies between proteins. Proteins are macromolecules that consist of amino acid residues, which are themselves smaller molecules (built from atoms). The proteins can be represented as graphs, as we have seen in the discussion of the graph symmetry analysis results (the `chem` class in Section 4.3.1). However, in PPI networks the proteins are modeled as nodes and the edges represent interactions induced by the formation of covalent bonds (Sardiu and Washburn, 2011, p. 23647). Stelzl et al. (2005) create a human PPI network and—although the authors do not explicitly analyze it regarding symmetry—it can be seen that the resulting graphs (Stelzl et al., 2005, Figures 5 A and B) are definitely symmetric. This symmetry could possibly vanish if the graph would have been created by using the exact relations on the smaller molecule level or even on the atomic level.

Yet another example for symmetry by aggregation is given by Lorrain and White (1977). They describe a method for the analysis of social networks, which takes different relations between individuals into account by utilizing a category theoretic approach. Each relation (which the authors call "role") is a morphism, and several morphisms can be composed to new morphisms. The authors define two objects (i.e. entities of the social relationship) as structurally equivalent if all morphisms are exactly the same (Lorrain and White, 1977, p. 81). Note that this definition is just the one of graph automorphisms, as the morphisms are represented by edges. If structurally equivalent objects are identified, they are combined into a reduced category. This symmetry, however, results from an aggregation of several different relations between the objects.

We could not find any examples for symmetry by error or chance in the literature. This is, of course, not too surprising, as both causes can not be easily detected by their nature. Nonetheless, we believe that the symmetry in the Karate network (Zachary, 1977) could be seen as an example for symmetry by chance: The nodes are members of a karate club, who are related to each other in terms of friendships. The existing symmetries (see Figure 6.6 in Chapter 6) are not explained by Zachary (1977), and they are only very local (for a formalization of local symmetry, see the next Section 5.2). The emergence of the graph is based on the fact that there was a fission of the club's members into two groups, one led by the president of the group and one by the instructor of the karate lessons. The groups are formed by the supporters of either of the two influential persons in the club (nodes 0 and 33). The persons represented by the nodes 14, 15, 18, 20, and 22 all have the mutual friends 32 and 33 (the club's president, see Figure 5.1).

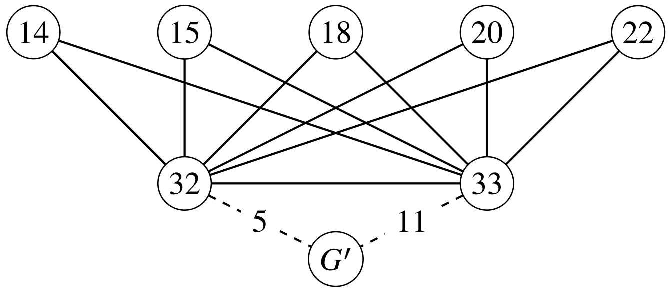


Figure 5.1: An extract of the Karate network (Figure 6.6) that shows a certain existing symmetry. G' denotes the rest of the graph. Nodes 14, 15, 18, 20, 22 are all equivalent; nodes 32 and 33 are not. The latter have a different number of connections to G', which is illustrated by the edge weights.

A simple explanation for the symmetry (the nodes 14, 15, 18, 20, and 22 are all equivalent) could be that person 32 brought friends to the club (thus all of them know this person) and introduced them to the president first (which definitely makes sense). However, these friends of 32 are from different social communities, which explains why they are not befriended with each other. Our attempt to give an explanation of the shown symmetry fits quite well to the idea of symmetry by chance, as this linkage pattern seems to be quite obvious and is likely to occur (also discovered by MacArthur et al., 2008).

Xiao et al. (2008b) introduce a modified preferential attachment model (see Section 2.2.3), which they call *similar linkage pattern*. They are inspired by the findings of MacArthur et al. (2008) and analyze the actual occurrences of bicliques (full bipartite graphs) as subgraphs in real-world graphs. In the extended model, a parameter $\alpha \in (0, 1]$ controls the ratio between plain preferential attachment ($\alpha = 1$) and preferential attachment with single linkage ($\alpha < 1$). For $\alpha < 1$, graph instances of the model have a very high probability to be symmetric in the described way (existence of bicliques). We pick up this type of frequently occurring symmetric motif in Section 6.3.2, where we analytically show the impact of this symmetry on modularity clustering.

5.2 Local and Global Symmetry

The possible impact of a non-trivial automorphism group is closely tied to the number of nodes that are affected. The (normalized) network redundancy presented in Section 4.1.2 is based on the number of orbits. This measure reflects—as the name suggests—how redundant the structure of the graph is. However, network redundancy does not suffice to give an estimation on which and on how many areas in the graph are affected by the automorphism group. The length of an orbit gives a hint how many nodes are equivalent, nonetheless, it is not a good indication of how many nodes are affected *independently*.

Example 31. *Consider the two generating sets (for $j \leq n$, j even) $S_1 = \{(1\ 2), (3\ 4), \ldots, ([2j-1]\ [2j])\}$ and $S_2 = \{(1\ 2)(3\ 4)\cdots([2j-1]\ [2j])\}$. Both generated groups induce j orbits of length two, but the group generated by S_1 acts much more "locally" in terms of the number of affected nodes (only pairs of nodes are mapped onto each other independently), whereas $\langle S_2 \rangle$ always affects all $2j$ nodes at the same time (therefore "globally").*

5.2.1 Definitions of Local and Global Symmetry

To deal with the issue shown in Example 31, the idea of decomposing a group into k normal subgroups H_i, as defined in Section 2.4, is used. Additionally, the method of dividing a set of generators (MacArthur et al., 2008) can be applied to create these subgroups.

The result is a partition of the set of generators S into k disjoint parts and $Aut(G) = H_1 \times H_2 \times \ldots \times H_k = \langle S_1 \rangle \times \langle S_2 \rangle \times \ldots \times \langle S_k \rangle$ holds. Let us extend the definition of the support of a permutation to a set of permutations S:

$$\text{supp}\,(S) := \bigcup_{p \in S} \text{supp}\,(p)\,. \tag{5.1}$$

All subsets $S_i, S_j \subset S$, $i \neq j$ are pairwise support disjoint, which means that $\text{supp}\,(S_i) \cap \text{supp}\,(S_j) = \varnothing$ (provided $S/Aut(G)$ is decomposable). Clearly, if $\langle S \rangle = H$ then $\text{supp}\,(S) = \text{supp}\,(H)$ (for any finite permutation group H). Note also that

$$|\,\text{supp}\,(H)\,| = \sum_{o \in O(H):|o|>1} |o| \tag{5.2}$$

where $O(H)$ is the orbit partition of H.

By Theorem 9 (theorem of Frucht (1939)), there always must exist a connected and undirected graph whose automorphism group is isomorphic to a given permutation group. This is important here, as we want to define local symmetry in graphs, which—intuitively—should have something to do with a spatial proximity of nodes. This spatial proximity corresponds to short paths between nodes that is given by their connectedness to each other. The purpose of this definition is to understand how symmetry in complex real-world networks acts. The findings of MacArthur et al. (2008), which show how a graph's automorphism group can be decomposed into smaller permutation groups that act on small disjoint sets of nodes, suggest that the symmetry found in real-world networks is restricted to certain affected areas in the graph. Theorem 10 formalizes the idea that nodes that are affected by an indecomposable normal subgroup of the automorphism group must be all part of a connected area of the graph.

Theorem 10 (Connected orbits)**.** *Let G be a graph with the decomposable automorphism group $Aut(G) = \prod_i H_i$. Every subgroup $H_i \lhd Aut(G)$ induces a* ***connected*** *subgraph G_{H_i} via the edge set $E_{H_i} := \{uv \in E(G) \mid u \in \operatorname{supp}(H_i)\}$. The node set V_{H_i} contains all nodes that are adjacent by edges in E_{H_i}. It follows $Aut(G_{H_i}) \cong H_i$ (the groups are generally not identical, as H_i acts on $V(G)$ and $Aut(G_{H_i})$ acts on $V(G_{H_i}) \subseteq V(G)$). G_{H_i} describes the local area on which H_i acts in G.*

Proof. By definition, $\forall g \in H_i \; \exists ux \in E_{H_i} : [(ux)^g = vy \neq ux] \vee [g = \mathbf{1}]$, thus $g \in H_i \Rightarrow g \in Aut(G_{H_i})$. The other way round, $\nexists h \in Aut(G_{H_i}), h \notin H_i, ux \in E_{H_i} : (ux)^h = vy \neq ux$, because H_i is a group. Therefore, $h \in Aut(G_{H_i}) \Rightarrow h \in H_i$ and, as a consequence, $H_i \cong Aut(G_{H_i})$ holds.

Suppose G_{H_i} is disconnected, i.e. there exist at least two subgraphs $G^1_{H_i}$ and $G^2_{H_i}$ that are not connected by an edge: As H_i is an indecomposable subgroup of $Aut(G)$, there must exist an isomorphism $g \in H_i : (G^1_{H_i})^g = G^2_{H_i}$. Otherwise, H_i could be decomposed into support disjoint $H^1_i \times H^2_i \cong Aut(G^1_{H_i}) \times Aut(G^2_{H_i})$. However, then also two nodes $u \in V(G^1_{H_i})$, $v \in V(G^2_{H_i})$ with $v \in u^{Aut(G_{H_i})}$ must exist. Additionally, either $uv \in E_{H_i}$ or $ux, vx \in E_{H_i}$, as otherwise G would be disconnected. This is a contradiction and, therefore, G_{H_i} must be connected. □

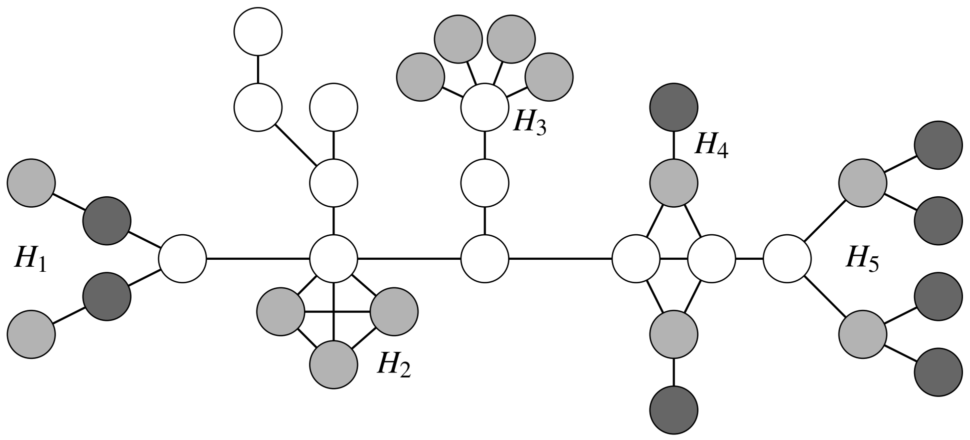
Example 32. *We use the example of MacArthur et al. (2008, p. 3529, Fig. 1) to show in Figure 5.2 how the different symmetric areas of a graph are characterized by Theorem 10.*

It follows that all permutations from a set of (essential, see Section 2.4.3) generators that do not fix the non-trivial orbit $u^{Aut(G)}$ must also act on the neighbors of u. Therefore, the decomposition of $Aut(G)$ into smaller non-trivial and normal subgroups H_i that cannot be decomposed any further reflects the different "areas" in the graph that are affected by independent symmetries. The support $\operatorname{supp}(H_i)$ of each subgroup H_i represents all affected nodes and it is the union of all the non-trivial orbits whose nodes are connected as described by Theorem 10.

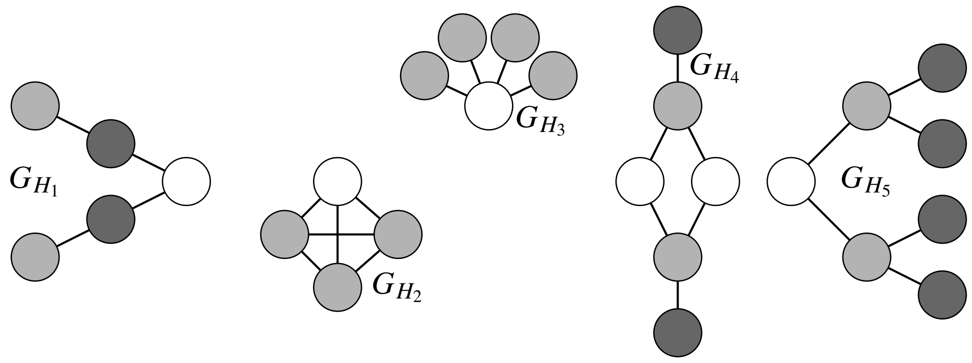
This connectedness allows us to state that all nodes affected by $H_i \lhd Aut(G)$ are only locally automorphic. The term "local" must always be considered relative to the size of the whole graph. If, e.g., a graph is transitive, $Aut(G)$ cannot be decomposed, as all nodes are affected and the symmetry is global. These considerations lead to the following two definitions:

Definition 1. *Given a graph G, its relative symmetry rs_G is given by*

$$rs_G := \frac{|\operatorname{supp}(Aut(G))|}{n}. \tag{5.3}$$



(a) The symmetric graph G with the decomposable automorphism group $Aut(G) = H_1 \times H_2 \times H_3 \times H_4 \times H_5$. Nodes on the same orbit (per subgroup H_i) have the same color, white nodes are fixed by $Aut(G)$. This example is from MacArthur et al. (2008, p. 3529, Fig. 1).



(b) The five connected subgraphs G_{H_i} of G that are induced by the indecomposable normal subgroups H_i.

Figure 5.2: Example of subgraphs induced by indecomposable subgroups H_i of $Aut(G)$.

Definition 2. *Given a graph G, whose automorphism group can be decomposed into k subgroups H_i as described above, its mean global symmetry $\overline{gs}_G$ is given by*

$$\overline{gs}_G := \frac{1}{k}\sum_{i=1}^{k}\frac{|\operatorname{supp}(H_i)|}{n}. \tag{5.4}$$

As the subgroups H_i are support disjoint by definition, the equation

$$\overline{gs}_G = \frac{|\operatorname{supp}(Aut(G))|}{kn} = \frac{1}{k} rs_G \tag{5.5}$$

is, of course, equivalent.

The relative symmetry in Definition 1 can also be considered as an overall measure of symmetry and the smaller the mean global symmetry from Definition 2 becomes compared to rs_G, the more local the symmetry acts in the graph. If $Aut(G)$ can not be decomposed, $\overline{gs}_G = rs_G$ follows. Otherwise, if $Aut(G)$ can be decomposed “completely”, each subgroup acts on exactly two nodes and there exist $k = |\operatorname{supp}(Aut(G))|/2$ subgroups in total. Normalizing Definition 2 leads to a relative measure of global symmetry.

Definition 3. *The normalized mean global symmetry for a given a graph G and its mean global symmetry* $\overline{gs}_G = \frac{|\operatorname{supp}(Aut(G))|}{kn}$ *is*

$$\overline{ngs}_G := \begin{cases} 0 & , \textit{if} \, |\operatorname{supp}(Aut(G))| = 0 \\ 1 & , \textit{if} \, |\operatorname{supp}(Aut(G))| = 2 \; . \\ \frac{\frac{1}{k} rs_G - \frac{2}{n}}{rs_G - \frac{2}{n}} = \frac{\frac{1}{k}|\operatorname{supp}(Aut(G))| - 2}{|\operatorname{supp}(Aut(G))| - 2} & , \textit{else} \end{cases} \tag{5.6}$$

The normalized global symmetry takes values between 0 and 1 and only depends on the number of affected nodes and on the number of decomposed subgroups. This means, any fixed nodes are not taken into account and the measure is constant in terms of the spatial proximity of symmetry.

Next, we discuss some of the properties of $\overline{ngs}_G$:

- The minimum value of 0 is only reached if $Aut(G)$ can be decomposed into k groups with $|\operatorname{supp}(H_i)| = 2, \; \forall i = 1, \ldots, k$.
- The maximum value of 1 is reached if $k = 1$, which means $Aut(G)$ can not be decomposed.
- If G is asymmetric, $\overline{ngs}_G = 0$. There is no meaningful interpretation of this fact; however, it does not make sense to try to distinguish local from global symmetry if no symmetry exists.
- The general form in Equation 5.6 is undefined for $|\operatorname{supp}(Aut(G))| = 2$. However, the value of $\overline{ngs}_G$ can safely be set to 1 in this case, as no decomposition of the automorphism group can be performed. Thus the symmetry is global.
- Not every value in the interval $[0, 1]$ can be reached: For $k > 1$ follows $\frac{\frac{1}{k}|\operatorname{supp}(Aut(G))| - 2}{|\operatorname{supp}(Aut(G))| - 2} < \frac{1}{2}$. This first seems counter-intuitive, but makes sense from a mathematical point of view, as $\lim_{|\operatorname{supp}(Aut(G))| \to \infty} \overline{ngs}_G = \frac{1}{k}$.
- As seen in the previous bullet point, $\overline{ngs}_G$ can be approximated for large $|\operatorname{supp}(Aut(G))|$ by $\frac{1}{k}$.
- The measure does not take into account the distribution of the different subgroup supports. But what would it mean, if the group could be decomposed into, e.g. say, two subgroups with supports $|\operatorname{supp}(H_1)| = |\operatorname{supp}(H_2)|$ or $|\operatorname{supp}(H_1)| \gg |\operatorname{supp}(H_2)|$, respectively? How could the different sizes be interpreted? Would the one area of the graph be more globally affected than the other? There is no simple answer to that, which could be used to pour this information into a single measure.

5.2.2 Examples

Example 33. *Given the graph in Figure 5.3, which has* $n = 3j + 1$ *nodes. It has a total of j independent symmetries, as only each pair of nodes connected to the path* $2j + 1, 2j +$

$2,\ldots,3j,n$ can be mapped onto each other by a simple transposition (e.g. $(1\ 2)$). There are $|\operatorname{supp}(Aut(G))| = 2j$ nodes affected, which results in a relative symmetry of $rs_G = \frac{2j}{3j+1} \approx \frac{2}{3}$ (for large j). The automorphism group can be "perfectly" decomposed ($j = k$) into $Aut(G) = H_1 \times \ldots \times H_k = \langle(1\,2)\rangle \times \ldots \times \langle(2j-1\,2j)\rangle$, which yields a mean local symmetry of $\overline{gs}_G = \frac{1}{j} \cdot \frac{2j}{3j+1} = \frac{2}{3j+1}$ and a normalized global symmetry of $\overline{ngs}_G = \frac{\frac{1}{j} \cdot 2j-2}{2j-2} = 0$. All orbits have a length of two.

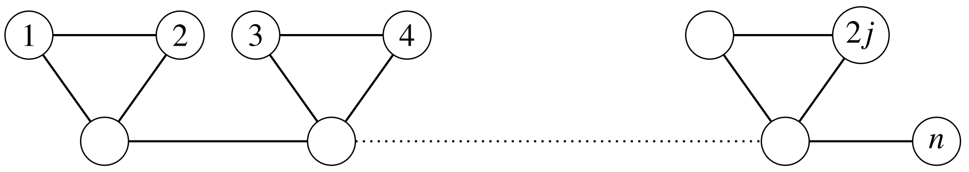


Figure 5.3: A relatively symmetric graph ($r'_G = 1 - \frac{(2j+1)-1}{(3j+1)-1} = \frac{1}{3}$), where about two thirds of all nodes are affected by the automorphism group. However, all symmetries are completely local in terms of the Definitions 1–3.

In contrast to Figure 5.3, the graph in Figure 5.4 also has only j short orbits of length two and $n = 2j + j + 1$ nodes. However, its automorphism group can not be decomposed, which results in a very global symmetry: Again, $rs_G = \frac{2j}{3j+1}$, but $\overline{gs}_G = \frac{1}{1} \cdot rs_G = rs_G$ and $\overline{ngs}_G = \frac{rs_G-2}{rs_G-2} = 1$.

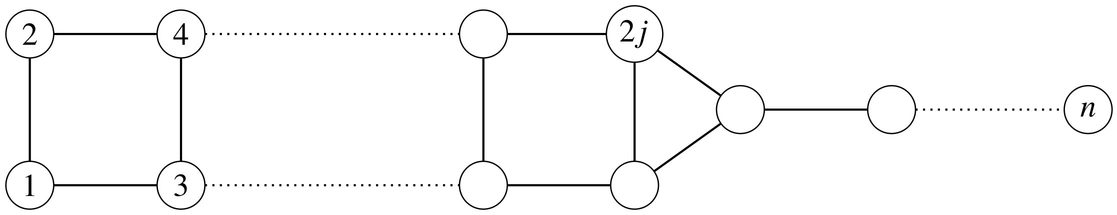


Figure 5.4: Another relatively symmetric graph, which shares many properties with the graph in Figure 5.3 (same number of nodes and orbits), but is in contrast to it completely globally symmetric.

Note that the normalized global symmetry of the graph in Figure 5.4 would be the same if there was only one additional node $2j + 1$ to break the symmetry "along the y-axis". We constructed the graph in Figure 5.4 to keep the number of nodes between this one and the graph in Figure 5.3 identical and, therefore, have the same relative symmetry. This underlines that $\overline{ngs}_G$ is a relative measure of local graph symmetry. These two graphs are examples for the two different sets of generators discussed in Example 31, which motivated the idea of local symmetry.

Example 34. *Consider again the graph in Figure 5.2. It has $n = 33$ nodes and $|\operatorname{supp}(Aut(G))| = 21$. Therefore, G has a relative symmetry $rs_G = \frac{21}{33} \approx 0.636$ and a mean global symmetry of $\overline{gs}_G = \frac{1}{5} \cdot \frac{21}{33} = \frac{7}{55} \approx 0.127$, as $Aut(G)$ can be decomposed into $k = 5$ support disjoint subgroups. The normalized mean global symmetry is $\overline{ngs}_G = \frac{\frac{1}{5} \cdot 21-2}{21-2} = \frac{\frac{11}{5}}{19} \approx 0.116$.*

5.2.3 Implications of Local Symmetry

The reason why we motivated and introduced local and global symmetry is simple: Graph clustering (algorithms) merge nodes that are similar in terms of their connections to each other into the same cluster. Hence, if only quite local symmetries exist, the probability that they affect a clustering solution by mapping nodes from one cluster to nodes of other clusters decreases with larger cluster sizes. Nonetheless, the mean global symmetry must always be seen in contrast to the relative symmetry of the graph and, additionally, it is not sufficient to judge the actual impact of symmetry. That is because the clustering partition of a relatively symmetric graph with very global symmetry can, nevertheless, be unaffected by the symmetry. This effect is shown in Section 6.3 for two specific locally symmetric structures for modularity clustering. Furthermore, the actual impact of existing symmetry on modularity clustering partitions is investigated in Chapter 7.

It would be even possible to further decompose the support disjoint normal subgroups. As we have seen in Section 2.4.3, (permutation) groups can be written as products of smaller groups. For the following explanations we refer to Aschbacher (2004). A (finite) group is called simple if there do not exist any normal subgroups other than the trivial group, which only contains the identity permutation and the group itself. For a group G, one can write $\{\mathbf{1}\} = G_0 \lhd G_1 \lhd \ldots \lhd G_n = G$, which is a series of proper normal subgroups, thus named *normal series*. If no proper normal subgroup is "left out" (i.e. $\nexists i', G_i \lhd G_{i+1} : G_i \lhd G_{i'} \lhd G_{i+1}$), the series is called *composition series* and each G_{i+1}/G_i is itself a simple group called *factor group*. G_{i+1}/G_i is defined as the set of cosets of G_i concerning G_{i+1}, i.e. $G_{i+1}/G_i := \{gG_i \mid g \in G_{i+1}\}$ (recall that $gG_i = \{gh \mid h \in G_i\}$ is a coset of G_i; see Section 2.4), together with the group operation $gG_i \circ hG_i = ghG_i$. The operation is a result of the normality of G_i, as per definition $gG_i = G_ig$ and thus $gG_i \circ hG_i = (gG_i)(hG_i) = g(G_ih)G_i = g(hG_i)G_i = (gh)(G_iG_i) = (gh)G_i$. The group G_{i+1}/G_i is, therefore, defined on the cosets of G_i, i.e. not on its elements, and each G_{i+1}/G_i is isomorphic to a specific simple group whose classification problem is the issue Aschbacher (2004) addresses.

Most interestingly, there exists only a small number of different types (not groups!) of finite simple groups that are sufficient to compose any other finite simple group. Moreover, the factorization of a group needs not be unique but, nonetheless, these finite simple groups are often analogously referred to in the same way as prime numbers, which can be used to factorize any integer numbers (Aschbacher, 2004, p. 736). Within the scope of this thesis, we do not go into further detail of this topic. However, further group decomposition could maybe give additional insights into the structure of complex symmetric networks.

5.3 Weak Symmetry Based on Clustering

Picking up the idea from Section 2.5.1 of seeing the matrix $\boldsymbol{E}$, which is a result of computing the modularity of a partition P, as the adjacency matrix of a partition induced graph $G(P)$, we now want to give a weak symmetry definition.

First, let us repeat the definition of $\boldsymbol{E} = (e_{ij})$. P is a partition of $V(G)$, possibly as the result of a clustering algorithm. The entries of $\boldsymbol{E}$ are defined as $e_{ij} = \frac{\sum_{v_x \in C_i, v_y \in C_j} m_{xy}}{2|V(E)|}$, where $\boldsymbol{M} = (m_{xy})$ is the adjacency matrix of G. The normalizing factor $(2|V(E)|)^{-1}$ is only important for the modularity definition and not for the following idea, but it is kept to make clear the analogy between modularity and weak symmetry.

Definition 4. *Given a graph G, a partition P of $V(G)$, and the matrix $\boldsymbol{E} = (e_{ij})$ that is derived from the definition of modularity (see Equation 2.105). P induces the undirected weighted graph $G(P)$ with adjacency matrix*

$$\boldsymbol{A}_{G(P)} = (a_{ij}) = \begin{cases} e_{ii} & i = j \\ 2e_{ij} & i \neq j \end{cases}. \tag{5.7}$$

The loops of $G(P)$ represent the intra-cluster edge fractions and every node i corresponds to the cluster $C_i \in P$. The weight a_{ij} of an edge ij ($i \neq j$) reflects the fact that the entries $e_{ij} = e_{ji}$ correspond to directed weighted edges of the graph with adjacency matrix $\boldsymbol{E}$. This concept is similar to the quotient graph described in Section 3.4.

Example 35. *Let G be the graph in Figure 5.5a with the partition $P = \{\{1,\ldots,5\},\{6,\ldots,10\}\}$. The adjacency matrices are given in Equation 5.8.*

$$\boldsymbol{A}_G = \begin{pmatrix} 0&0&1&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0\\ 1&0&0&1&1&0&0&0&0&0\\ 0&1&1&0&1&0&0&0&0&0\\ 0&0&1&1&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&1&0&0\\ 0&0&0&0&0&1&0&1&1&0\\ 0&0&0&0&0&1&1&0&0&0\\ 0&0&0&0&0&0&1&0&0&1\\ 0&0&0&0&0&0&0&0&1&0 \end{pmatrix} \overset{\text{given } P}{\rightsquigarrow} \quad \boldsymbol{E}_G = \begin{pmatrix} \frac{5}{11} & \frac{1}{22} \\ \frac{1}{22} & \frac{5}{11} \end{pmatrix} \tag{5.8}$$

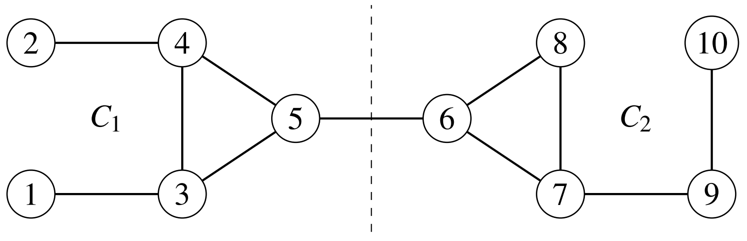
It is important to note that $\boldsymbol{E}_G$ is implicitly the adjacency matrix of a directed weighted graph (Figure 5.5b) due to the definition of modularity: The entry $e_{12} = \frac{1}{22}$ reflects the inter-cluster connections from C_1 to C_2 and e_{21} vice versa. Derived from $\boldsymbol{E}_G$, the adjacency matrix $\boldsymbol{A}_{G(P)}$ is shown in Equation 5.9.

$$\boldsymbol{E}_G = \begin{pmatrix} \frac{5}{11} & \frac{1}{22} \\ \frac{1}{22} & \frac{5}{11} \end{pmatrix} \quad \rightsquigarrow \quad \boldsymbol{A}_{G(P)} = \begin{pmatrix} \frac{5}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{5}{11} \end{pmatrix} \tag{5.9}$$

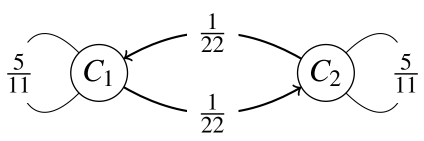
The edge between the nodes C_1 and C_2 of the graph $G(P)$ has weight $\frac{1}{11}$, which is the sum of the weights of the two directed edges e_{12} and e_{21} (Figure 5.5c).

Based on this coarsened graph definition, we now define weak symmetry.

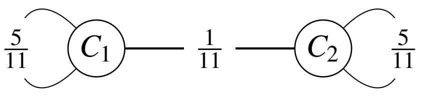
Definition 5. *Given a partition P of a graph G that induces the graph $G(P)$. If $|Aut(G(P))| > 1$, G is weakly symmetric.*



(a) A graph G divided into two clusters. $Aut(G) = \langle (1\,2)(3\,4) \rangle$



(b) The directed weighted graph that has adjacency matrix $\boldsymbol{E}_G$.



(c) The partition induced subgraph $G(P)$ with the adjacency matrix $\boldsymbol{A}_{G(P)}$ that is derived from $\boldsymbol{E}_G$.

Figure 5.5: Example of a partition induced subgraph of a graph ($n = 10$, $m = 11$) and a given partition of two clusters (represented by the dashed line).

Definition 6. *A graph G is strictly weakly symmetric given a partition P if* $|Aut(G)| = 1$ *and* $|Aut(G(P))| > 1$.

Definitions 5 and 6 formalize the idea that parts of a graph (i.e. subgraphs) are nearly the same, but no exact automorphism that would map one part onto the other exists. This does not mean that the original graph must be asymmetric itself (except in the strict case) as Example 35 shows. Reasons for this can be

- different connections of the parts to the rest of the graph or
- slightly different connections within the parts.

The clusters are regarded as equivalence classes and the partition induced graph aggregates these equivalent nodes. The exact connections between those nodes are relaxed, only the number of connections (the intra-cluster edges count) remains. They are represented by the weighted loops. The same happens for the inter-cluster connections, which are also aggregated to single weighted edges. As a result, the partition induced graph is made up of aggregated information from the source graph. Furthermore, symmetries of the partition induced graph itself are weak symmetries of the original graph.

See for example Figure 5.5: The graph is weakly symmetric given the partition into two clusters, as clearly $Aut(G(P)) = \{\mathbf{1}, (C_1\ C_2)\}$. In loose accordance to geometry, we want to call two subgraphs that are induced by $C_i, C_j \in P$ for some partition P congruent if $(C_i\ C_j) \in Aut(G(P))$. Jiang et al. (2017) make use of a similar technique to match two graphs based on a coarser aggregation level of the nodes (i.e. by using partitions).

6 Partition Stability

In this chapter we present several formal definitions of partition stability (Sections 6.1 and 6.2). By this we mean the property of a partition (as defined in Section 2.3) to be invariant under the graph's automorphism group. This means that applying a permutation (an automorphism) on the partition does not change it given the standard properties of a set.

Furthermore, we prove partition stability under certain circumstances for two often occurring graph motifs, namely stars and complete bipartite graphs, assuming modularity clustering is used (Section 6.3). We finally transfer the definition of Kolmogorov-Sinai Entropy to the graph clustering domain, which results in a quantitative uncertainty measure of the problem's solution space (Section 6.4).

6.1 Search Space Partition

In Chapter 2.4 orbits of permutation groups were defined. It could be seen that the group (of a graph) induces a partition of the elements (nodes) of the set on which the group acts, called orbit partition. But a group that acts on a set of elements also acts on combinatorial structures of this set (e.g. Beth et al., 1993, p. 149). Hence, $Aut(G)$ also acts on the set of partitions $\boldsymbol{P}(V)$. Each permutation $g \in Aut(G) : V \longrightarrow V$ acts on $P \in \boldsymbol{P}(V)$ by element-wise application of g:

$$P^g := \{C^g \mid C \in P\}, \tag{6.1}$$

$$C^g := \{v^g \mid v \in C\}. \tag{6.2}$$

This allows us to define the orbit partition of $\boldsymbol{P}(V)$ analogously to the orbit partition of V as

$$\boldsymbol{P}(V)^{Aut(G)} := \left\{P^{Aut(G)} \mid P \in \boldsymbol{P}(V)\right\}, \tag{6.3}$$

where the term $P^{Aut(G)}$ is the image set (i.e. the orbit)

$$P^{Aut(G)} := \{P^g \mid g \in Aut(G)\}. \tag{6.4}$$

From these definitions we want to deduce a first definition of partition stability.

Definition 7. *A partition P of a graph G is said to be stable if* $\left|P^{Aut(G)}\right| = 1$.

In the next Section 6.2 we present three stability definitions, which are all equivalent. As $Aut(G)$ induces a partition of equivalence classes of the node set V, it also induces an orbit partition of $\boldsymbol{P}(V)$. Analogously to the definitions from Section 2.3, the space of partition equivalence classes can be arranged in a diagram. Each node in the diagram is now an equivalence class of partitions and the partial ordering is given by $\leq_H$ (H is a group acting on V, e.g. $H = Aut(G)$), which is defined as

$$P^H, Q^H \in \boldsymbol{P}(V)^H \colon P^H \leq_H Q^H \;:\Longleftrightarrow\; \exists P' \in P^H,\ Q' \in Q^H : P' \leq Q'. \tag{6.5}$$

Example 36. *Consider again $\boldsymbol{P}(\{1,2,3,4\})$ from Figure 2.10. Let G be the "paw graph" shown in Figure 6.1 with its automorphism group $Aut(G) = \{(), (1\ 2)\}$.*

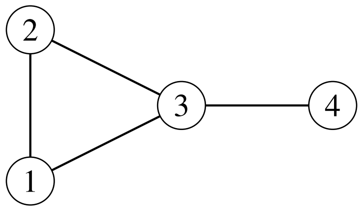


Figure 6.1: The paw graph whose space of symmetry induced partition equivalence classes is shown in Figure 6.2.

The equivalence classes induced by $\boldsymbol{P}(\{1,2,3,4\})^{Aut(G)}$ together with the partial ordering $\leq_{Aut(G)}$ results in the space shown in Figure 6.2. It can be seen that, e.g., the partitions $P = \{\{1,3,4\},\{2\}\}$ and $Q = \{\{2,3,4\},\{1\}\}$ lie on the same orbit because they are equivalent. Certainly, P is unstable, as $|P^{Aut(G)}| = 2 > 1$ holds.

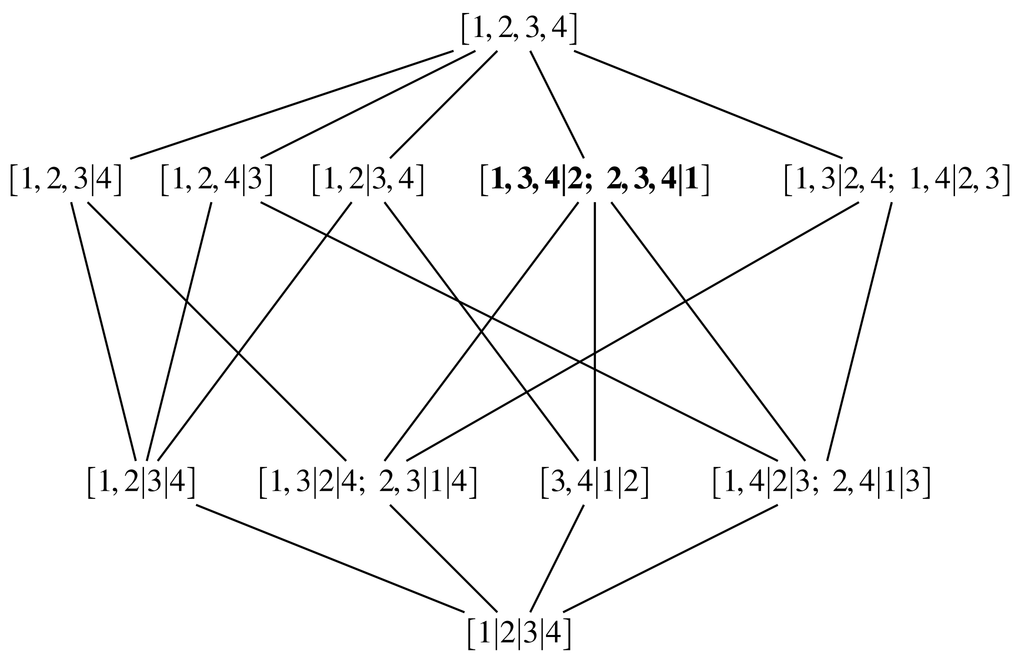


Figure 6.2: Diagram for $\boldsymbol{P}(\{1,2,3,4\})^{Aut(G)}$. Every node is an equivalence class that is induced by the automorphism group of the paw graph. The equivalence class that contains P and Q is set in bold font.

Each path through the space of partitions was called a join path in Figure 2.10, and we now see that a (non-trivial) automorphism group of a graph reduces the number of possible join paths and, therefore, the number of possible solutions for a hierarchical clustering algorithm. For the complete graph K_n, only one unique (of course only up to isomorphism) path exists through the space of equivalence classes. However, to possibly benefit from this observation in clustering, the equivalence classes have to be identified.

6.2 Equivalent Stability Definitions

In this section we present three equivalent definitions of partition stability (Ball and Geyer-Schulz, 2018c). Each of them (i) represents a different point of view on the problem and (ii) may be better practically applicable depending on the specific setting of the analysis. We use those definitions to present a conceptual algorithm that tests partition stability (Section 6.2.5).

Clustering partition stability is not a very tangible term and there are several definitions (probably not exhaustive) that could be thought of. Given a partition P . . .

. . . as the result of an algorithm (e.g. clustering), the algorithm's result on the *same* input should always be the same. This implies that the used method must be deterministic.

. . . as the result of an algorithm (e.g. clustering), the algorithm's result on *slightly perturbed* input should not differ significantly (Ben-Hur et al., 2001; Tibshirani and Walther, 2005; von Luxburg, 2010).

. . . being optimal concerning some (quality) criterion (e.g. modularity), the addition and/or removal of a few edges should change the criterion's value only marginally, so that the optimal partition stays the same (e.g. Karrer et al., 2008). This implies stability of the graph's topology concerning the given criterion against small changes.

. . . the class memberships should be unambiguous (Gfeller et al., 2005), which means there should exist a unique clustering partition and no nodes that cannot be assigned to exactly one cluster.

. . . (independent of its origin), no visible changes should occur due to symmetries of the graph (captured by its automorphism group).

The last point is the approach considered here. It is somewhat similar to the idea of Gfeller et al. (2005) who identify nodes that cannot be uniquely assigned to exactly one cluster of the partition. However, such ambiguities need not be caused by automorphisms of the graph.

6.2.1 Splitting the Automorphism Group

The automorphism group $Aut(G)$ is a finite set of permutation functions that can be split into two subsets, one of which may be empty.

Definition 8. *Let P be a partition of G. $Aut(G)$ is split into two subsets Π_{intra} and Π_{inter} with Π_{intra} containing all permutations for which $P^g = P$ and Π_{inter} containing all permutations for which $P^g \neq P$. The partition P is stable iff $\Pi_{inter} = \emptyset$.*

Definition 8 implies that for a stable partition P every automorphism of G must only act "locally", by mapping nodes onto each other that are within the same cluster anyway, or "globally", by mapping entire clusters onto each other. As we have seen, the automorphism groups can easily become very large so that testing stability for each of the permutations is very expensive to perform. However, generating the whole group is not necessary.

Theorem 11 (Partition stability by splitting the set of generators)**.** *Given a partition P of G and a set of generators S ($Aut(G) = \langle S \rangle$) that is split into $\tilde{\Pi}_{intra} = \{g \in S \mid P^g = P\}$ and $\tilde{\Pi}_{inter} = \{g \in S \mid P^g \neq P\}$. P is stable iff $\tilde{\Pi}_{inter} = \emptyset$.*

Proof. Let S be a set of generators, i.e. $\langle S \rangle = Aut(G)$. Furthermore, two subsets of $Aut(G)$ regarding a partition P as described in Definition 8 are given, namely Π_{intra} and $\Pi_{inter} \neq \emptyset$. Suppose $\tilde{\Pi}_{intra} = S$ and $\tilde{\Pi}_{inter} = \emptyset$. Then for some $g \in \Pi_{inter}$ (obviously $g \notin \tilde{\Pi}_{intra}$), there must exist a sequence of generators $(h_1, \ldots, h_k)$ for which $h_1 \circ \ldots \circ h_k = g$ holds and $h_i \in \tilde{\Pi}_{intra}$, $i = 1, \ldots, k$. But this is a contradiction, as each h_i individually either fixes all nodes within all $C_i \in P$ or maps all nodes from one cluster to another cluster. For arbitrary compositions of multiple h_i in the sequence above, this property is retained, as composing permutations means successively executing the mapping of each permutation in the given order. Associativity assures that it does not matter which permutation is used first, as long as the order does not change. So for $h_1 \circ \ldots \circ h_k = g \in \Pi_{inter}$, at least one $h_i \in \tilde{\Pi}_{inter}$ must exist, which contradicts $\tilde{\Pi}_{inter} = \emptyset$. □

Example 37. *Let $S = \{(1\ 2), (7\ 8)\}$ so that $\langle S \rangle = \{\mathbf{1}, (1\ 2), (7\ 8), (1\ 2)(7\ 8)\} = Aut(G)$. The partition $P = \{\{1,2,3\}, \{4,5,6\}, \{7,8\}\}$ is stable because $P^{(1\ 2)} = P$ as well as $P^{(7\ 8)} = P$. The partition $Q = \{\{1,3\}, \{2\}, \{4,5,6\}, \{7,8\}\}$ is, of course, unstable, as*

$$Q^{(1\ 2)} = \{\{2,3\}, \{1\}, \{4,5,6\}, \{7,8\}\} \neq Q. \tag{6.6}$$

The partitions of S and $Aut(G)$ regarding Q are $S = \tilde{\Pi}_{intra} \cup \tilde{\Pi}_{inter} = \{(7\ 8)\} \cup \{(1\ 2)\}$ and $Aut(G) = \Pi_{intra} \cup \Pi_{inter} = \{\mathbf{1}, (7\ 8)\} \cup \{(1\ 2), (1\ 2)(7\ 8)\}$, respectively.

It is obvious that $\langle \tilde{\Pi}_{intra} \rangle = \Pi_{intra} \leq Aut(G)$ is the subgroup that *stabilizes* the partition P.

6.2.2 Partition of Blocks

This approach builds on the block definition from Section 2.4.2.3. Recall that for a permutation group $Aut(G)$ a block is defined as a subset $C \subseteq V$ for which either $C \cap C^g = C$ or $C \cap C^g = \emptyset$ holds for all $g \in Aut(G)$. Informally, the permutations either map the block onto itself or all nodes to a different subset $C' \subseteq V$, which of course must also be a block.

Definition 9. *A partition P is stable*

1. *if every $C_i \in P$ is a block of $Aut(G)$ and*
2. *if for every $C_i \in P$ with $C_i \cap C_i^g = \emptyset$, $C_i^g \in P$ holds.*

Definition 9 is strongly coupled with the first approach in Section 6.2.1. It explicitly uses a general property of permutation groups (1.), but additionally requires (2.) to be true. The second condition is important as the example will show.

Example 38. *Let* $S = \{(1\ 4)(2\ 3), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\}$ *generate the permutation group*

$$\begin{aligned} Aut(G) = \{\mathbf{1},\ &(1\ 4)(2\ 3), (5\ 8)(6\ 7), (1\ 5\ 4\ 8)(2\ 6\ 3\ 7), (1\ 8\ 4\ 5)(2\ 7\ 3\ 6), \\ &(1\ 4)(2\ 3)(5\ 8)(6\ 7), (1\ 8)(2\ 7)(3\ 6)(4\ 5), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\}. \end{aligned} \tag{6.7}$$

The partition $P = \{\{1,2\},\{3,4\},\{5,6\},\{7,8\}\}$ *is stable because both conditions hold. Each* $g \in Aut(G)$ *either fixes* C_i *or maps it onto another* $C_j = C_i^g \in P$.

When analyzing $Q = \{\{1,2,3,4\},\{5,6,7\},\{8\}\}$ *we can see that it is not sufficient to only consider all* $g \in S$. $\{5,6,7\}^{(1\ 4)(2\ 3)} = \{5,6,7\}$ *fixes the cluster and* $\{5,6,7\}^{(1\ 5)(2\ 6)(3\ 7)(4\ 8)} \cap \{5,6,7\} = \{1,2,3\} \cap \{5,6,7\} = \emptyset$. *However,* $\{5,6,7\}$ *is not a block as, e.g., for* $(5\ 8)(6\ 7) \notin S$ *the equation* $\{5,6,7\}^{(5\ 8)(6\ 7)} \cap \{5,6,7\} = \{8,7,6\} \cap \{5,6,7\} = \{6,7\}$ *holds, but clearly* $\{6,7\} \notin \{\emptyset, \{5,6,7\}\}$.

Consider $R = \{\{1,2,3,4\},\{5,6\},\{7,8\}\}$. *For example* $\{5,6\}$ *is a block of* $Aut(G)$ *but* $\{5,6\}^{(1\ 5)(2\ 6)(3\ 7)(4\ 8)} = \{1,2\} \notin R$. *Therefore, R is unstable because condition (2.) of Definition 9 is not fulfilled and we see that condition (1.) alone is not sufficient.*

The second approach for defining the stability of a partition based on the graph's automorphism group shows the connection between a pure permutation group property and its application to data analysis. Compared to Definition 8, this comes at the cost of the necessity to completely enumerate the automorphism group. However, the decomposition method of MacArthur et al. (2008) could be incorporated if this approach is implemented.

6.2.3 Partition Refinement Lattice

The last approach involves the refinement lattice of all possible partitions of the node set V (see Section 2.3). The set of all partitions is denoted by $\boldsymbol{P}(V)$.

Recall from Section 2.3 that $\boldsymbol{P}(V)$ can be arranged as lattice given the partial ordering $\leq$. Informally, $P \leq Q$ is true if each cluster in P is a subset of a cluster in Q. The refinement lattice also illustrates the search space of a hierarchical clustering method. Each dendrogram represents a path through the lattice from level 1 to level n.

Definition 10. $P \in \boldsymbol{P}(V)$ *is stable iff* $\forall g \in Aut(G) : P^g \leq P$.

The point of view of Definition 10 shows that there must always exist a special finest stable partition for a given automorphism group (see also Ball and Geyer-Schulz, 2017). Naturally, this is the partition of all orbits O.

Theorem 12 (Stability of partitions coarser than the orbit partition)**.** *Given* $P \in \boldsymbol{P}(V)$ *and the orbit partition O regarding the automorphism group* $Aut(G)$*, P is stable if* $P \geq O$.

Proof. Clearly, O is stable by definition, as each subset/cluster contains exactly those nodes that can be mapped onto each other. This property is retained for any $C_{\cup} = C' \cup C''$, $C', C'' \in O$. The argument holds recursively for joins. Thus any partition P that can be created by successively joining clusters of O must be stable, too. However, this is exactly the refinement definition $P \geq O$. □

It is worth noting that the conclusion of Theorem 12 is not that *any* stable partition must be coarser than O. There can be numerous other stable partitions that are not coarser or even finer than the orbit partition. One trivial counterexample is the partition of singletons (each node forms one cluster) $P_{\perp}$, which is always stable.

Example 39. *Let us again look at* $Aut(G) = \{\mathbf{1}, (1\ 2), (7\ 8), (1\ 2)(7\ 8)\}$ *and the partition* $P = \{\{1,2,3\},\{4,5,6\},\{7,8\}\}$. *It is coarser than the orbit partition* $O = \{\{1,2\},\{3\},\{4\},\{5\},\{6\},\{7,8\}\}$ *(see Figure 6.3) and it is of course coarser than every* P^g, $g \in Aut(G)$. *Thus,* P *is stable.*

The partition $Q = \{\{1,3\},\{2\},\{4,5,6\},\{7,8\}\}$ *is unstable as, e.g.,* $Q^{(1\ 2)} \nleq Q$. *In Figure 6.3 one can see that* Q *and* $Q^{(1\ 2)}$ *are of course on the same level in the lattice, but none of them is coarser than* O.

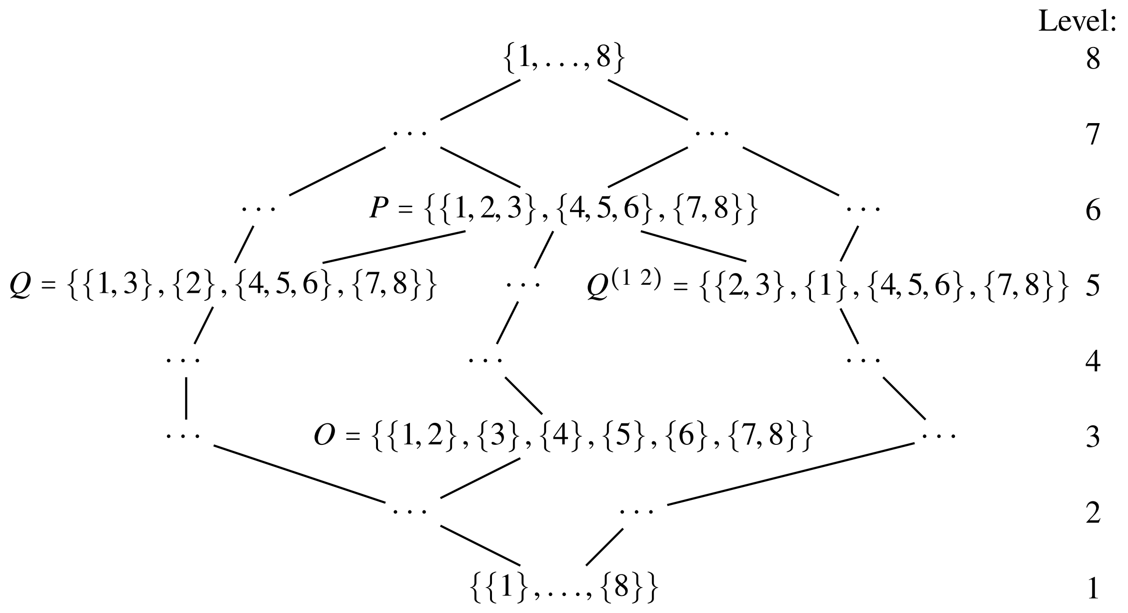

Level:
$\{1,\ldots,8\}$
8
...
7
...
$P = \{\{1,2,3\},\{4,5,6\},\{7,8\}\}$
6
$Q = \{\{1,3\},\{2\},\{4,5,6\},\{7,8\}\}$
$Q^{(1\ 2)} = \{\{2,3\},\{1\},\{4,5,6\},\{7,8\}\}$
5
4
$O = \{\{1,2\},\{3\},\{4\},\{5\},\{6\},\{7,8\}\}$
3
2
$\{\{1\},\ldots,\{8\}\}$
1

Figure 6.3: Extract of the partition refinement lattice for $V = \{1,\ldots,8\}$. Partitions become coarser on higher levels. O is the orbit partition, which is the special finest stable partition of $Aut(G) = \{\mathbf{1}, (1\ 2), (7\ 8), (1\ 2)(7\ 8)\}$.

6.2.4 Equivalence of the Definitions

The equivalence of Definitions 8–10 is quite obvious, however, we formally prove it in this section.

Theorem 13. *Definitions 8 and 9 are equivalent.*

Proof. Given a partition P of G and the corresponding automorphism group $Aut(G)$ that is divided into Π_{intra} and Π_{inter}.

1. a) If $\Pi_{inter} = \varnothing$ then $P = P^g$ for all $g \in Aut(G)$ holds. This can only be true if $\forall C \in P,\ \forall g \in Aut(G) : C^g \in P$ holds. $C^g \in P$ for some g and C means either $C^g = C$ or $C^g = C' \neq C$ with $C \cap C' = \varnothing$ and $C' \in P$. Therefore, each $C \in P$ is a block of $Aut(G)$ and $C^g \in P,\ \forall g \in Aut(G)$.

   b) If conversely all $C \in P$ are blocks of $Aut(G)$ and $\forall C \in P,\ \forall g \in Aut(G) : C^g \in P$. This directly implies $P^g = P,\ \forall g \in Aut(G)$ and, therefore, $\Pi_{inter} = \varnothing$.

2. a) If $\Pi_{inter} \neq \varnothing$ then $\exists g \in \Pi_{inter} : P \neq P^g$. This also means $\exists C \in P : C^g \notin P$, which violates the second condition of Definition 9.

   b) Consider there exists $C \in P$ that is not a block. This implies $C \cap C^g \notin \{\varnothing, C\}$ and consequently $C^g \notin P$ for some $g \in Aut(G)$. Therefore, $P \neq P^g$ and with this also $\Pi_{inter} \neq \varnothing$.

□

Theorem 14. *Definitions 8 and 10 are equivalent.*

Proof. Again, given a partition P of G and the corresponding automorphism group $Aut(G)$, which is divided into Π_{intra} and Π_{inter}.

1. a) From $P = P^g,\ g \in Aut(G)$ directly follows $P \geq P^g,\ g \in Aut(G)$.

   b) Given $P \geq P^g$ for all $g \in Aut(G)$. By definition, for each $C' \in P^g$ must exist $C \in P$ so that $C' \subseteq C$. As permutations g are bijective, $|P| = |P^g|$. By definition of the refinement lattice, there cannot be any neighbors on the same level. Therefore, the only possibility for $P \geq P^g$ to hold is $P = P^g$.

2. a) If $P \neq P^g$ for some $g \in \Pi_{inter}$, there exists $C^g \notin P$. Additionally, $\nexists C' \in P : C^g \subset C'$, as g is bijective. Therefore, $|C| = |C^g|$, and the only possibility for $C^g \subseteq C' \in P$ is $C^g = C'$, which is a contradiction.

   b) In the case $P \ngeq P^g$ for some $g \in Aut(G)$, a cluster $C^g \nsubseteq C' \in P$ must exist. Due to the bijectivity of g it can directly be deduced that $C^g \notin P$. Therefore, $P \neq P^g$ and $\Pi_{inter} \neq \varnothing$.

□

Lemma 1. *Definitions 9 and 10 are equivalent.*

Proof. Follows from the transitivity of the equivalence relation together with the equivalences of Definitions 8 and 9, and Definitions 8 and 10. □

For the three equivalent definitions no special properties of graphs (besides how $Aut(G)$ is defined) are used. We only require partitions of a finite set of length n and a permutation group that acts on this set and hence also on the partitions. The reason for this is that these definitions are completely generalizable for *any* partition of objects, provided the symmetry group is known. In Section 6.2.5 we exploit Definitions 8 and 10 for an implementation of a fast stability testing algorithm.

6.2.5 Fast Partition Stability Analysis

After the formal equivalence of the three provided characterizations of stability based on symmetry was proved, one might ask, which one should be preferred. Before arguing about consequences of the definitions, it is important to mention that all our considerations only become relevant if the data contains symmetry at all. For a trivial automorphism group (i.e. $|Aut(G)| = |\{\mathbf{1}\}| = 1$) every partition of nodes is stable.

From an analytic point of view, Definition 9 shows the connection to a permutation group property (blocks). The practical applicability of it is, nonetheless, limited because it requires to actually enumerate the whole group to check if clusters of a partition are blocks. However, from a didactic point of view, it shows that the transfer of results from the theory of permutation groups to data science often requires some modifications.

Definition 8 is of greater practical use. An algorithm that computes the automorphism group of a graph (like `nauty` or `saucy`) normally computes a set of generators of the group and returns them to the caller. Theorem 11 shows that the generators suffice to test stability by checking $P^g \geq P, \ \forall g \in S$.

Looking at Definition 10 also gives some insights what happens if symmetries exist, especially concerning the search space. The stability test procedure is similar to the one of Definition 8, as the proof of Theorem 14 shows the equality of the stability definitions $P = P^g$ and $P \geq P^g$.

The additional idea of having a special finest stable partition leads to Algorithm 1. The orbit partition O can be computed from the generators and the test involves only one comparison of partitions (instead of $|S|$).

The computation of the orbit partition O in Algorithm 1 can be memoized (e.g. Geyer-Schulz, 1989) to avoid repeated computation for stability tests of different partitions of the same graph. The actual implementation of this computation and the partition comparison test ($\geq$) strongly depends on the data structures that are used to represent partitions and permutations. `nauty` and `saucy` represent permutations using an explicit form[1], where each permutation g is an array of length n with $g[i] = i^g$. Partitions can also be represented by arrays of length n, where the entry at position i is some arbitrary *cluster id*. All nodes in the same cluster have the same *cluster id*. Another representation—which relates to the mathematical notation of a set of sets—is storing a partition as array of arrays.

[1] W. l. o. g., we assume $V = \{0, \dots, n-1\}$ to allow array indexing starting from 0, which normally is the standard way in programming languages.

Algorithm 1 Test partition stability based on the automorphism group of the graph

Require: P is a partition of G; S is a set of generators for $Aut(G)$, i.e. $\langle S \rangle = Aut(G)$

```
 1: function testStability($P$, $S$)
 2:     if $S = \varnothing$ then                 ▷ The group is trivial, the graph is asymmetric
 3:         return True
 4:     $O \leftarrow$ computeOrbitPartition($S$, $|P|$)
 5:     if $P \geq O$ then                        ▷ The partition is coarser than the orbit partition
 6:         return True
 7:     for $g \in S$ do
 8:         if $P \ngeq P^g$ then                 ▷ Instability detected; $P \geq P^g$ is equivalent to $P = P^g$
 9:             return False
10:     return True
```

We present efficient algorithms to compute the orbit partition from the set of generators S (Algorithm 2) and to check if one partition is coarser than another partition (Algorithm 3) in Appendix D. The complexity of Algorithm 1 mostly depends on the number of generators $|S|$. Finding upper bounds or the exact value for the size of a minimal generating set is an old but still open research issue except for special classes of groups (Lucchini et al., 2004). However, the results in Chapter 4 show that the algorithmic output of `saucy` is very promising in terms of the actual number of generators returned.

6.3 Impact of Specific Graph Structures on Modularity Optimal Partitions

In this section we want to show for two rather trivial symmetric motifs of graphs under which circumstances they affect modularity optimal partitions.

6.3.1 Star Graphs

A k-star is a special tree of depth one that consists of $k+1$ nodes and is created by connecting each node of the empty graph $\bar{K}_k$ to the $k+1$st node. This graph is equivalent to the complete bipartite graph $K_{1,k}$, and its automorphism group is of course isomorphic to the symmetric group S_k. MacArthur et al. (2008, p. 3529) state “that stars were the predominant symmetry structure present in all the networks”. Following this conclusion, we now want to examine if the existence of star subgraphs in a graph affects the modularity optimal partition. As the leaf nodes of this structure are connected to the rest of the graph through exactly one node (the root), it is sufficient to look at those k nodes and the cluster the root node is contained in (see Figure 6.4).

The partition is only stable if all these nodes remain singletons or become part of the root node’s cluster. Already, Ovelgönne et al. (2010, section 3.2) showed that nodes connected through only one edge to the graph can be preprocessed by merging them with their only

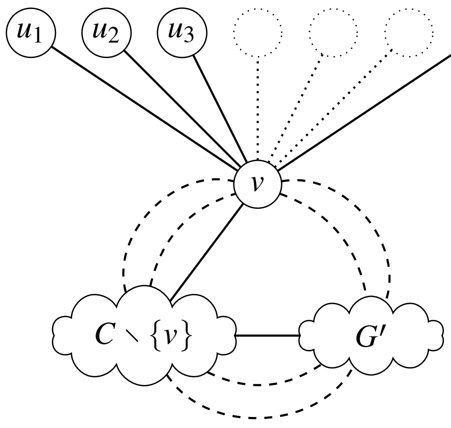


Figure 6.4: A star subgraph $K_{1,k}$ with nodes $v, u_1, \ldots, u_k \in V(G)$. v is part of cluster C, thus connected to it via at least one edge (possibly more). C is of course connected to G' (the "rest" of G) via at least one edge, too, and v is also possibly connected to other nodes in G'. The dashed edges represent those optional edges.

neighbor if modularity graph clustering is performed. This is because there always will be a positive contribution to the overall modularity.

We generalize and transfer this idea to our symmetry considerations and prove that merging the leaf nodes of a star (sub-)graph to their root's cluster is necessary to possibly yield maximum modularity. As a consequence, modularity optimal partitions of graphs that have stars as subgraphs will not be affected by the symmetries these stars "contribute" to the automorphism group (each $H \cong Aut(K_{1,k})$ is a subgroup of $Aut(G)$).

Theorem 15 (Effect of star subgraphs on partitions)**.** *The symmetry of star subgraphs does not affect a modularity optimal partition.*

Proof. The proof idea is to show that the contribution to modularity of the cluster C that contains the root node of the star graph is higher if all k leaf nodes are part of this cluster (say C'). Let m_C be the number of edges within C, k the number of edges that connect the leaf nodes to their root (therefore to C), and l the number of edges that connect C to the rest of the graph. The contribution to modularity of C is

$$\frac{m_C}{m} - \left(\frac{2m_C + k + l}{2m}\right)^2, \tag{6.8}$$

the contribution of C' is

$$\frac{m_{C'}}{m} - \left(\frac{2m_{C'} + l}{2m}\right)^2, \tag{6.9}$$

and clearly $m_{C'} = m_C + k$. So

$$\frac{m_C + k}{m} - \left(\frac{2(m_C + k) + l}{2m}\right)^2 > \frac{m_C}{m} - \left(\frac{2m_C + k + l}{2m}\right)^2 \tag{6.10}$$

must hold.

It follows:

$$\frac{m_C + k}{m} - \left(\frac{2(m_C + k) + l}{2m}\right)^2 > \frac{m_C}{m} - \left(\frac{2m_C + k + l}{2m}\right)^2 \tag{6.11}$$

$$\iff \quad \frac{k}{m} - \frac{1}{4m^2}(2m_C + 2k + l)^2 > -\frac{1}{4m^2}(2m_C + k + l)^2 \tag{6.12}$$

$$\iff \quad 4mk - ((2m_C + l) + 2k)^2 > -((2m_C + l) + k)^2 \tag{6.13}$$

$$\tag{6.14}$$

The term $(2m_C + l)^2$ appears on both sides of the inequality after expansion and can be canceled out:

$$4mk - 2(2m_C + l)2k - (2k)^2 > -2(2m_C + l)k - k^2 \tag{6.15}$$

$$\iff \quad 4mk - 4k(2m_C + l) - 4k^2 > -2k(2m_C + l) - k^2 \tag{6.16}$$

$$\iff \quad 4mk > 2k(2m_C + l) + 3k^2 \tag{6.17}$$

$$\iff \quad 4m > 4m_C + 2l + 3k \tag{6.18}$$

$$\iff \quad m > m_C + \frac{1}{2}l + \frac{3}{4}k \tag{6.19}$$

The last term 6.19 is always true. □

Lemma 2 (Modularity optimal partition of a star graph)**.** *The optimal partition of $K_{1,k}$ is the trivial partition with modularity $Q^* = 0$.*

Proof. Direct consequence of the proof of Theorem 15: $m > m_C + \frac{1}{2}l + \frac{3}{4}k$ with $m = k$ and $m_C = l = 0$, so $k > 0 + \frac{1}{2} \cdot 0 + \frac{3}{4}k$ holds and all $k + 1$ nodes must be put into the same cluster for maximal modularity. This results in the trivial partition, which always has modularity zero. □

This result is quite pleasing, as this means a large number of actually occurring symmetric structures in real-world graphs does not affect the optimality of modularity.

6.3.2 Complete Bipartite Subgraphs

A star $K_{1,k}$ can be generalized to an arbitrary complete bipartite graph $K_{j,k}$. We can even relax the exact condition of being bipartite (no adjacent nodes within the two disjoint node sets V_1 and V_2) by allowing edges between nodes of one of the two disjoint node sets (say V_1). The nodes in V_1 are connected to the rest of the graph and, as described, possibly interconnected to each other. The nodes in V_2 are not connected, and because of the completeness of connections between V_1 and V_2, clearly a group isomorphic to S_k acts on V_2.

Modeling the Problem in Terms of Modularity We restrict our examination to the case where each of the j nodes in V_1 is in a different cluster (see Figure 6.5). Moreover, no isomorphisms between the j clusters C_i exist and they must neither be merged nor split due to the resolution

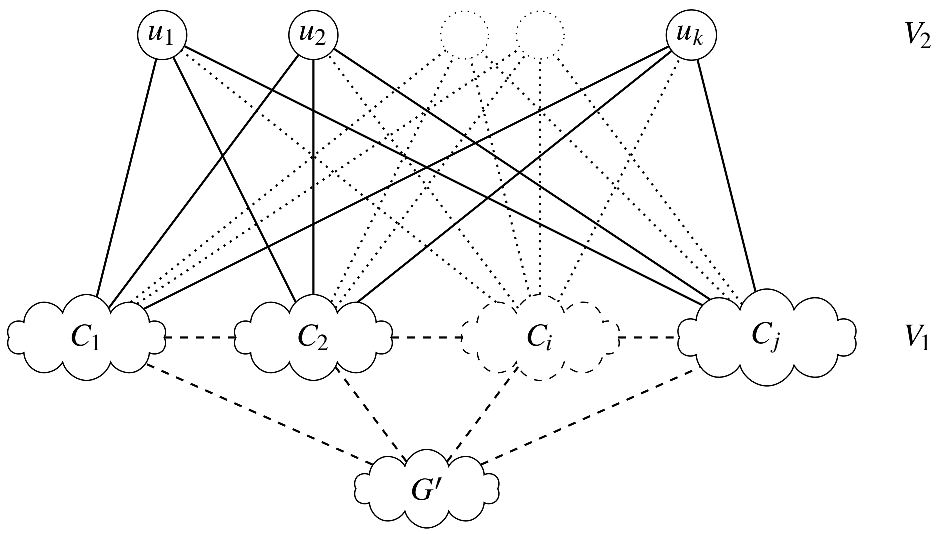


Figure 6.5: A complete bipartite subgraph $K_{j,k}$ with node sets $V_1, V_2 \subset V(G)$. Each $v_i \in V_1$ is part of cluster C_i. The C_i are connected to G' (not necessarily every C_i) in a way that G is a connected graph. Again, the dashed edges represent possibly more than one edge (or maybe even no edge). For example, C_1 could also be adjacent to C_j, but no edges are drawn to keep the figure clear. All nodes u_i are obviously on the same orbit on which a group isomorphic to S_k acts.

limit of modularity (Fortunato and Barthélemy, 2007). The contribution to modularity of each C_i is influenced by the number of intra-cluster-edges m_i, the fraction λ_i of the k nodes from V_2 that are also part of C_i, and the number of edges l_i that connect C_i to the rest of the graph. This results for each C_i, $i = 1, \ldots, j$ in

$$e_{ii} - a_i^2 = \frac{\overbrace{m_i + \lambda_i k}^{T_1}}{m} - \left(\frac{2\overbrace{(m_i + \lambda_i k)}^{2 \cdot T_1} + \overbrace{\lambda_i k(j-1)}^{T_2} + \overbrace{(1-\lambda_i)k}^{T_3} + \overbrace{l_i}^{T_4}}{2m} \right)^2 \tag{6.20}$$

with m the total number of edges of the graph, as usual. Remember that each edge in the term for a_i is counted twice (see Section 2.5.1). Let us explain the different parts T_1–T_4:

$\boldsymbol{T_1 = m_i + \lambda_i k}$ This is the number of intra-cluster-edges. m_i are all edges within cluster C_i and $\lambda_i k$ are the additional edges by adding λ_i of the k nodes from V_2.

$\boldsymbol{T_2 = \lambda_i k(j-1)}$ The number of edges between those nodes $u \in C_i$ to the remaining C_l $(l \neq i)$. Each node $u \in V_2$ has j neighbors. Therefore, each $u \in C_i$ has $j-1$ connections to all other clusters C_l $(l \neq i)$ and $\lambda_i k$ is the number of nodes of V_2 that are part of C_i.

$\boldsymbol{T_3 = (1-\lambda_i)k}$ This is the number of remaining connections to those nodes $u \notin C_i$.

$\boldsymbol{T_4 = l_i}$ These are the edges that are incident with C_i and either the rest of the graph G' or any of the other C_l $(l \neq i)$. Those edges are not part of the complete bipartite subgraph.

The term $2T_1 + T_2 + T_3 + T_4$ is, in accordance to the definition of a_i, the sum of all edges that are incident with a cluster C_i. However, the definition in Equation 6.20 is only an approximation because the $\lambda_i \in [0, 1]$ are not restricted in a way that $\lambda_i k \in \mathbb{N}$ holds.

To check if all the $j \cdot k$ edges of the complete bipartite graph $K_{j,k}$ are correctly assigned to the j clusters in dependence of the λ_i, we sum over the edges between the clusters C_i and the nodes $u \in V_2$. The edges from one cluster C_i to all the nodes of V_2 are expressed by the term

$$2T_1 + T_2 + T_3 + T_4 - 2m_i - l_i = 2\lambda_i k + \lambda_i k(j-1) + (1-\lambda_i)k. \tag{6.21}$$

If $\lambda_i = 0$, the term in Equation 6.21 yields k, which are all the edges from C_i to all $k = |V_2|$ nodes. In contrast, if $\lambda_i = 1$, Equation 6.21 yields $2k + k(j-1)$, which are twice the k edges from C_i to every $u \in V_2$ plus the $j-1$ edges from every $u \in V_2$ to all other clusters. Eventually, this results in

$$\begin{aligned}
\sum_{i=1}^{j} (2\lambda_i k + \lambda_i k(j-1) + (1-\lambda_i)k) &= k \sum_{i=1}^{j} (2\lambda_i + \lambda_i(j-1) + 1 - \lambda_i) \\
&= k \sum_{i=1}^{j} (\lambda_i + \lambda_i(j-1) + 1) \\
&= k \sum_{i=1}^{j} (\lambda_i j + 1) \\
&= k \left(\sum_{i=1}^{j} \lambda_i j + \sum_{i=1}^{j} 1 \right) \\
&= k\,(j + j) = 2kj
\end{aligned} \tag{6.22}$$

as of course $\sum_i \lambda_i = 1$. Again, each edge is counted twice and we see that the sum above gives the correct result.

The per-cluster modularity from Equation 6.20 can be simplified to

$$e_{ii} - a_i^2 = \frac{m_i + \lambda_i k}{m} - \left(\frac{2m_i + \lambda_i jk + k + l_i}{2m} \right)^2. \tag{6.23}$$

The term $\lambda_i jk + k$ describes all edges between C_i and the $u \in V_2$: On the one hand, there are always k edges from C_i to all $u_1, \ldots, u_k$, independent of the actual number of these nodes that are part of C_i. On the other hand, there is a fraction λ_i of all jk edges from V_2 to V_1 that are part of C_i.

Next, we want to check under which conditions the partition of the graph that contains the described subgraph construction will be stable. Again, this means all k nodes must be part of one of the j clusters. The modularity ratio of the j clusters together with the k nodes connected to them is

$$\begin{aligned} q(\lambda) = q(\lambda_1, \dots, \lambda_j) &:= \sum_{i=1}^{j} \left(\frac{m_i + \lambda_i k}{m} - \left(\frac{2m_i + \lambda_i jk + k + l_i}{2m} \right)^2 \right) \\ &= \sum_{i=1}^{j} \frac{m_i}{m} + \sum_{i=1}^{j} \frac{\lambda_i k}{m} - \sum_{i=1}^{j} \left(\frac{2m_i + \lambda_i jk + k + l_i}{2m} \right)^2 \\ &= \sum_{i=1}^{j} \frac{m_i}{m} + \frac{k}{m} - \frac{1}{4m^2} \sum_{i=1}^{j} (2m_i + \lambda_i jk + k + l_i)^2 . \end{aligned} \tag{6.24}$$

Formulation of the Optimization Problem This leads to the following (continuous) optimization problem:

$$\begin{aligned} &\max q(\boldsymbol{\lambda}) = \min -q(\boldsymbol{\lambda}) \\ \text{s. t.} \quad &\sum_{i=1}^{j} \lambda_i - 1 = 0 \\ &-\lambda_i \leq 0 \quad \forall i = 1, \dots, j \end{aligned} \tag{6.25}$$

which can be solved using a system of Karush-Kuhn-Tucker conditions (Karush, 2014; Kuhn and Tucker, 1951, and Appendix E). The Lagrangian[2] for the problem above is

$$L(\boldsymbol{\lambda}, u, v) = -q(\boldsymbol{\lambda}) + \sum_{i=1}^{j} u_i(-\lambda_i) + v \left(\sum_{i=1}^{j} \lambda_i - 1 \right) \tag{6.26}$$

and it is sufficient to solve

$$\begin{aligned} \nabla L(\boldsymbol{\lambda}, u, v) = \nabla(-q(\boldsymbol{\lambda})) + \sum_{i=1}^{j} u_i \nabla(-\lambda_i) + v \nabla \left(\sum_{i=1}^{j} \lambda_i - 1 \right) &= 0 \\ u_i &\geq 0 \quad i \in I(\boldsymbol{\lambda}) \\ -\lambda_i &= 0 \quad i \in I(\boldsymbol{\lambda}) \\ -\lambda_i &< 0 \quad i \notin I(\boldsymbol{\lambda}) \\ \sum_{i=1}^{j} \lambda_i - 1 &= 0 \end{aligned} \tag{6.27}$$

with $\nabla f(x_1, x_2, \dots)$ denoting the vector of partial derivatives

$$\nabla f(x_1, x_2, \dots) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \end{pmatrix} \tag{6.28}$$

[2] The definition is for minimization; therefore, we take $-q(\boldsymbol{\lambda})$, as we have a maximization problem.

and $I(\boldsymbol{\lambda})$ being the set of indices of active constraints. Computing $\nabla L(\boldsymbol{\lambda}, u, v)$ results in

$$\nabla L(\boldsymbol{\lambda}, u, v) = \begin{pmatrix} \frac{\partial L}{\partial \lambda_1} \\ \vdots \\ \frac{\partial L}{\partial \lambda_j} \end{pmatrix} = \frac{jk}{2m^2} \begin{pmatrix} 2m_1 + \lambda_1 jk + k + l_1 \\ \vdots \\ 2m_j + \lambda_j jk + k + l_j \end{pmatrix} + \begin{pmatrix} -u_1 \\ \vdots \\ -u_j \end{pmatrix} + v \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}. \tag{6.29}$$

Clearly, all (in-)equality constraints of 6.27 are linear (thus convex) and also $-q(\lambda)$ is convex, as using the C^2-characterization of convexity (e.g. Stein, 2012) shows: The Hessian is

$$D^2(-q(\lambda)) = \begin{pmatrix} \frac{j^2k^2}{2m^2} & 0 & & \cdots & 0 \\ 0 & \frac{j^2k^2}{2m^2} & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 0 & \frac{j^2k^2}{2m^2} \end{pmatrix} = \frac{j^2k^2}{2m^2}\mathbf{1}, \tag{6.30}$$

which is clearly a positive definite matrix. Due to the linearity of all constraints, the linearity constraint qualification holds as well. Therefore, the problem in 6.25 is a convex optimization problem, which allows us to state that each local optimum is also a global optimum.

This, however, does not imply that only one unique solution can exist. We have restricted that the C_i are not isomorphic, but there could exist weakly isomorphic clusters in terms of the definition of Section 5.3. In that case, the contribution to modularity of the weakly isomorphic C_i (without taking the $u \in V_2$ into account) is the same, and, therefore, the assignment of the nodes u is not unique.

Solving the Optimization Problem Instead of solving 6.27 in general, we are interested in the case $\lambda_{i'} = 1$ for some cluster $C_{i'}$ and of course $\lambda_i = 0,\ \forall i \neq i'$. The labels for the j clusters are arbitrary, so relabeling is possible. With the assumption $\lambda_j = 1$ we have $I(\boldsymbol{\lambda}) = \{1, \ldots, j-1\}$. Simplifying 6.29 together with 6.27 results in

$$\begin{aligned} \frac{jk}{2m^2} \begin{pmatrix} 2m_1 + k + l_1 \\ \vdots \\ 2m_{j-1} + k + l_{j-1} \\ 2m_j + jk + k + l_j \end{pmatrix} + v \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix} &= \begin{pmatrix} u_1 \\ \vdots \\ u_{j-1} \\ 0 \end{pmatrix} \\ u_i \geq 0 \quad & i \in I(\boldsymbol{\lambda}) \\ -\lambda_i = 0 \quad & i \in I(\boldsymbol{\lambda}) \\ -1 = -\lambda_j < 0 & \\ \sum_{i=1}^{j} \lambda_i - 1 = \sum_{i=1}^{j-1} \lambda_i + \lambda_j - 1 = 0 & \end{aligned} \tag{6.31}$$

which leads to

$$\frac{jk}{2m^2}\begin{pmatrix}2m_1+k+l_1\\ \vdots \\ 2m_{j-1}+k+l_{j-1}\end{pmatrix} - \frac{jk}{2m^2}(2m_j+jk+k+l_j)\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix} \geq 0 \tag{6.32}$$

$$\iff \begin{pmatrix}2(m_1-m_j)+(l_1-l_j)-jk\\ \vdots \\ 2(m_{j-1}-m_j)+(l_{j-1}-l_j)-jk\end{pmatrix} \geq 0. \tag{6.33}$$

The solution in inequality 6.33 can be interpreted as follows:

- Each cluster but C_j, which contains all k additional nodes of the graph motif, must ideally have many more intra-cluster-edges as well as inter-cluster-edges (to exceed jk) than C_j.
- With increasing j and/or k, the ratio must increase, too.
- At the same time, the amount of inter-cluster-edges normally must be reasonably smaller than the intra-cluster edges, as otherwise the partition would not have been modularity optimal. This is a direct consequence of the above assumption that the j clusters are neither merged nor split.

This means $j-1$ clusters need to be of rather balanced size and the j-th cluster is reasonably smaller so that assigning the k additional nodes to it will result in an overall more balanced partition. This is exactly the behavior that is described by the resolution limit of modularity (Fortunato and Barthélemy, 2007). Inequality 6.33 shows that multiple optimal solutions cannot exist under the made assumptions, as otherwise a cluster C_i with $m_i = m_j$ and $l_i = l_j$ must exist, which contradicts $2(m_i - m_j) + (l_i - l_j) - jk = -jk \geq 0$.

Examples

Example 40. *Let $j = k = 2$ and $m_1 = 10$, $m_2 = 8$, and $l_1 = l_2 = 1$. As $m_1 > m_2$ we assume $\lambda_2 = 1$ and therefore*

$$2(m_1-m_2)+(l_1-l_2) = 2(10-8)+(1-1) = 2\cdot 2+0 \geq jk = 2\cdot 2 \tag{6.34}$$

is actually true. If, for instance, if $C_1 \cup C_2 \cup \{u_1, u_2\} = V$ we have $m = 10 + 8 + 1 + 4 = 23$ and

$$\begin{aligned} Q &= \frac{20}{23} - \left(\frac{2\cdot 10+4\lambda_1+2+1}{46}\right)^2 - \left(\frac{2\cdot 8+4\lambda_2+2+1}{46}\right)^2 \\ &= \frac{20}{23} - \left(\frac{23+4\lambda_1}{46}\right)^2 - \left(\frac{19+4\lambda_2}{46}\right)^2. \end{aligned} \tag{6.35}$$

Putting the two nodes u_1, u_2 into C_2 ($\lambda_2 = 1$) yields $Q_{\lambda_2=1} = \frac{20}{23} - \left(\frac{23}{46}\right)^2 - \left(\frac{23}{46}\right)^2 \approx 0.3696$. This is the maximum, as the clusters are of equal size regarding the intra-cluster-edges.

Example 41. *Modifying Example 40 by setting $k = 4$ we get*

$$2(m_1 - m_2) + (l_1 - l_2) = 2(10 - 8) + (1 - 1) = 2 \cdot 2 + 0 \geq jk = 2 \cdot 4 \quad \lightning \tag{6.36}$$

and again for $C_1 \cup C_2 \cup \{u_1, \ldots, u_4\} = V$ we have $m = 10 + 8 + 1 + 8 = 27$ with

$$\begin{aligned} Q &= \frac{22}{27} - \left(\frac{2 \cdot 10 + 8\lambda_1 + 4 + 1}{54}\right)^2 - \left(\frac{2 \cdot 8 + 8\lambda_2 + 4 + 1}{54}\right)^2 \\ &= \frac{22}{27} - \left(\frac{25 + 8\lambda_1}{54}\right)^2 - \left(\frac{21 + 8\lambda_2}{54}\right)^2 . \end{aligned} \tag{6.37}$$

The modularity is maximal for $\lambda_1 = \frac{1}{4}$, as, again, edges are distributed equally among both clusters: $Q_{\lambda_1 = \frac{1}{4}} = \frac{22}{27} - \left(\frac{25+8\cdot\frac{1}{4}}{54}\right)^2 - \left(\frac{21+8\cdot\frac{3}{4}}{54}\right)^2 = \frac{22}{27} - 2\left(\frac{27}{54}\right)^2 \approx 0.3148$. However, the partition is unstable as one of the four equivalent nodes $\{u_1, \ldots, u_4\}$ is part of one cluster and the other three nodes are part of the other cluster.

Example 42. *For the special case $j = 1$ we have the same situation as for star graphs in Section 6.3.1 and from Equation 6.23 follows*

$$e_{ii} - a_i^2 = \frac{m_i + 1 \cdot k}{m} - \left(\frac{2m_i + 1 \cdot 1 \cdot k + k + l_i}{2m}\right)^2 = \frac{m_C + k}{m} - \left(\frac{2m_C + 2k + l}{2m}\right)^2, \tag{6.38}$$

which is the same as in Equation 6.9.

Example 43. *Another example is based on the well known Karate network (Zachary, 1977). The graph is shown in Figure 6.6 and the clusters of the modularity optimal partition are distinguished by four different colors. Additionally, the four non-trivial orbits of the automorphism group of the Karate network are emphasized by different node shapes (other than circles). The partition is stable, as none of the orbits crosses the cluster boundaries.*

However, due to a slight misclassification, the partition with $C_1' := C_1 \cup \{0, 11, 17\}$ and $C_2' := C_2 \setminus \{0, 11, 17\}$ is sometimes found as clustering solution. This leads to an unstable situation, where the two nodes 17 and 21 (which are on the same orbit) cross the cluster boundaries of C_1' and C_2'. According to Figure 6.5, we have $j = k = 2$ and the question is, for which $\lambda_1 = 1 - \lambda_2$ is the modularity maximal? There are $m_1 = 11$ ($m_2 = 11$) intra-cluster-edges and $l_1 = 9$ ($l_2 = 14$) inter-cluster-edges (see Figure 6.7). To test, the sum $\left(e_{11} - a_1^2\right) + \left(e_{22} - a_2^2\right)$ must be computed for every possible value of λ_1 and by using the parameters above (utilizing Equation 6.23). Of course, only $\lambda_1 \in \{0, 0.5, 1\}$ makes sense:

$$\underset{\lambda_1 \in \{0, 0.5, 1\}}{\arg\max} \left(e_{11} - a_1^2\right) + \left(e_{22} - a_2^2\right) = 1. \tag{6.39}$$

Testing the requirement of Equation 6.33 for $\lambda_1 = 1$ yields

$$2(11 - 11) + (14 - 9) - 2 \cdot 2 = 1 \geq 0 \tag{6.40}$$

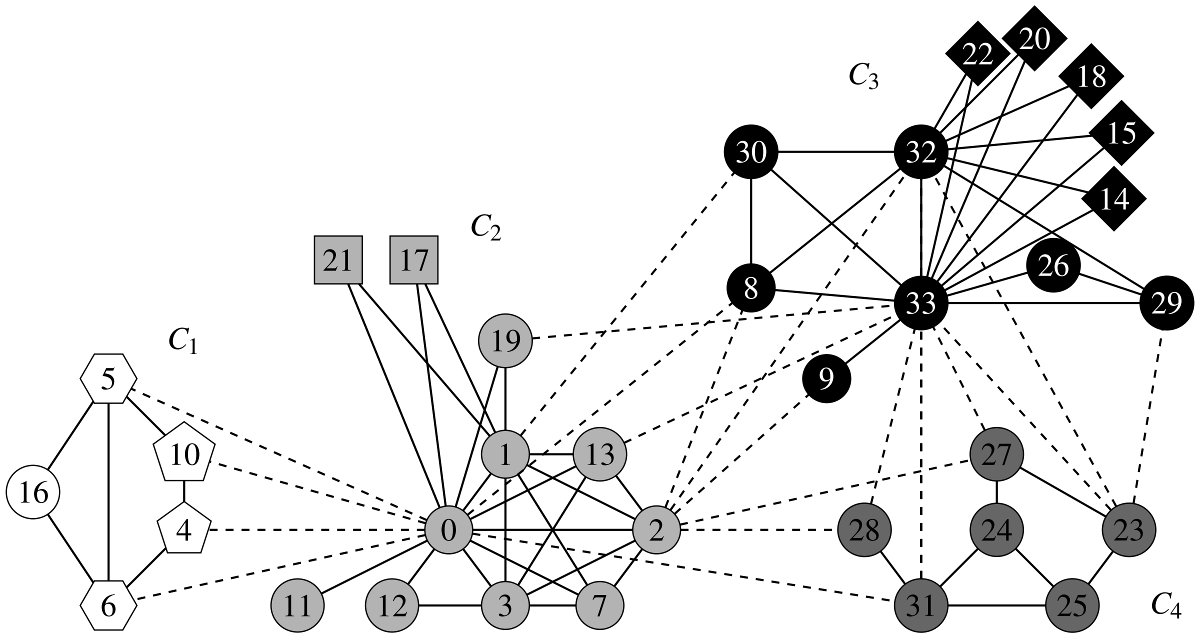


Figure 6.6: The modularity optimal partition ($Q \approx 0.4197$; determined by node colors, separated by dashed inter-cluster-edges) of the Karate network K. Non-trivial orbits are emphasized by node shapes other than circles. The nodes 17 and 21 (on which S_2 acts), as well as the nodes 14, 15, 18, 20, and 22 (on which S_5 acts), are, together with their neighbors, isomorphic to the complete bipartite subgraphs $K_{2,2}$ and $K_{2,5}$, respectively. However, both orbits do not cross cluster boundaries, therefore the proposed effect does not impact the partition. In fact, $Aut(K) \cong S_5 \times S_2^2$, where one of the S_2 groups is isomorphic to a group that acts on the nodes 4, 5, 6, and 10, thus $|Aut(K)| = 5! \cdot 2!^2 = 480$.

and for $\lambda_2 = 1$

$$2(11 - 11) + (9 - 15) - 2 \cdot 2 = -9 \ngeq 0. \tag{6.41}$$

This result seems to be a contradiction, because it suggests to put both nodes of the orbit into cluster C_1'. Certainly, there is a simple explanation for this issue: The suboptimal partition was found using the RG algorithm, which has a certain probability of misclassifications due to its randomized behavior. This means that an unstable solution can be found that is only locally but not globally optimal. A refinement strategy that tries to exchange nodes over cluster borders to achieve a higher modularity could resolve this issue (Ovelgönne, 2011, section 3.2.6).

Conclusion If complete bipartite subgraphs as shown in Figure 6.5 exist, the resulting partition will only be stable if one of the clusters to which all nodes u_i are connected to is reasonably smaller than the others and/or has reasonably less inter-cluster-edges. Otherwise, the nodes are spread over the different clusters, which leads to an instability.

6.4 Kolmogorov-Sinai Entropy

The Kolmogorov-Sinai Entropy is named after Andrey Nikolaevich Kolmogorov—the “grandfather” of modern probability theory—and Yakov Grigorevich Sinai—a student of Kolmogorov—who received the Abel Prize for his work in 2014 (Raussen and Skau, 2015). The

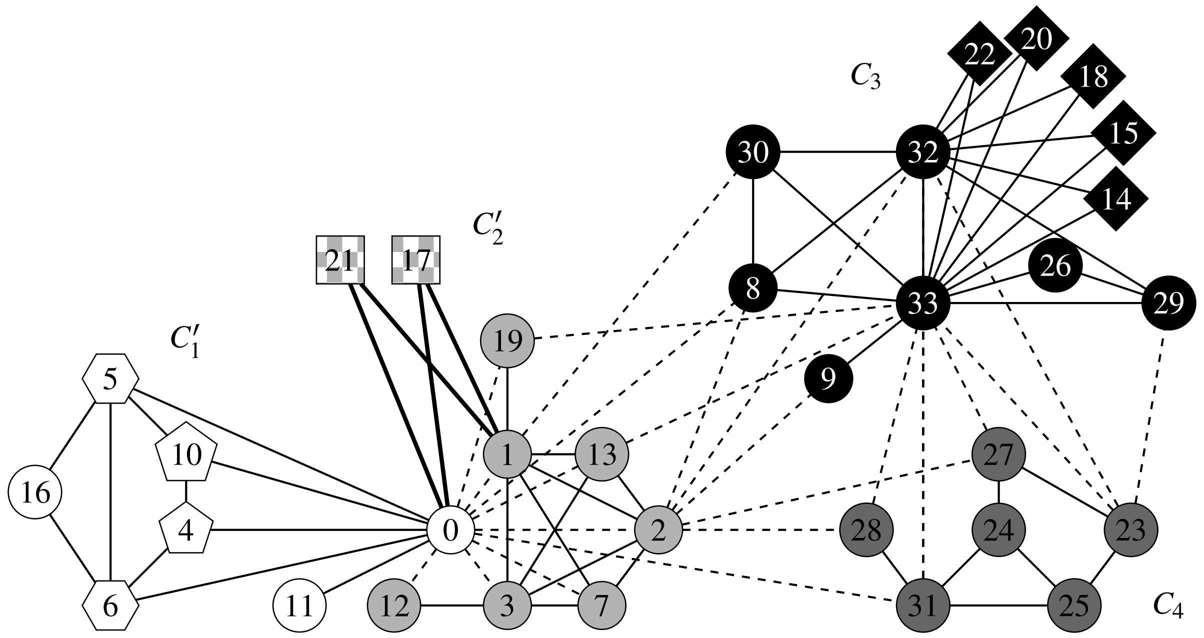


Figure 6.7: A suboptimal partition of the Karate graph where the orbit $\{17, 21\}$ crosses the boundaries of the clusters C'_1 and C'_2. Therefore, the partition is unstable.

Kolmogorov-Sinai Entropy is the application of the entropy concept, which is introduced in Section 2.6, to ergodic theory, which "is the study of the long-term behavior of systems preserving a certain form of energy" (Coudène, 2016, p. 3). We abbreviate the Kolmogorov-Sinai Entropy by KSE.

6.4.1 Dynamical Systems

A dynamical system consists of some state-space X together with a state transition function $T : X \to X$. Given some state $x(t) \in X$ at some point in time t, $T(x(t)) = x(t+1)$ is the state in the next period. The point in time is arbitrary and only needed for descriptive reasons. The next state is solely defined by the previous one, independent of t.

Given an additional sigma-algebra $\mathcal{X}$ of X and a measure μ that is invariant under T, the tuple $(X, \mathcal{X}, \mu, T)$ is called a *measure-preserving dynamical system*. Measure-preserving means that for each measurable $Y \subseteq X$ the equality $\mu(T^{-1}(Y)) = \mu(Y)$ must hold.

Example 44. *This example is from Einsiedler and Ward (2011, pp. 14 ff.). It is widely used throughout the literature. The space X is the half-closed interval $[0, 1)$ and the subsets Y are the intervals $[a, b) \subseteq [0, 1)$, $a \leq b$. The measure μ is the so called Lebesgue measure, which is simply defined as the length $\mu([a, b)) = b - a$ and $\mu(\bigcap_i Y_i) = \sum_i \mu(Y_i)$ for all pairwise disjoint Y_i (i.e. $\sup Y_i \leq \inf Y_j$ or $\sup Y_j \leq \inf Y_i$ for $i \neq j$). A measure preserving transformation is $T : [0, 1) \to [0, 1)$, $x \mapsto 2x \mod 1$. Its inverse T^{-1} is clearly determined by the pre-image $\{\frac{x}{2}, \frac{x}{2} + \frac{1}{2}\}$, which corresponds to the union of point-intervals $[\frac{x}{2}, \frac{x}{2}) \cup [\frac{x}{2} + \frac{1}{2}, \frac{x}{2} + \frac{1}{2})$.*

For an interval $Y = [a, b)$ this extends to $T^{-1}(Y) = [\frac{a}{2} + \frac{b}{2}) \cup [\frac{a}{2} + \frac{1}{2}, \frac{b}{2} + \frac{1}{2})$. Therefore,

$$\begin{aligned}
\mu(T^{-1}([a,b))) &= \mu([\frac{a}{2} + \frac{b}{2}) \cup [\frac{a}{2} + \frac{1}{2}, \frac{b}{2} + \frac{1}{2})) \\
&= \mu([\frac{a}{2} + \frac{b}{2})) + \mu([\frac{a}{2} + \frac{1}{2}, \frac{b}{2} + \frac{1}{2})) \\
&= \left(\frac{b}{2} - \frac{a}{2}\right) + \left(\left(\frac{b}{2} + \frac{1}{2}\right) - \left(\frac{a}{2} + \frac{1}{2}\right)\right) \\
&= b - a = \mu([a,b))
\end{aligned} \tag{6.42}$$

shows that T is measure preserving.

In general, it is important to require $\mu(T^{-1}(x)) = \mu(x)$, as $T(x)$ needs not be defined.

Example 45. *Let the situation be as in Example 44. $\mu(T([0.25, 0.5))) = \mu([0.5, 0)) = \mu(\varnothing) = 0 \neq 0.25 = \mu([0.25, 0.5))$ is a counterexample that shows why the inverse is used in the definition of a measure preserving transformation. The shown problem relates to the fact that the given transformation T is surjective, as, e.g., $T(0) = T(0.5) = 0$.*

6.4.2 Ergodic Theory and Entropy

According to Petersen (1983, p. 1), "[e]rgodic theory is the mathematical study of the long-term average behavior of systems". One could, e.g., think of a spinning top that looks like to spin forever. However, with ongoing time the friction will slow down the spinning motion and the system will eventually become unstable and the top will collapse.

The important pointwise ergodic theorem by Birkhoff (1931) states that for a measure preserving system for any $f \in L^1(X, \mathcal{X}, \mu)$ the following relation holds "almost everywhere":

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} f(T^i x) = \bar{f}(x). \tag{6.43}$$

This simply means that an average behavior of the system actually exists.

Entropy can be used to measure the disorder of the space that emerges from the transformation T. Petersen (1983, chapter 5) nicely motivates the topic, also beginning with entropy in physics as we did (Section 2.6). He then brings an example of an infinite symbol stream $\dots, x_i, x_{i+1}, \dots$ over some finite alphabet A of symbols and argues that—instead of measuring the occurrence probabilities of single symbols—one can group them into blocks. He defines the average information per symbol as

$$H_k := -\frac{1}{k} \sum_{B \in \mathbb{B}_k} \mathbb{P}(B) \operatorname{ld} \mathbb{P}(B), \tag{6.44}$$

with $\mathbb{B}_k$ the set of all blocks of length k given A.

Example 46. *Think of a classic typewriter, which has a finite amount of different letters (the alphabet), and instead of a sheet of paper, there is an infinite tape that is pulled through the machine (i.e. there are no line-breaks). Every time a key associated to a letter is hit, the letter* *is printed on the tape and it is pulled one step further. For simplicity, the amount of ink is also infinite. The person (or probably a cat) that hits the keys corresponds to the transformation T and we are now interested in the uncertainty of the overall system.*

Therefore, we begin looking at the tape at some arbitrary point in time and count the occurrences of the different letters in the order they appear on the tape. This is the procedure for $k = 1$. For $k > 1$, one has to count the occurrences of groups of length k.

Let us assume a very minimalist typewriter that has only the two letters $A = \{a, b\}$ and a part of the stream may look like

```
..baaababbabbbaabaaaabbababbabbababababababaababababaababbabbbabaaabaaab..
```

Using this cutout yields $\mathbb{P}(a) = \frac{35}{68}$ and $\mathbb{P}(b) = \frac{33}{68}$ for $k = 1$ and $\mathbb{P}(aa) = \frac{12}{67}$, $\mathbb{P}(ab) = \frac{23}{67}$, $\mathbb{P}(ba) = \frac{23}{67}$, and $\mathbb{P}(bb) = \frac{9}{67}$. The entropy is $H_1 = 0.9994$ and $H_2 = 0.9462$, respectively.

The question that now arises is: Which k should be chosen? There is no clear answer to it, but now the pointwise ergodic theorem from Equation 6.43 comes into play. Utilizing the theorem results in the definition of the "entropy of the source" (Petersen, 1983, p. 231):

$$h := \lim_{k\to\infty} -\frac{1}{k} \sum_{B\in\mathbb{B}_k} \mathbb{P}(B) \operatorname{ld} \mathbb{P}(B). \tag{6.45}$$

The ergodic theorem assures that this limit exists, but what does it mean? Taking the limit assures that any patterns that possibly occur in the symbol stream are captured. A pattern is just a fixed sequence of symbols that occurs more often than random and, therefore, represents a certain amount of order in the dynamical system.

Example 47. *Continuing Example 46, the two letters* `a` *and* `b` *occur nearly equiprobable. However, for $k = 2$, it appears that the two patterns* `ab` *and* `ba` *have a higher probability than* `aa` *and* `bb`*. Note that this rather short cutout of the whole stream does not suffice to yield good results for larger k, as the number of possible blocks increases exponentially in k ($|A|^k$, to be exact). It could even be possible that the occurrence of the two more likely patterns is only a random effect that results from the short total length of the cutout.*

Every deterministic system has an entropy of zero. An infinite symbol stream can be interpreted as a sequence of system states but also as the output of a system:

- A deterministic system either stops after a certain amount of time, which means that only the symbol representing this state is put into the stream, or
- it produces a finite output that "ends" at some time, or
- it produces an infinite output that follows a deterministic pattern (e.g. the states of a traffic light).

For the first two cases, the block size k can be chosen large enough so that it covers the complete output that does not represent the end-state. For the third case, Example 48 shows what happens.

Example 48. *Consider the symbol stream*

```
...aabbaabbaabbaabbaabbaabb...
```

which is infinite, but for a certain large enough k it will be found that this stream follows a deterministic pattern. Table 6.1 shows the probabilities for different blocks of a certain length k and it can be seen that with increasing k the number of different blocks is constant for k > 1.

k	Blocks B	Probability $\mathbb{P}(B)$
1	a, b	0.5
2	aa, ab, bb, ba	0.25
3	aab, abb, bba, baa	0.25
4	aabb, abba, bbaa, baab	0.25
$\vdots$	$\vdots$	0.25

Table 6.1: Blocks that occur with positive probability for certain k for the given symbol stream.

Thus, the entropy of the dynamical system that produces the symbol stream is

$$h = \lim_{k\to\infty} -\frac{1}{k}H_k = \lim_{k\to\infty} -\frac{1}{k}\cdot 4\left(\frac{1}{4}\,\mathrm{ld}\,\frac{1}{4}\right) = \lim_{k\to\infty}\frac{2}{k} = 0. \tag{6.46}$$

As we have seen in Section 2.6, the maximum entropy is bounded by the number of possible outcomes of a random variable. Here, this is tied to the number of possible blocks $|\mathbb{B}_k| = |A|^k$. This maximum value is of course reached for complete uncertainty, i.e. each block is seen with the same probability, independent of k.

Example 49. *We look again at the symbol stream from Example 46, but consider we can look at it for much longer, i.e. we can increase k without risk of running into sampling errors too early. The result is shown in Table 6.2.*

k	Block entropy H_k
1	1.0000
2	1.0000
3	0.9999
4	0.9999
5	0.9999
6	0.9998
7	0.9997
8	0.9992
9	0.9988
10	0.9979

Table 6.2: Observed entropy for a simulated symbol stream that randomly yields one of the two symbols for the first ten values of k. The cutout of the stream has length 2×10^{15}. It can be seen that all values are very close to the maximal value of 1 but slowly drop due to an increase in the sampling error.

As the stream is completely random, every block should ideally occur equiprobable. However, there is an increasing sampling error for larger k because of the fact that—although the sample *size is very large—some blocks occur less frequent than the theoretical consideration (Table 6.3) would suggest.*

k	Blocks B	Probability $\mathbb{P}(B)$	Block entropy H_k
1	a, b	0.5	1
2	aa, ab, bb, ba	0.25	1
3	aaa, aab, aba, abb, bbb, bba, bab,baa	0.125	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	...	2^{-k}	1

Table 6.3: Probabilities for all possible blocks to occur for given k if the symbol stream is completely random. The entropy is constantly 1.

Analyzing the random symbol stream theoretically yields the maximum entropy:

$$h = \lim_{k\to\infty} H_k = \lim_{k\to\infty} -\frac{1}{k} \sum_{B\in\mathbb{B}_k} \frac{1}{2^k} \operatorname{ld} \frac{1}{2^k} = \lim_{k\to\infty} -\frac{1}{k} \cdot 2^k \cdot \frac{1}{2^k} \operatorname{ld} \frac{1}{2^k} = \lim_{k\to\infty} \frac{1}{k} \operatorname{ld} 2^k = 1. \tag{6.47}$$

A similar analysis as in Example 49 is performed by Choe and Kim (2000) for the number π, which is assumed to be normal, i.e. each digit in the infinite decimal places occurs equiprobable. They report an approximate entropy of 1, which underlines the assumption ($\log_{10}$ was used by the authors). The actual proof that π is normal is still an open problem (e.g. Bailey and Borwein, 2016).

Frigg (2004) also provides the definitions we have made above, but, furthermore, motivates the case where the next symbol in the stream depends on the previous ones. He calls this the "history" of the system. However, for our purpose we can assume that all probabilities of the different symbols are independent. The author also states that every system having positive entropy is uncertain in the sense that there exists no point in time where the system's history is sufficient to completely predict the future behavior.

Sinai (1959) (Sinai, 2010, for an English translation) defines the entropy for Lebesgue spaces given an automorphism T. This space is just $(X, \mathcal{X}, \mu, T)$, which we have introduced earlier: A measure preserving system. To achieve this, he divides X into disjoint subsets A_i, which have the properties of a partition (see Section 2.3). The partition itself is $A = \{A_1, A_2, \ldots\}$. The entropy of such a partition is simply

$$h(A) := -\sum_i \mu(A_i) \log \mu(A_i), \tag{6.48}$$

and, in accordance to the entropy of a source,

$$h_T(A) := \lim_{k\to\infty} \frac{1}{k} h(\bigvee_k T^{k-1} A). \tag{6.49}$$

The unfamiliar definition that is used as a parameter for the entropy function is the repeated refinement of two partitions. It is defined as

$$A \vee B := \left\{ A_i \cap B_j \mid A_i \in A,\ B_j \in B,\ A_i \cap B_j \neq \varnothing \right\}. \tag{6.50}$$

According to the definition of a partial ordering of partitions in Section 2.3, $A \vee B$ is the coarsest partition that satisfies both, $A \vee B \leq A$ and $A \vee B \leq B$. $T^i A$ is simply the application of i times T on A. The repeated refinement is defined as

$$\bigvee_k T^{k-1} A := A \vee TA \vee T^2 A \vee \ldots \vee T^{k-1} A. \tag{6.51}$$

Please note that these definitions are equivalent to those in Section 2.5.2, where core group partitions are introduced. There, we used the more intuitive notation "$\bigwedge$" instead of "$\bigvee$"; the latter is widely-used in the literature on ergodic theory.

Definition 11 (Kolmogorov-Sinai Entropy). *The KSE is finally defined as*

$$h_T := \sup_A h_T(A). \tag{6.52}$$

The purpose of taking the supremum is to assure a finite partition is chosen that yields the maximum possible entropy as, e.g., $h_T(\bigvee_k T^{k-1}\{X\}) = h_T(\{X\}) = 0$ holds (independent of T).

6.4.3 Kolmogorov-Sinai Entropy of a Graph

In this thesis' setting, $(V, \mathcal{P}(V), \mu, T(Aut(G)))$ can be considered a measure-preserving dynamical system. V is the set of nodes of a graph G and $\mathcal{P}(V)$ is the power set of V. $Aut(G)$ is—as usual—the automorphism group of G. It is crucial to understand that, although in the literature on ergodic theory T is called an automorphism of the space, our case of application is discrete and not continuous. Therefore, the transformation that acts on V is not a single permutation but the group itself. Let $\sigma : \Omega \times Y \to \Omega$ be the function that selects an element $\omega \in \Omega$, given the random variable Y with the outcomes Ω. That means

$$\sigma(\Omega, Y) := \omega \in \Omega, \quad \text{if } Y = \omega \tag{6.53}$$

returns an arbitrary element from Ω depending on the outcome of Y. Hence, we define

$$T(Aut(G)) := \sigma(Aut(G), Y) \tag{6.54}$$

together with

$$\mathbb{P}(Y = g) = \frac{1}{|Aut(G)|}, \quad g \in Aut(G). \tag{6.55}$$

Therefore, each $T(Aut(G))$ is an arbitrary permutation of $Aut(G)$, and each element $g \in Aut(G)$ has the same probability to be chosen. This means in particular, e.g., $T(Aut(G))^2 = gh$ for two randomly chosen $g, h \in Aut(G)$.

The measure μ can be any measure that is unaffected by $Aut(G)$. If μ is a probability measure, $(X, \mathcal{X}, \mu)$ is a probability space. Here, μ is the probability $\mu(C) := \mathbb{P}(v \in C \mid v \in V)$. To achieve this kind of measurement, the different clusters must be identifiable. One possibility is to define partitions as ordered sets, where each element (a set that represents a cluster) gets assigned some number that allows a unique ordering, independent of the elements in the cluster:

$$P_{ord} := (\{\ldots\}_1, \{\ldots\}_2, \ldots, \{\ldots\}_k)\,. \tag{6.56}$$

The order is denoted by the subscripts of the different sets. This definition allows to distinguish clusters that contain the same elements, e.g., $P = \{\{1\},\{2\}\} = \{\{2\},\{1\}\}$ but $P_{ord} = (\{1\}_1, \{2\}_2) \neq (\{2\}_1, \{1\}_2)$. The actual ordering of sets does not matter, and, therefore, it is also not important of what happens to the ordering when two partitions are refined using "$\vee$". A second possibility to represent partitions with identifiable clusters is to use a (column) vector $\boldsymbol{P}$ of labels L, i.e. $\boldsymbol{P} \in L^n$. The entry at position $i = 1, \ldots, n$, determines the cluster id of which node i is part of. In accordance to the definition of P_{ord}, we choose $L = \{1, 2, \ldots, k\}$.

Clearly, defining μ as described is invariant regarding $Aut(G)$. Because permutations are bijective, using the transformations T^k instead of T^{-k} (as argued in Section 6.4.1) is equivalent.

Frigg (2004) shows the equivalence of KSE and "CTE" ("communication-theoretic entropy") by a construction that is relatively similar to what will follow. The idea of KSE is to successively refine the partition of the state-space and measure the cells of the partition for computing the entropy. The author compares this measurement to the (infinite) symbol stream by saying that continuously "reporting" the cell that contains the current system's state is the same as observing the symbol stream of elements from an alphabet. Hence, the cells A_i of the partition A correspond to the alphabet A, but with the difference that A is not constant for KSE. Forming the limit over the blocks of increasing length is equivalent to the repeated refinement of the state-space. This refinement can be interpreted as improving the measurement of the process by looking at it at a more detailed level. However, if a dynamical-system has positive KSE, there is no finite measurement that is so exact (in terms of uncovering all invisible information) that it can completely describe and predict the behavior of the system.

Before we define the KSE for graphs, there is a last important fact that must be pointed out. All the given definitions and explanations are based on *one* dynamical system, i.e. there exists exactly one orbit. Indeed, the automorphism group of a graph is not restricted to describe only one orbit, but it can have between one (transitive graph) and n (asymmetric graph) orbits: Some of the orbits may be trivial, others are not. Some of the orbits may be independent of others, whereas other orbits may depend on each other due to the same indecomposable permutation group that affects more than one orbit. For information on local and global symmetry, see Section 5.2.

Example 50. *The interplay of different orbits can be illustrated nicely by imagining a tabletop soccer. It consists of:*

- *A plane board that represents the soccer field (the state-space).*
- *The ball that is moved on the field, for simplicity we assume there are no holes that represent the goals.*
- *A couple of bars on which the "players" are mounted at some fixed position. All bars are installed in parallel across the shorter side of the field and can be moved to and fro a bit to move the players. Again, for simplicity, we omit the rotation of the bars that simulate the kicks.*

Within this state-space, the movements of the ball and each individual player represent different dynamical systems with different orbits. The movement of different bars is independent, however, the movement of the players mounted on the same bar are clearly not independent. These observations are similar to the considerations from Section 5.2: The automorphism group can be decomposed into support disjoint subgroups, but indecomposable subgroups can also possibly act on more than one orbit.

To formalize the entropy of multiple random variables, we must extend the definitions. Fortunately, there is already the concept of joint entropy that captures this case:

$$H(X,Y) = H(X) + H(Y \mid X) \tag{6.57}$$

where $H(Y \mid X)$ is the conditional entropy of Y given X (see Section 2.6). This can be generalized to

$$\begin{aligned} H(X_1, X_2, \ldots, X_k) = H(X_1) + H(X_2 \mid X_1) + H(X_3 \mid X_1, X_2) + \ldots \\ + H(X_k \mid X_1, X_2, \ldots, X_{k-1}). \end{aligned} \tag{6.58}$$

The question now is, what are the different X_i? As we stated above, a dynamical system is characterized by its orbit, hence each orbit of the graph is a separate dynamical system.

6.4.3.1 Entropy of a Graph Partition

Bringing together all these considerations, we can define the entropy of a graph G and a given partition P:

$$h_{Aut(G)}(P) := \sum_i h_{H_i}(P) = -\sum_i \sum_{\boldsymbol{l} \in L(P)^k} \mu_{H_i}(\boldsymbol{l}) \operatorname{ld} \mu_{H_i}(\boldsymbol{l}). \tag{6.59}$$

The outer sum goes over all normal subgroups H_i for which $Aut(G) = \prod_i H_i$ holds, and $\mu_{H_i}(\boldsymbol{l})$ is the probability that the label vector $\boldsymbol{l}$ (see below) is observed under the actions of H_i. This decomposition corresponds to the different independently symmetric areas of G we have presented in Section 5.2. This simplification is possible, as these areas imply also statistical independence so that all other orbits vanish in the conditional entropy. As a result, if (w.l.o.g.) all $X_1, X_2, \dots X_i$ are dependent but independent of all other $X_{i+1}, \dots, X_k$, and $X_{i+1}, \dots, X_{i'}$ are dependent as well but independent of all others, etc., then

$$\begin{aligned} H(X_1, X_2, \dots, X_k) &= H(X_1) + H(X_2 \mid X_1) + \dots + \\ &\quad H(X_i \mid X_1, X_2, \dots, X_{i-1}) \\ &\quad + H(X_{i+1}) + H(X_{i+2} \mid X_{i+1}) + \dots + \\ &\quad H(X_{i'} \mid X_{i+1}, \dots, X_{i'-1}) \\ &\quad + \dots \\ &= H(X_1, X_2, \dots, X_i) + H(X_{i+1}, \dots, X_{i'}) + \dots \end{aligned} \tag{6.60}$$

holds. Also, all trivial orbits of length one vanish, as they have entropy zero.

The inner sum in Equation 6.59 is over $\boldsymbol{l} \in L(P)^k$. L is the set of all cluster labels, as defined above, and we write $L(P)$ here to clarify that these are the labels of P. As $L(P)^k = L(P) \times \dots \times L(P)$ (k times), $\boldsymbol{l}$ is a vector of cluster labels; k is the number of (non-trivial) orbits of the current subgroup H_i. This means, instead of measuring the clusters separately, we measure an ensemble of clusters to capture the interdependence of orbits. The state of the dynamical system is, therefore, expressed by a vector of states of the different dependent subsystems. It is clear, for $k = 1$, this corresponds to the “usual” KSE definition. Coming back to the symbol stream analogy, this corresponds to have k symbol streams in parallel at which we look at the same time. Or, alternatively, this can be thought of as one symbol stream over the alphabet $L(P)^k$. Please note that summing over the different support disjoint subgroups is already a simplification of the situation as the following example will show.

Example 51. *Consider the graph G in Figure 6.8 and the partition illustrated in this picture.*

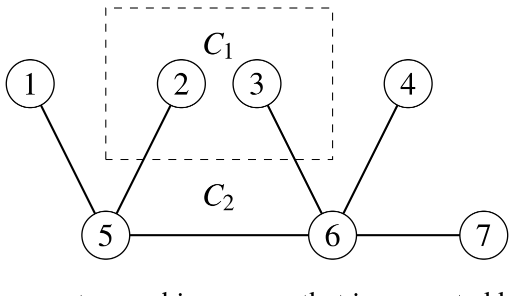


Figure 6.8: A graph G that has an automorphism group that is generated by $S = \{(1\ 2), (3\ 4)\}$. It has two non-trivial orbits and three trivial ones. The partition P is denoted by the dashed box: Cluster C_1 consists of the nodes within the box, C_2 of all nodes outside of it.

The full automorphism group is

$$Aut(G) = \{(), (1\ 2), (3\ 4), (1\ 2)(3\ 4)\}. \tag{6.61}$$

To compute $h_{Aut(G)}(P)$, we form the sum of the entropies per disjoint subgroup. These are $H_1 = \langle (1\ 2) \rangle$ and $H_2 = \langle (3\ 4) \rangle$ and both consist of only one orbit. Thus, given the alphabet of system states $L(P) = \{C_1, C_2\}$,

$$\begin{aligned} h_{Aut(G)}(P) &= -\big[\, (\mu_{H_1}(C_1) \operatorname{ld} \mu_{H_1}(C_1) + \mu_{H_1}(C_2) \operatorname{ld} \mu_{H_1}(C_2)) + \\ &\qquad (\mu_{H_2}(C_1) \operatorname{ld} \mu_{H_2}(C_1) + \mu_{H_2}(C_2) \operatorname{ld} \mu_{H_2}(C_2)) \,\big] \\ &= -\left[\left(\frac{1}{2} \operatorname{ld} \frac{1}{2} + \frac{1}{2} \operatorname{ld} \frac{1}{2}\right) + \left(\frac{1}{2} \operatorname{ld} \frac{1}{2} + \frac{1}{2} \operatorname{ld} \frac{1}{2}\right)\right] \\ &= -2 \cdot \operatorname{ld} \frac{1}{2} = 2. \end{aligned} \tag{6.62}$$

Alternatively, we could have computed $h_{Aut(G)}(P)$ without decomposing the group by assuming that all orbits are dependent of each other. Therefore, we must measure all $|L(P)|^5 = 32$ possible vectors of cluster labels in $L(P)^5$ (there are five orbits in total):

$$\begin{aligned} h_{Aut(G)}(P) = -\Bigg(& \mu(\begin{pmatrix} C_1 \\ C_1 \\ C_1 \\ C_1 \\ C_1 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_1 \\ C_1 \\ C_1 \\ C_1 \\ C_1 \end{pmatrix}) + \mu(\begin{pmatrix} C_1 \\ C_1 \\ C_1 \\ C_1 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_1 \\ C_1 \\ C_1 \\ C_1 \\ C_2 \end{pmatrix}) + \\ & \ldots + \\ & \mu(\begin{pmatrix} C_2 \\ C_2 \\ C_2 \\ C_2 \\ C_1 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_2 \\ C_2 \\ C_2 \\ C_2 \\ C_1 \end{pmatrix}) + \mu(\begin{pmatrix} C_2 \\ C_2 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_2 \\ C_2 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \Bigg). \end{aligned} \tag{6.63}$$

As all vectors but four have measure zero, Equation 6.63 can be simplified to

$$\begin{aligned} h_{Aut(G)}(P) = -\Bigg(& \mu(\begin{pmatrix} C_1 \\ C_1 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_1 \\ C_1 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) + \mu(\begin{pmatrix} C_1 \\ C_2 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_1 \\ C_2 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) + \\ & \mu(\begin{pmatrix} C_2 \\ C_1 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_2 \\ C_1 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) + \mu(\begin{pmatrix} C_2 \\ C_2 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_2 \\ C_2 \\ C_2 \\ C_2 \\ C_2 \end{pmatrix}) \Bigg), \end{aligned} \tag{6.64}$$

and, as the entry in the vectors for the trivial orbits is always the same in Equation 6.64, we can omit them from the computation:

$$\begin{aligned} h_{Aut(G)}(P) = &-\Big(\mu(\begin{pmatrix} C_1 \\ C_1 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_1 \\ C_1 \end{pmatrix}) + \mu(\begin{pmatrix} C_1 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_1 \\ C_2 \end{pmatrix}) + \\ &\mu(\begin{pmatrix} C_2 \\ C_1 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_2 \\ C_1 \end{pmatrix}) + \mu(\begin{pmatrix} C_2 \\ C_2 \end{pmatrix}) \operatorname{ld} \mu(\begin{pmatrix} C_2 \\ C_2 \end{pmatrix})\Big) \\ = &-4\left(\frac{1}{4} \operatorname{ld} \frac{1}{4}\right) = 2. \end{aligned} \tag{6.65}$$

We see, the results of Equations 6.62 and 6.65 are the same. Also, the one to one correspondence of the independence of H_1 and H_2 and the resulting symbol stream is visible: $Aut(G) = H_1 \times H_2$ means computing the cross product of all elements from the two groups and exactly the same happens for the clusters that have positive measure. Here, C_1 and C_2 have positive measure for H_1 and H_2, therefore, all four combinations of C_1 and C_2 have positive measure and it is just the product of measures, e.g.,

$$\mu(\begin{pmatrix} C_1 \\ C_2 \end{pmatrix}) = \mu(C_1) \cdot \mu(C_2). \tag{6.66}$$

Example 52 shows the importance of not considering dependent orbits as independent.

Example 52. *Given the tree graph T in Figure 6.9 and the shown partition.*

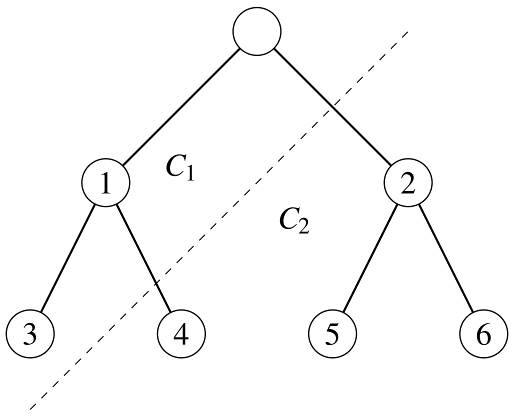


Figure 6.9: A binary tree graph T that has an automorphism group that is generated, e.g., by $S = \{(1\ 2)(3\ 5)(4\ 6), (3\ 4)\}$. It has three orbits, all nodes per level are on the same orbit. Clearly, the root node lies on a trivial orbit, the (inner) leaves lie on non-trivial orbits. The partition P is denoted by the dashed line.

The full automorphism group is

$$\begin{aligned} Aut(T) = \{&(), (3\ 4), (5\ 6), (3\ 4)(5\ 6), (1\ 2)(3\ 5)(4\ 6), \\ &(1\ 2)(3\ 6)(4\ 5), (1\ 2)(3\ 5\ 4\ 6), (1\ 2)(3\ 6\ 4\ 5)\}, \end{aligned} \tag{6.67}$$

and it is indecomposable.

If we would proceed as in Example 51 and consider the two non-trivial orbits as independent, the result would be

$$\begin{aligned} h'_{Aut(T)}(P) &= -\underbrace{\left(2 \cdot \frac{1}{2} \operatorname{ld} \frac{1}{2}\right)}_{Orbit\ \{1,2\}} - \underbrace{\left(\frac{1}{4} \operatorname{ld} \frac{1}{4} + \frac{3}{4} \operatorname{ld} \frac{3}{4}\right)}_{Orbit\ \{3,4,5,6\}} \\ &= 1 + \left(\frac{1}{2} - \frac{3}{4}(\operatorname{ld} 3 - \operatorname{ld} 4)\right) \\ &= 1 + \left(2 - \frac{3}{4} \operatorname{ld} 3\right) \approx 1.8113. \end{aligned} \tag{6.68}$$

However, the orbits are clearly not independent of each other. Given we have chosen the nodes 1 *and* 3 *as orbit representatives that we use for evaluation. Then, e.g., if the orbit* $\{1,2\}$ *is in state* C_2*, the orbit* $\{3,4,5,6\}$ *cannot be in* C_1 *at the same time, as there exists no permutation* $g \in Aut(T)$ *that maps node* 1 *onto node* 2 *(therefore state* C_1 *to state* C_2*) but fixes node* 3 *and, consequently, the state* C_1*. This results in the joint probabilities* $\mathbb{P}(C_1, C_1) = \frac{1}{4}$, $\mathbb{P}(C_1, C_2) = \frac{1}{4}$, $\mathbb{P}(C_2, C_1) = 0$, *and* $\mathbb{P}(C_2, C_2) = \frac{1}{2}$*. Using these values yields*

$$h_{Aut(T)}(P) = -\left(2 \cdot \frac{1}{4} \operatorname{ld} \frac{1}{4} + \frac{1}{2} \operatorname{ld} \frac{1}{2}\right) = -\left(-\frac{1}{2} \cdot 2 - \frac{1}{2}\right) = \frac{3}{2} \tag{6.69}$$

and clearly $h_{Aut(T)}(P) < h'_{Aut(T)}(P)$ *as a consequence of less "disorganization" of the graph.*

6.4.3.2 Entropy of the Graph

Until now we have defined the entropy of a graph for a given partition of the graph. In the KSE definition for dynamical systems, the partition of the space must be successively refined to get a more and more detailed picture how T acts on the state-space. The crucial part is to form the limit for $k \to \infty$, which means to infinitely refine the state-space. For the case of (finite) graphs, this is of course not possible, as the most refined partition (depending on the actual graph) is the singleton partition. However, even though the singleton partition will always result in the maximum possible entropy, this is not necessarily the only partition for which this is the case.

Several special partitions can be distinguished:

1. The finest non-trivial partition that has entropy zero, which is just the orbit partition O. Independent of how the different orbits are interrelated, every measurement of the cells/clusters of O will have measure zero. Moreover, every finer partition would mean to divide a non-trivial orbit into disjoint parts and for these parts, the measure must then be positive.

2. The coarsest non-trivial partition having maximum entropy, which is, informally spoken, the opposite of O. It combines all trivial orbits into one large cluster and all non-trivial orbits are split up into separate clusters. However, clusters that contain nodes from different orbits can be merged again. This partition is, therefore, not unique.

3. An invariant coarsest partition with maximum entropy that is defined as the coarsest partition above, but with the additional constraint that it is not affected by the automorphism group and, therefore, stable in terms of the definitions of Section 6.2.

Note that the third type of partition implies that there can exist stable partitions that, nonetheless, have positive KSE. Remember the situation where $\exists g \in Aut(G), C \in P : C^g \cap C = \varnothing$, i.e. all nodes are mapped into a different cluster. Due to the definition of partitions based on sets, nothing changes. The definition of KSE for partitions, however, requires the identifiability of clusters that eventually leads to a positive entropy, as the following example shows:

Example 53. *Reconsider the butterfly graph B (Figure 3.6) and the stable partition* $P = \{\{1,2\}, \{3\}, \{4,5\}\}$. *There exist automorphisms that map* $\{1,2\}$ *onto* $\{4,5\}$ *and vice versa. The entropy of the partition is* $h_{Aut(B)}(P) = 1$.

This finally brings us to the definition of the KSE of a graph, which is simply the standard definition applied to Equation 6.59:

$$h_{Aut(G)} := \sup_P h_{Aut(G)}(P). \tag{6.70}$$

Every partition P' that separates all nodes of each orbit into different clusters satisfies the supremum condition of Equation 6.70, e.g., $P' = P_\perp$.

When applying the definition from Equation 6.70 to two structurally different graphs with the same number of nodes, one can find a quite surprising result.

Example 54. *Let* K_n *and* C_n *be a complete graph and a cycle graph, respectively, of n nodes. Both are transitive, i.e. they have only one orbit and the only partition for which* $h_{Aut(G)} = h_{Aut(G)}(P)$ *is the singleton partition. As a consequence, independent of which orbit representatives are chosen,* $h_{Aut(G)} = \operatorname{ld} n$ *holds, as each state is equiprobable. Of course, for every other partition* P' *also* $h_{Aut(K_n)}(P') = h_{Aut(C_n)}(P')$ *holds.*

Have we done something wrong? The complexity of $Aut(K_n)$ is much higher than the one of $Aut(C_n)$ in terms of group order, why is this not reflected in the entropy?

The short answer is: No, the group order does not matter! The general purpose of entropy is to measure uncertainty; for a dynamical system this means, the uncertainty of the next state. Generally, the state-space is continuous and, therefore, there are infinitely many states, which we cannot enumerate, but we can successively refine a partition of the state-space and enumerate and measure its cells. In the limit, the system has positive KSE if, no matter how fine the partition is, there is an increasing number of cells with positive measure. The exact states are unknown and there exist theoretically infinitely many possibilities to reach one state from another. Frigg (2004, Figure 2) illustrates this fact in a clear manner: He shows a state-space $M \subset \mathbb{R}^2$ that is divided into 18 equally sized cells and draws a trajectory that crosses several of them. However, the state is not measured in every cell, i.e. what happens during the transition from one measured state to another is unknown (see Figure 6.10).

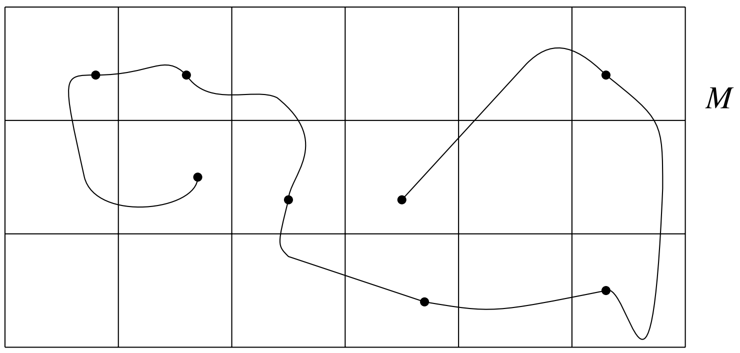


Figure 6.10: A state-space $M \subset \mathbb{R}^2$ (Frigg, 2004, Figure 2) with a trajectory through it. Only the black dots are observed and thus measured.

In our graph setting, we already have seen that no infinite refinement is possible (and necessary), as the state-space is discrete. Moreover, only finitely many transitions from one state to another are possible and there exist more of them in K_n than in C_n. Suppose $E(C_n) = \{\{i \mod n, i+1 \mod n\} \mid i \in \mathbb{N}\}$, then state j can be reached from state i ($i \neq j$) in exactly two ways: Following around the cycle in either direction and passing all other states in between. For K_n, every path from one state to another is possible by definition. This phenomenon is also reflected by the assumption that every possible transition (automorphism) is equiprobable, which means that every state (node on the orbit) is equiprobable, too (recall the proof in Section 2.4.2).

In the continuous case, the question, which path was taken from one state to another, is addressed by the successive refinement of the state-space partition. In our discrete case, however, this is only possible to a certain degree. Therefore, the group order does not matter.

6.4.3.3 Discussion

The definitions of the two previous sections strongly depend on how the state-space of the dynamical system is defined. We have defined the state-space in a way that the orbits of the graphs' automorphism groups correspond to the orbits of the dynamical system, because the nodes represent the different states. This is why the state is observed and measured per orbit and Example 51 showed that only the non-trivial orbits are actually important for the computation of the KSE. However, as we discussed shortly after Example 54, which showed that the entropy of the complete and the cycle graph of n nodes are identical, our definitions do not capture the uncertainty caused by the larger order of the automorphism group of the complete graph.

If the order of the group is of importance, another definition of the state space is necessary: The states X are the set of all possible partitions of G and $\mathcal{X}$ is a corresponding sigma-algebra. The measure μ still counts the occurrences of the different states, but now the label vectors that represent the states contain entries for *all* nodes instead of only one entry for each orbit. The partitions of the state-space are also not directly related to the graph partitions anymore. Instead, they are partitions of the space of all partitions. When the supremum is taken over the partitions, the maximum value is reached for the partition that results in the maximum number of different cluster labels per node. This maximum number is, of course, exactly the group order.

Example 55. *Reconsider Example 54. The graphs K_n and C_n both have only one orbit of length n and instead of using one arbitrary node to observe the current state, all n nodes are observed in parallel. Clearly, for both graphs the singleton partition yields the maximum entropy, but with the alternative definition of the state-space, there are $|C_n| = n$ different states for C_n and $|K_n| = n!$ different states for K_n. As a consequence, the entropy of C_n is smaller ($-n \cdot \frac{1}{n} \operatorname{ld} \frac{1}{n} = \operatorname{ld} n$) than the entropy of K_n ($-n! \cdot \frac{1}{n!} \operatorname{ld} \frac{1}{n!} = \operatorname{ld} n!$).*

Both definitions of the state-space have advantages and disadvantages. The one we have presented in detail and will use in the following Chapter 7 has the advantage that the graph symmetry and the partitions directly relate to the associated dynamical system. The disadvantage is that it cannot distinguish the different group complexities in terms of the group order. For the alternate definition, which we briefly discussed in this section, the situation is exactly the other way round, i.e. the advantages become disadvantages and vice versa.

6.4.3.4 Computing the Entropy

In this section we describe how the KSE of a graph can be efficiently computed. First, the entropy of a graph partition is discussed, as it is the more complex part on the one hand, and the more interesting part, in terms of an analysis, on the other hand.

As we have discussed above,

$$h_{Aut(G)}(P) = -\sum_{i} \sum_{\boldsymbol{l} \in L(P)^k} \mu_{H_i}(\boldsymbol{l}) \operatorname{ld} \mu_{H_i}(\boldsymbol{l}) \tag{6.71}$$

is already a simplification, as we decompose the automorphism group into its support disjoint normal subgroups H_i.

A further computational simplification is to skip all subgroups H_i for which $\forall o \in O(H_i) \exists C \in P : o \subseteq C$. $O(H_i)$ is the orbit partition that is induced by H_i. This is of course equivalent to $|P^{H_i}| = 1$. The reason is obvious: If each orbit that is induced by the subgroup is a subset of one of the clusters, all local symmetry is restricted to nodes of the cluster(s). As a consequence, $\mu_{H_i}(C) = 1$, and the entropy for this subgroup is zero.

Although $Aut(G)$ is decomposed and some subgroups can possibly be omitted for the computation, generating a subgroup by complete enumeration is often still computationally expensive. Fortunately, it is not necessary to actually generate the group. All that needs to be done is to find one permutation $g \in H_i$ for which

1. $\operatorname{supp}(g) = \operatorname{supp}(H_i)$ and,
2. with $g = c_1 c_2 \cdots c_k$, $k = |O(H_i)|$.

Condition 1 assures that g acts on every node on which the whole group acts on, and from condition 2 follows that all nodes per orbit are mapped onto each other. The c_i in condition 2 are the different disjoint cycles of g. Note that no subgroup H_i fixes any node, i.e. there are no trivial orbits in $O(H_i)$. If a permutation g that suffices these conditions is found (there not necessarily exists such a permutation), only $h_{\langle\{g\}\rangle}(P)$ needs to be computed.

Theorem 16 (Equality of KSE for groups with the same orbits)**.** *For $g \in H_i$ with* $\operatorname{supp}(g) = \operatorname{supp}(H_i)$ *and g consists of exactly $k = |O(H_i)|$ cycles, $h_{\langle\{g\}\rangle}(P) = h_{H_i}(P)$ holds.*

Proof. The two conditions that g must fulfill assure that $\langle\{g\}\rangle \le H_i$ has the same orbits as H_i. The result of Theorem 3 states that every node on an orbit has the same probability to be reached (in terms of permutation maps) by any other node on the same orbit. Therefore, $h(P)$ only depends on the orbits, not on the actual group that defines them. □

Lemma 3. *If $|O(H_i)| = 1$, $h_{H_i}(P) = h_{\langle\{p_o\}\rangle}(P)$ holds, with p_o an arbitrary permutation of the nodes on the only orbit o.*

Proof. As there is only one non-trivial orbit, no dependencies to other orbits exist and Theorem 3 assures the relation. □

Combining all the simplifications, the definition of the KSE of a graph partition is

$$h_{Aut(G)}(P) := -\sum_{i:|P^{H_i}|>1} \sum_{\boldsymbol{l}\in L(P)^k} \mu_{\langle\{g_i\}\rangle}(\boldsymbol{l}) \operatorname{ld} \mu_{\langle\{g_i\}\rangle}(\boldsymbol{l}). \tag{6.72}$$

The g_i for each normal subgroup H_i must fulfill the conditions of Theorem 16.

Example 56. *Consider the graph C_4 in Figure 6.11 with the shown partition. The automorphism group is generated by $g_1 = (1\ 2\ 3\ 4)$ and $g_2 = (1\ 4)(2\ 3)$, clearly has only one orbit, and it is indecomposable. As a result, $h_{Aut(C_4)}(P) = -(\mu(C_1) \operatorname{ld} \mu(C_1) + \mu(C_2) \operatorname{ld} \mu(C_2)) = 1$. However,*

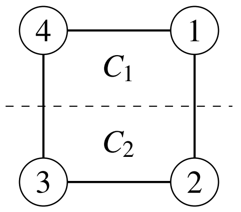


Figure 6.11: The cycle graph C_4, which is transitive and thus has only one non-trivial orbit.

for $g' = (1\ 3\ 4\ 2) \notin Aut(C_4)$ follows $h_{\langle\{g'\}\rangle}(P) = h_{Aut(C_4)}(P)$ due to Lemma 3.

In contrast to the situation in Figure 6.11, there are two dependent non-trivial orbits in the graph T in Figure 6.12.

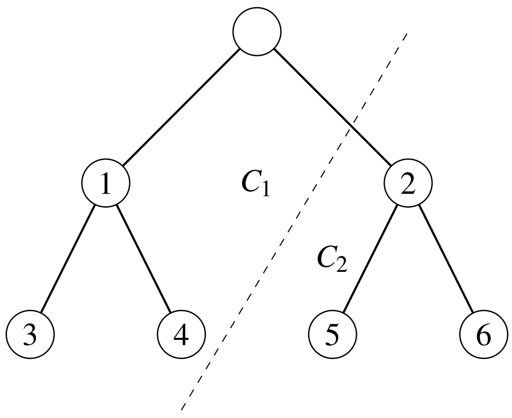


Figure 6.12: The same graph as in Figure 6.9, but with a different partition.

If we would use the permutation $g' = (1\ 2)(3\ 4\ 5\ 6) \notin Aut(T)$ *instead of, e.g.,* $g = (1\ 2)(3\ 5\ 4\ 6) \in Aut(T)$ *to generate a group that has the same orbits as* $Aut(T)$*, the entropy values differ:*

$$\begin{aligned} h_{\langle\{g'\}\rangle}(P) &= -\big(\mu((C_1 \quad C_1)^{\mathrm{T}})\,\mathrm{ld}\,\mu((C_1 \quad C_1)^{\mathrm{T}}) + \mu((C_2 \quad C_1)^{\mathrm{T}})\,\mathrm{ld}\,\mu((C_2 \quad C_1)^{\mathrm{T}}) + \\ &\qquad \mu((C_1 \quad C_2)^{\mathrm{T}})\,\mathrm{ld}\,\mu((C_1 \quad C_2)^{\mathrm{T}}) + \mu((C_2 \quad C_2)^{\mathrm{T}})\,\mathrm{ld}\,\mu((C_2 \quad C_2)^{\mathrm{T}})\big) \\ &= -\,\mathrm{ld}\,\frac{1}{4} = 2 \end{aligned} \tag{6.73}$$

and

$$\begin{aligned} h_{\langle\{g\}\rangle}(P) &= -\,\big(\mu((C_1 \quad C_1)^{\mathrm{T}})\,\mathrm{ld}\,\mu((C_1 \quad C_1)^{\mathrm{T}}) + \mu((C_2 \quad C_2)^{\mathrm{T}})\,\mathrm{ld}\,\mu((C_2 \quad C_2)^{\mathrm{T}})\big) \\ &= -\,\mathrm{ld}\,\frac{1}{2} = 1. \end{aligned} \tag{6.74}$$

Example 57. *Figure 6.13 shows the famous Petersen graph. All its nodes lie on the same orbit; however, none of the* 120 *permutations that form its automorphism group is a single cycle that goes over all nodes. Nonetheless, because of the equiprobability Theorem 3, it is sufficient to consider a group that is generated by, say,* $(a_0\ a_1\ \ldots\ a_4\ b_0\ \ldots\ b_4)$ *for the computation of the KSE of a partition.*

This brings us to the computation of the KSE of a graph. In Section 6.4.3.2 we have argued that any partition that separates all nodes on non-trivial orbits into different clusters yields the maximum entropy. That allows us to implicitly use the singleton partition. However, we do not need to use the explicit definition from Equation 6.72 but can derive and apply further simplifications. We still make use of the decomposition of $Aut(G)$ into its support disjoint normal subgroups that describe the local symmetries. By definition, each of these subgroups has to be taken into account for the computation and, thus, we want to simplify the computation of $h_{H_i}(P_\perp)$ ($P_\perp$ is the singleton partition, as defined in Section 2.3.2). We know from Theorem 3

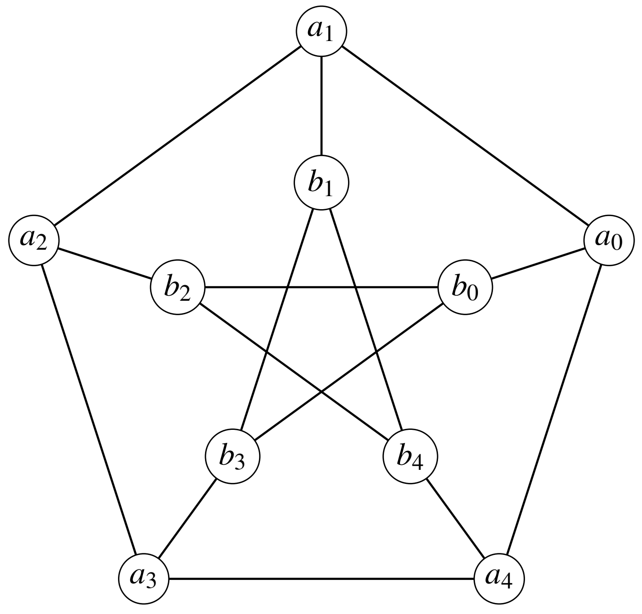


Figure 6.13: The Petersen graph, which is transitive, but its automorphism group does not contain a single cycle that permutes all the nodes at once.

that all nodes on the same orbit are equiprobable. This means, if H_i implies only one non-trivial orbit $O(H_i) = \{o\}$, the entropy is simply

$$-\sum_{C \in P_\perp} \mu_{H_i}(C) \operatorname{ld} \mu_{H_i}(C) = -|o| \frac{1}{|o|} \operatorname{ld} \frac{1}{|o|} = \operatorname{ld} |o|. \tag{6.75}$$

But what if $|O(H_i)| > 1$? Remember, the subgroups H_i induce only orbit partitions that consist of non-trivial orbits. Theorem 17 answers this question.

Theorem 17 (KSE of a graph in terms of a group with the same orbits generated by a single permutation)**.** *Let H be an indecomposable graph automorphism group and P a partition that suffices $h_H = \sup h_H(P)$ (e.g. $P = P_\perp$). Then, for $\langle\{g\}\rangle \leq H$ with $O(\langle\{g\}\rangle) = O(H)$, $h_H = h_{\langle\{g\}\rangle} = \operatorname{ld} |\langle\{g\}\rangle|$ holds.*

Proof. As we already have seen, the entropy of a graph partition depends only on the orbits. For $P = P_\perp$ we set $L(P) = V$, i.e. each singleton cluster has simply the node label of its containing node as cluster id. For each non-trivial orbit of H, choose an orbit representative and construct the initial label vector $\boldsymbol{l}^{id} \in L(P)^k$. For the chosen P, all the label vectors with $\mu_H(\boldsymbol{l}) > 0$ are just the images $(\boldsymbol{l}^{id})^H$. □

Lemma 4. *For a g that suffices Theorem 17, $h_H = \operatorname{ld} \operatorname{lcm}(|c_1|, |c_2|, \ldots, |c_j|)$, where* lcm *is the least common multiple of the lengths of the disjoint cycles of $g = c_1 c_2 \cdots c_j$.*

The least common multiple of two numbers is defined as the smallest positive integer that is divided by both numbers. Clearly, the least common multiple of j numbers is then the smallest possible integer divisible by all the j numbers. It can be calculated relatively efficient (Knuth, 1997, pp. 333 ff.).

Proof. Given $g = c_1 c_2 \cdots c_j$. For each individual cycle c, $c^{|c|} = ()$ holds. So for some fixed c',

$$g^{|c'|} = c_1^{|c'|} \cdots c'^{|c'|} \cdots c_j^{|c'|} = c_1^{|c'|} \cdots () \cdots c_j^{|c'|}, \tag{6.76}$$

i.e. c' vanishes. The smallest k for which $g^k = ()$, i.e. all cycles vanish simultaneously, is just $k = \mathrm{lcm}(|c_1|, \ldots, |c_j|)$. □

Lemma 5. *For an indecomposable group H, $h_H = \mathrm{ld}\,\mathrm{lcm}(|o_1|, |o_2|, \ldots, |o_k|)$ with $o_i \in O(H)$.*

Proof. As it is known that every possible node on the orbit is reached, it is not necessary to actually find a permutation that generates a subgroup that realizes this behavior. It is only important to find the number of different symbol vectors, which are all equiprobable. However, this number is just the least common multiple of the lengths of all orbits. □

Lemma 5 gives a very convenient way to actually compute the KSE of a graph. Note, however, that the lemma is only applicable on indecomposable groups. If $Aut(G)$ can be decomposed into several subgroups, $h_{Aut(G)} = \sum_i h_{H_i}$ holds, and the sum goes over all support disjoint normal subgroups.

Example 58. *Reconsider the two graphs in Figures 6.11 and 6.12 of Example 56. C_4 has one non-trivial orbit of length 4, therefore $h_{Aut(C_4)} = \mathrm{ld}\, 4 = 2$.*

The tree graph T has an indecomposable automorphism group. For applying Theorem 17, the permutation $g = (1\ 2)(3\ 5\ 4\ 6)$ can be used, which generates the group

$$\langle g \rangle = \{(), (1\ 2)(3\ 5\ 4\ 6), (3\ 4)(5\ 6), (1\ 2)(3\ 6\ 4\ 5)\} < Aut(T). \tag{6.77}$$

An initial label vector is, e.g., $\boldsymbol{l}^{id} = (1 \quad 3)^{\mathrm{T}}$ and the resulting label vectors are $(\boldsymbol{l}^{id})^{\langle g \rangle} = \{(1 \quad 3)^{\mathrm{T}}, (2 \quad 5)^{\mathrm{T}}, (1 \quad 4)^{\mathrm{T}}, (2 \quad 6)^{\mathrm{T}}\}$. This results in $h_{\langle g \rangle} = \mathrm{ld}\,|\langle g \rangle| = \mathrm{ld}\, 4 = 2$. When Lemma 4 is used, g consists of the two cycles $c_1 = (1\ 2)$ and $c_2 = (3\ 5\ 4\ 6)$ of lengths 2 and 4, respectively. The least common multiple is of course $\mathrm{lcm}(2, 4) = 4$, which also results in $h_{\langle g \rangle} = \mathrm{ld}\, 4 = 2$. And, of course, as $Aut(T)$ induces the two orbits $\{1, 2\}$ and $\{3, 4, 5, 6\}$ of the same lengths as the two cycles of g, also the application of Lemma 5 gives to correct result.

7 Impact of Symmetry on Clustering Partitions of Real-world Graphs

In this chapter a second empirical analysis of real-world graphs is carried out to finally test the actual impact of symmetry on graph clustering. The datasets are the same simple and symmetric graphs as in Chapter 4, therefore, we refer to this chapter for a rough description of them. We want to find evidence and the extent of unstable "optimal" partitions if we use the iterated version of the randomized greedy algorithm, which is based on the core group graph clustering schema (CGGCRGi). The algorithm is a heuristic, but the iterated approach and the use of an ensemble of partitions (the CGGC schema) assures a very good quality of the clustering result, which hopefully is close to the global optimum. Furthermore, we want to get an understanding of the causes of instability of partitions.

In Section 7.1 we describe how the impact can be quantified by measurement. After that, in Section 7.2, we formulate hypotheses that are checked in Section 7.4. Section 7.3 gives an overview of the analysis procedure.

7.1 Measurement

We have already defined a rich toolset to characterize and quantify symmetry and the corresponding stability of graph partitions. In Section 5.2 local graph symmetry was introduced, which is based on considerations of the decomposability of the automorphism group. The relative symmetry of a graph rs_G is a measure of overall graph symmetry, and it is based on the number of nodes on non-trivial orbits. Derived from it, the mean global symmetry $\overline{gs}_G$ is a measure of expected global symmetry. This means that by using the information of how many decomposable subgroups exist, the smaller the measure becomes, the more locally the automorphism group acts on the graph. Finally, a normalized version $\overline{ngs}_G$ was presented. In Appendix D, we present Algorithm 4, which computes a decomposition of the set of generators S into sets of support disjoint generators S_i $(i = 1, \dots k)$.

We now define two additional measures that will be used in our analysis. The average support of a permutation group H is

$$\text{avg}\,|\,\text{supp}\,(H)\,| := \frac{|\,\text{supp}\,(H)\,|}{k}, \tag{7.1}$$

where k is the number of support disjoint (i.e. decomposed) subgroups. Note that this definition implicitly assumes equally sized subgroup supports.

The normalized version

$$\frac{\frac{|\operatorname{supp}(H)|}{k} - 2}{|\operatorname{supp}(H)| - 2} \tag{7.2}$$

equals $\overline{ngs}_G$. Another indicator is the maximum support, which is simply defined as

$$\max_i |\operatorname{supp}(H_i)| \tag{7.3}$$

over all $i = 1, \ldots, k$. When a partition P of the node set is given,

$$\frac{n}{|P|} \tag{7.4}$$

reflects the average cluster size. The assumption of balanced cluster sizes is supported by the findings of Fortunato and Barthélemy (2007), who show that modularity optimal partitions tend to have balanced cluster sizes. Normalizing the average cluster size gives

$$\frac{\frac{n}{|P|} - 1}{n - 1} \approx \frac{1}{|P|}. \tag{7.5}$$

The fraction of generators that cause instability

$$\frac{|\tilde{\Pi}_{inter}|}{|S|} \tag{7.6}$$

is a measure of partition stability: The higher the relation, the more generators result in another partition when they are applied on P. $\tilde{\Pi}_{inter}$ is the set of generators that cause instability, as defined in Section 6.2.1. Recall that the number of generators does not reflect the size of the generated permutation group. Therefore, a better measure would be $\frac{|\Pi_{inter}|}{|Aut(G)|}$, but this requires an enumeration of the whole group, and we argued already that this is often computationally infeasible.

As a quantitative measure of partition stability, we have defined the Kolmogorov-Sinai Entropy (KSE) for graph partitions in Section 6.4, which we also use for the analysis in this chapter. An overview of all the measures and how they are be interpreted is given in Table 7.1.

7.2 Hypotheses

Before we delve into the details of the analysis, let us formulate some hypotheses that are going to be tested. For all hypotheses that are related to partitions, we consider the partitions to be modularity optimal.

$H_{1.1}$ Graphs with a low mean global symmetry (i.e. with local symmetries) are unlikely to have an unstable partition.

$H_{1.2}$ Graphs with a high mean global symmetry are likely to have an unstable partition.

Name	Symbol	Description
Relative symmetry	rs_G	The relative fraction of nodes that are affected by symmetry. A high value means that many nodes lie on a non-trivial orbit.
Mean global symmetry	$\overline{gs}_G$	The relative fraction of nodes that are affected per decomposed subgroup of the automorphism group. Lower values compared to rs_G imply a good decomposability of the group, and the symmetry is rather local.
Normalized mean global symmetry	$\overline{ngs}_G$	A normalized version of $\overline{gs}_G$ with respect to the maximum relative symmetry. A high (low) value implies bad (good) decomposability of the group. It must always be viewed in the context of rs_G or $\overline{gs}_G$, as it does not measure relative symmetry of the graph.
Average support	$\frac{\lvert \text{supp}(H) \rvert}{k}$	Measures the absolute average number of affected nodes per decomposed subgroup of the group H. Must be compared to other absolute values to make valid statements.
Maximum support	$\max_i \lvert \text{supp}\,(H_i) \rvert$	The maximum number of nodes affected by a decomposed subgroup.
Average cluster size	$\frac{n}{\lvert P \rvert}$	An absolute measure for the average number of nodes per cluster.
Normalized average cluster size	$\frac{n/\lvert P \rvert - 1}{n-1}$	A relative measure of cluster size.
Fraction of instability causing generators	$\frac{\lvert \tilde{\Pi}_{inter} \rvert}{\lvert S \rvert}$	High values may imply a bad decomposability of the group or generally a large symmetry impact on the partition. A value of zero implies partition stability.
Kolmogorov-Sinai Entropy	KSE	Measures the disorganization of the space. A value of zero implies partition stability; higher values reflect disorganization of the space but not necessarily instability of the partition.

Table 7.1: Overview of the different symmetry impact measures that are used in this chapter.

$H_{1.3}$ Graphs with a stable partition have a low mean global symmetry.

$H_{1.4}$ Graphs with an unstable partition have a high mean global symmetry.

H_2 Graphs with unstable partitions have a higher average mean global symmetry.

$H_{3.1}$ Graphs with an average cluster size smaller than the average support are very likely to have an unstable partition.

$H_{3.2}$ Graphs with an average cluster size smaller than the maximum subgroup support are very likely to have an unstable partition.

H_4 Graphs with unstable partitions are no real-world networks.

7.3 Description of the Analysis Procedure

In the same manner as described in Chapter 4, the analysis is split into several phases. The two abstract steps are:

1. Analyze the datasets by computing a modularity optimal partition and a set of generators of the automorphism group, and then test if the partition is stable. Compute coefficients that allow the calculation of the measures shown in Table 7.1.

2. Use the obtained data to calculate statistics, plot illustrative figures, and test the hypotheses formulated in the preceding Section 7.2.

We use the same simple datasets as in Chapter 4 but exclude the asymmetric graphs as well as duplicates in advance. Other issues that are discussed in Chapter 4 and in Appendix B (like data formats, details of duplicate cleaning, etc.), are also important, but not repeated here. Moreover, only the simple graphs are considered. In the first step, we report the graph's name as well as its number of nodes n and edges m as "general" information of the dataset. Then, a modularity optimal partition P is computed using the iterated CGGCRG algorithm (Ovelgönne and Geyer-Schulz, 2013). From its result, we report the modularity Q and the number of clusters $|P|$. The partition itself is kept as well to allow testing its stability.

Subsequently, saucy is used to compute a generating set S. Each generator $s \in S$ is used to test $P^s = P$, and the number of generators that violate this partition stability condition $|\tilde{\Pi}_{inter}|$ and its counterpart $|\tilde{\Pi}_{intra}|$ are computed. The set S is partitioned into subsets of independent subgroups using the method of MacArthur et al. (2008). From this decomposition we can eventually derive the total support of the automorphism group $|\operatorname{supp}(Aut(G))|$, the number of disjoint subgroups k, and the maximum support $\max_i |\operatorname{supp}(H_i)|$. All other measures described in Section 7.1 can be computed from these coefficients.

The Kolmogorov-Sinai Entropy is computed by taking into account many of the simplifications described in Section 6.4.3.4. However, for some datasets it becomes necessary to enumerate the automorphism group or a normal subgroup (obtained by decomposition) of it, which is too large. Although we have shown how the computation of KSE is efficiently possible, it is not trivial to obtain a simpler subgroup that acts on the same set of orbits and, therefore, has the same entropy value. This is why we could not obtain the KSE for every graph. Besides this issue, the actual stability of the corresponding partition can also be derived from the coefficient $|\tilde{\Pi}_{inter}|$, which is zero if P is stable.

7.4 Analysis Results

First, the results are analyzed using mainly descriptive statistics. This allows a first glance on the symmetry impact. Second, graphs that have unstable partitions are compared to those with stable partitions. After that, we try to distinguish the graphs having stable partitions from those having unstable partitions by using a simple and naïve classifier that is based on hypothesis $H_{3.1}$.

The mediocre result of the attempted prediction leads to a more detailed investigation of the graphs that have unstable partitions. This reveals that there exist graphs with certainly unstable and “randomly” unstable partitions. A refined prediction that uses this fact yields better results and, therefore, supports this finding.

7.4.1 General

A summary of descriptive statistics of the analyzed datasets is shown in Table 7.2. In comparison to Table 4.2, Table 7.2a yields relatively similar distributions of the values: Overall, the average graph sizes (in terms of n and m) are slightly smaller here, the values of modularity as well. The average graph size is relatively small (remember the large class `chem`) and most of the graphs seem to have a “good” partition based on the modularity values.

	n	m	Q	$\lvert P\rvert$	$\frac{n}{\lvert P\rvert}$
count	629	629	629	629	629
mean	1.0997×10^5	3.7126×10^5	0.5578	39.4980	478.5000
std	8.1360×10^5	1.6998×10^6	0.2109	188.7100	2138.7000
min	2	1	0	1	2
25 %	24	47	0.4738	3	6.6667
50 %	36	71	0.5997	4	8.4000
75 %	54	103	0.6631	6	13
max	1.1951×10^7	1.7846×10^7	0.9988	2477	35,517

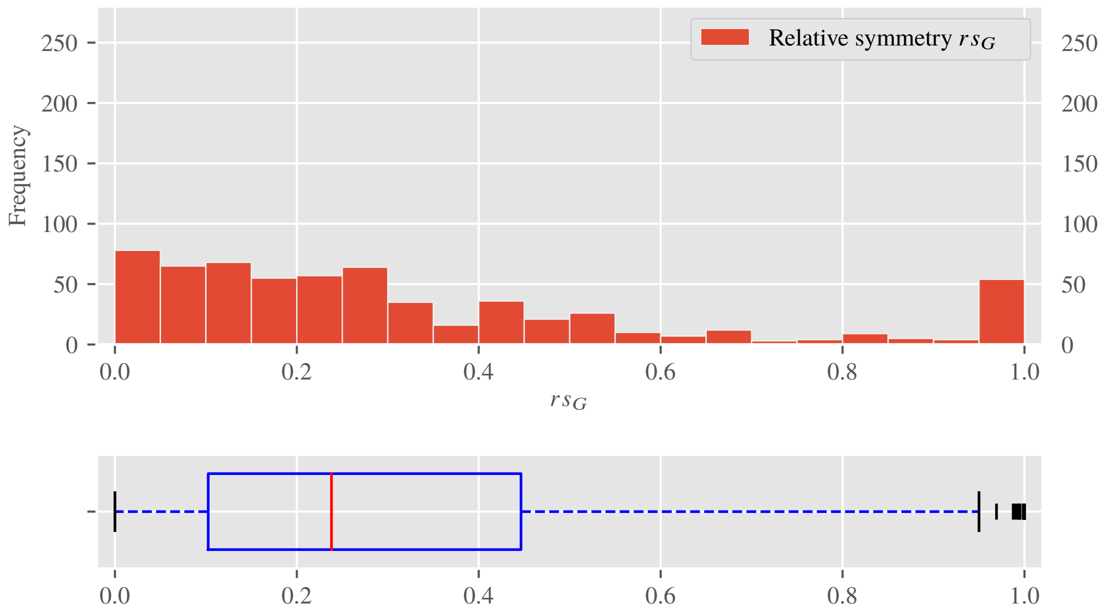
(a) Statistics on the graphs’ sizes and partitions.

	$\lvert\operatorname{supp}(Aut(G))\rvert$	$\frac{\lvert\operatorname{supp}(Aut(G))\rvert}{k}$	rs_G	$\overline{gs}_G$	KSE	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$
count	629	629	629	629	606	629
mean	21,576	2750.6000	0.3228	0.1493	0.2460	0.0554
std	1.1922×10^5	45,495	0.2855	0.2491	1.1048	0.2205
min	2	2	1.2795×10^{-5}	3.5568×10^{-7}	0	0
25 %	4	2	0.1026	0.0488	0	0
50 %	7	2.1154	0.2381	0.0735	0	0
75 %	18	3	0.4465	0.1136	0	0
max	1.5949×10^6	1.0486×10^6	1	1	14.3800	1

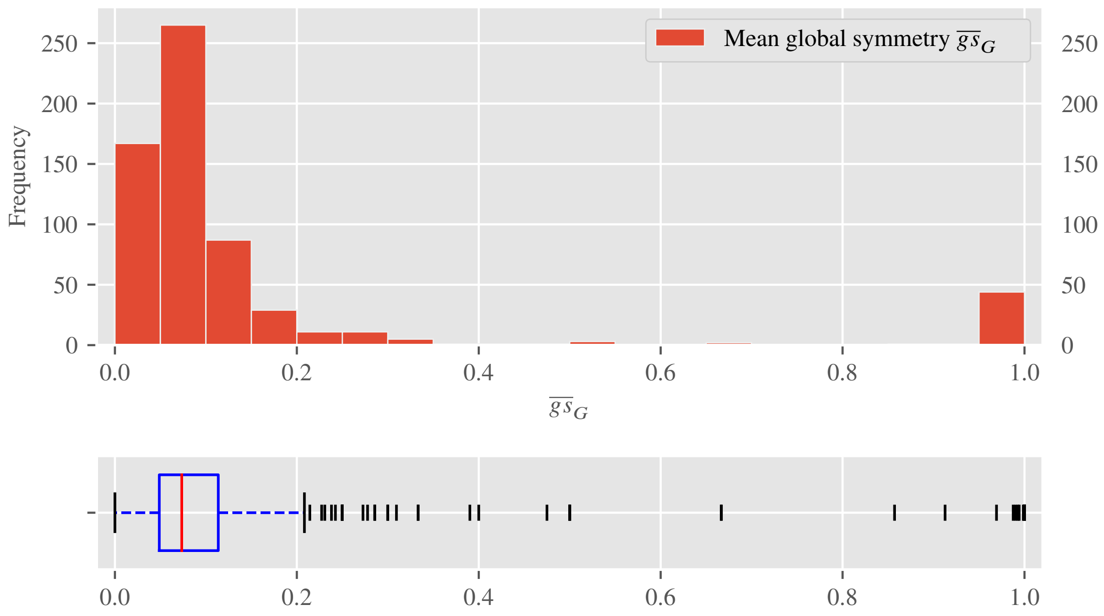
(b) Statistics on the graphs’ symmetry and its impact on partitions

Table 7.2: Partition stability statistics for `networkrepository.com` datasets: 557 of the 629 graphs that were analyzed have a stable modularity optimal partition. 72 graphs have unstable modularity optimal partitions. There exist 58 graphs with positive entropy, 3 of these have a stable modularity optimal partition. Note that KSE could not be computed for all graphs.

Table 7.2b reveals that the average relative symmetry rs_G is about 32 % and that most automorphism groups are decomposable. However, by comparing the maximum values of rs_G and $\overline{gs}_G$, we can see that at least one completely symmetric graph with an indecomposable group must exist. This assumption is supported by Figures 7.1a and 7.1b: Most graphs seem to have a decomposable automorphism group, except those with $rs_G \geq 0.95$. Moreover, this fact is, of course, also reflected in the two columns $\lvert\operatorname{supp}(Aut(G))\rvert$ and $\frac{\lvert\operatorname{supp}(Aut(G))\rvert}{k}$.



(a) The mean relative symmetry is about 0.3, the median is lower. The frequency of graphs with a higher rs_G decreases. As an exception, there exist more than 50 graphs with $rs_G \geq 0.95$.



(b) In direct comparison to (a), it can be assumed that most graphs have either a decomposable automorphism group or, if not, already have a low rs_G. In contrast, the groups of most graphs with $rs_G \geq 0.95$ cannot be decomposed.

Figure 7.1: Comparison of the distributions of rs_G and $\overline{gs}_G$. Most graphs—with only a few exceptions—have a relatively local symmetry.

The distribution of the values of Kolmogorov-Sinai Entropy is very skewed, which is due to the finding that only 72 out of 629 graphs actually have an unstable partition (see also Figure 7.2). The same effect can be seen for $\frac{|\tilde{\Pi}_{inter}|}{|S|}$ (see also Figure 7.3). For a list of the simple graphs with unstable modularity optimal partitions we refer the reader to Table 7.6.

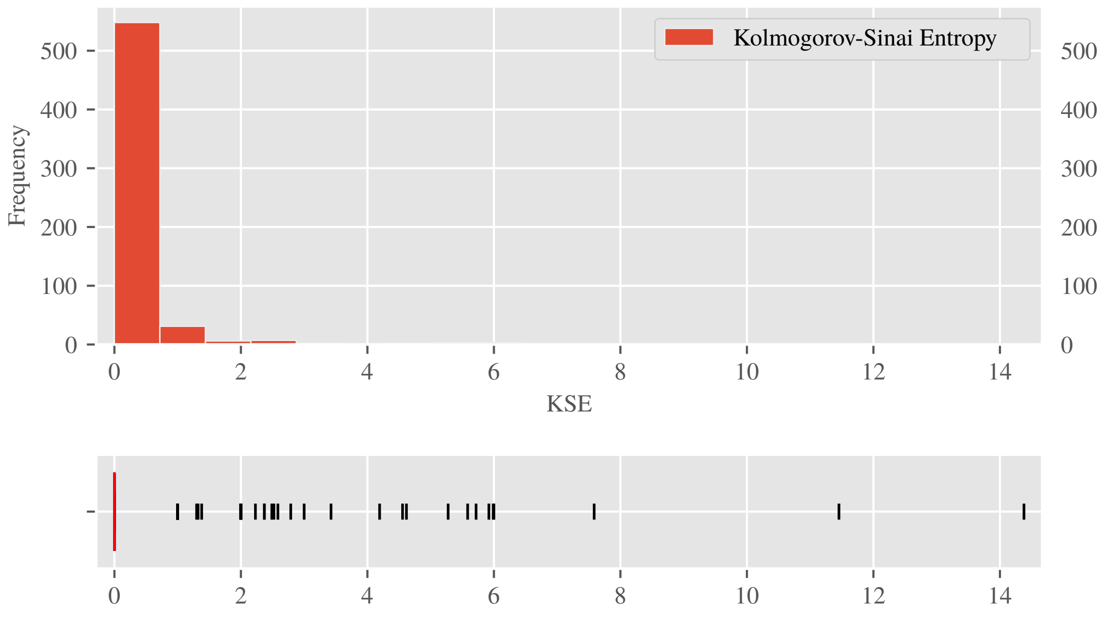


Figure 7.2: The histogram and boxplot of the distribution of KSE values. Only few graphs have positive KSE and even then the value is low, which implies minor instability. The KSE value could not be obtained for 23 of the 629 datasets (see also Table 7.2b).

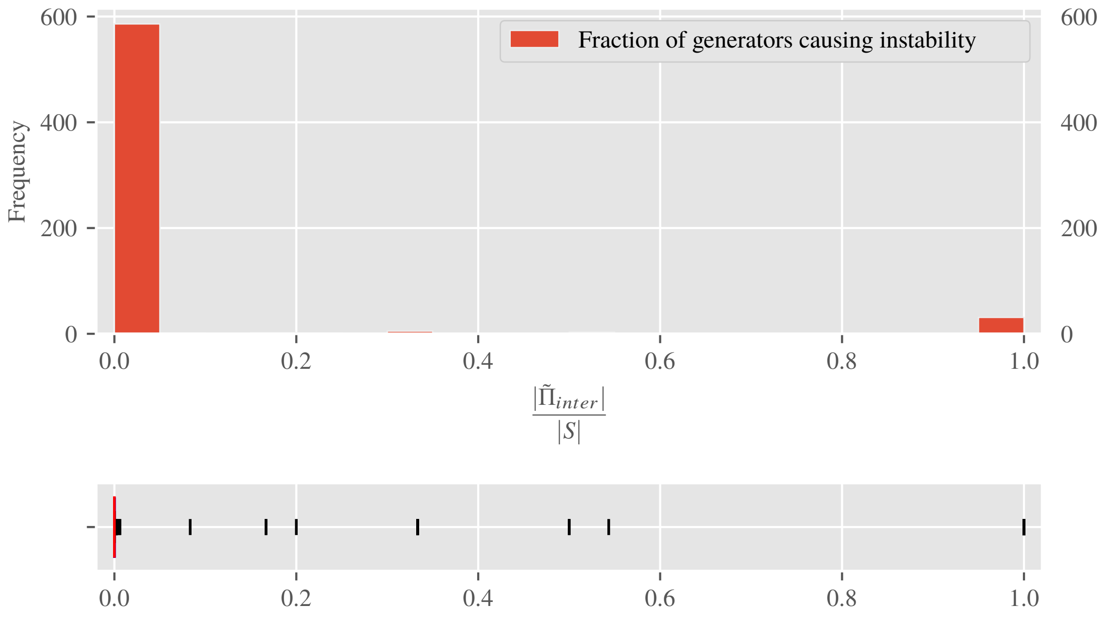


Figure 7.3: The fraction of generators that cause instability is mostly zero (or at least very low) or (near) one. Intermediate values $\frac{|\tilde{\Pi}_{inter}|}{|S|} \in [0.05, 0.95)$ are scarce.

Summary This first superficial descriptive analysis result shows that there indeed exist graphs that have unstable partitions, i.e. the graphs' symmetries have an effect on the clustering results. However, the actual effects are minor, as the (maximum) KSE is low. Another interesting insight is given by the fact that the automorphism groups are often decomposable into smaller independent subgroups, which implies more local symmetries. This finding matches the observations of MacArthur et al. (2008).

7.4.2 Comparison of Graphs with Stable and Unstable Partitions

To better understand what causes instability, we now distinguish stable and unstable partitions. Very natural conditions for instability—which we capture by hypotheses $H_{3.1}$ and $H_{3.2}$—are the relations

$$\frac{|\operatorname{supp}(Aut(G))|}{k} > \frac{n}{|P|} \tag{7.7}$$

and

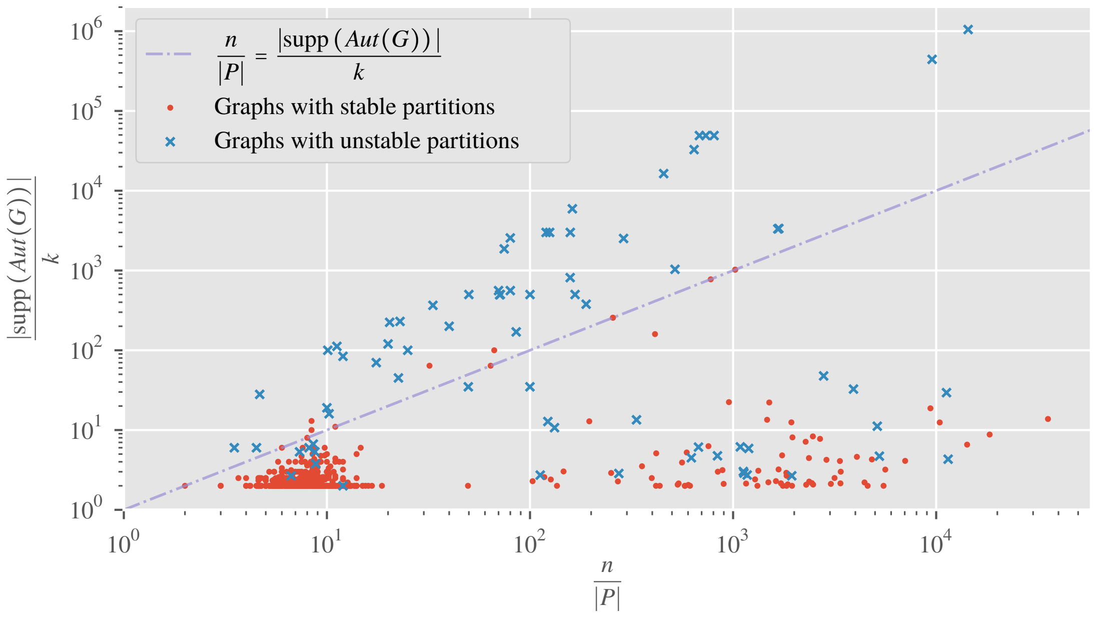
$$\max_i |\operatorname{supp}(H_i)| > \frac{n}{|P|}. \tag{7.8}$$

The H_i are normal subgroups of $Aut(G)$, obtained by decomposition. Equation 7.7 implies instability, because if the average support of the decomposed symmetry subgroups is larger than the average size of the clusters, it is very likely that at least one support set of nodes crosses the boundaries between two or more clusters. The second relation in Equation 7.8 suggests the same, but it is a weaker condition: Only the maximum support, not the average support is taken into account and, especially, $\max_i |\operatorname{supp}(H_i)| \geq \frac{|\operatorname{supp}(Aut(G))|}{k}$ holds. Therefore, if Equation 7.7 holds, also Equation 7.8 holds.

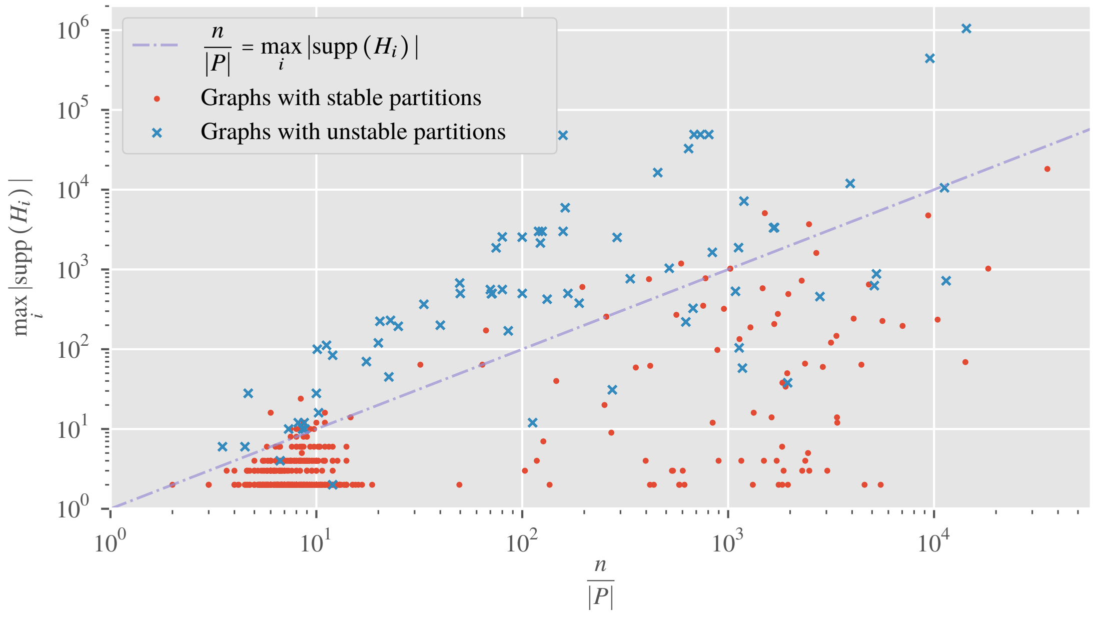
In Figure 7.4, both relations are visualized by scatterplots of the corresponding measures (i.e. $\frac{n}{|P|}$ compared to $\frac{|\operatorname{supp}(Aut(G))|}{k}$ or $\max_i |\operatorname{supp}(H_i)|$, respectively) for all datasets. Graphs with stable and unstable partitions are distinguished by color. The drawn border $\frac{|\operatorname{supp}(Aut(G))|}{k} = \frac{n}{|P|}$ ($\max_i |\operatorname{supp}(H_i)| = \frac{n}{|P|}$, respectively) reveals a reasonably good separation of stable and unstable partitions, especially in Figure 7.4a.

Nonetheless, there also exist a number of graphs for which neither relation holds, especially if the partition is unstable. A quick visual inspection of both Figures 7.4a and 7.4b shows that most datapoints above the drawn line seem to be the same in both figures. These points are those for which the relation from Equation 7.7 (or 7.8, respectively) holds. This suggests indecomposability of the automorphism groups of these graphs, as, for those points that are the same, $\frac{|\operatorname{supp}(Aut(G))|}{k} = \max_i |\operatorname{supp}(H_i)|$ holds. We investigate both findings in the following sections.

In Figure 7.5, we compare the mean global symmetry $\overline{gs}_G$ for graphs with stable and unstable partitions. Hypothesis $H_{1.3}$—graphs with stable partitions have low $\overline{gs}_G$—is supported by Figure 7.5a, albeit there exist seven exceptions. By direct inspection of the datasets, one finds out that all these graphs have either a partition with a modularity value $Q = 0$ and only one cluster (six graphs), or a partition consisting of two clusters and very low modularity $Q \approx 0.0439$ (one graph). Furthermore, one network (`ENZYMES-g193`) is the complete graph K_2. The networks are Hamming graphs that come from the `dimacs` category (Hasselberg et al., 1993, p. 465) and are very dense (which explains the low modularity). Thus, these exceptions only have stable partitions due to their high density, which implies the non-existence of meaningful clusters



(a) Scatterplot of all datasets for the values $\frac{n}{|P|}$ and $\frac{|\mathrm{supp}(Aut(G))|}{k}$. Most graphs with unstable partitions lie above the line $\frac{|\mathrm{supp}(Aut(G))|}{k} = \frac{n}{|P|}$, nearly all graphs with stable partitions lie below. The average support for the latter is relatively low for most graphs, independent of their average cluster size.



(b) As for (a), for most graphs with unstable partitions the relation $\max_i |\mathrm{supp}(H_i)| \leq \frac{n}{|P|}$ seems to hold. However, the values for $\max_i |\mathrm{supp}(H_i)|$ are much more scattered than for $\frac{|\mathrm{supp}(Aut(G))|}{k}$.

Figure 7.4: Visualization of the relations between average cluster size and average support (a) as well as maximum support (b). Graphs with stable partitions are represented by red dots, graphs with unstable partitions are represented by blue crosses. All axes are log-scaled.

under the assumptions that underlie the modularity definition. This means that the symmetry is “hidden” within the clusters and does not conflict with $H_{1.3}$, as partitions that consist of only one cluster are always stable, independent of the how global the symmetry is.

The distribution of $\overline{gs}_G$ for graphs with unstable partitions visualized in Figure 7.5b reflects the fuzzy picture of Figure 7.4. This is inspected further in the next sections.

However, as Table 7.3 shows, hypotheses $H_{1.3}$ and $H_{1.4}$ hold. The hypotheses are tested by the non-parametric Wilcoxon signed-rank test (Wilcoxon, 1945), which does not assume a normal distribution of the values. The threshold for a high mean global symmetry (0.5) is justified as follows: $\overline{gs}_G = 1$ if $rs_G = 1$ and the group is indecomposable ($k = 1$), therefore, $\overline{gs}_G = 0.5$ if $rs_G = 1$ and $k = 2$. Smaller values either imply a lower rs_G or a higher k, which both suggests a more local symmetry.

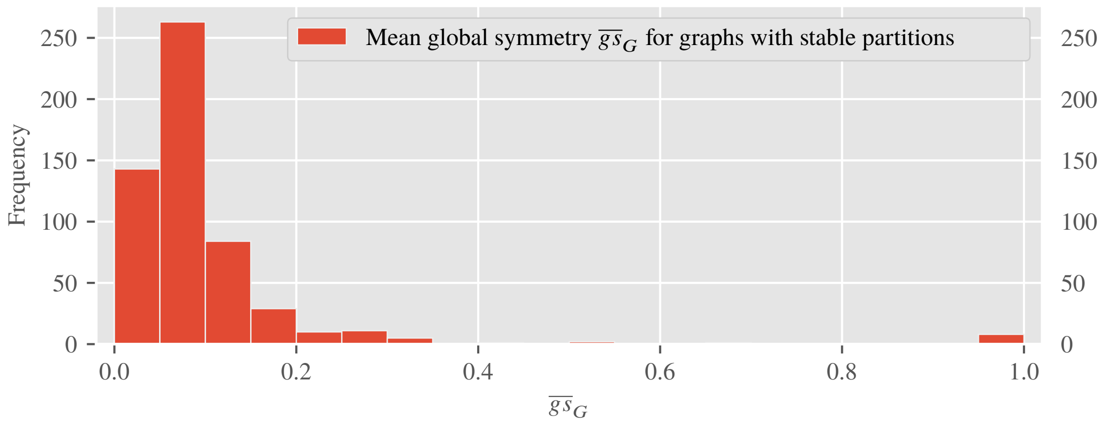
Hypothesis	$H_{1.1}$[§]	$H_{1.2}$[§]	$H_{1.3}$[†]	$H_{1.4}$[†]	H_2[‡]
	$p^s_{\overline{gs}_G<0.1} > p^s$	$p^s_{\overline{gs}_G>0.5} < p^s$	$\text{avg}\,\overline{gs}^s_G < 0.1$	$\text{avg}\,\overline{gs}^u_G > 0.5$	$\text{avg}\,\overline{gs}^s_G < \text{avg}\,\overline{gs}^u_G$
Tested null hypothesis	$p^s_{\overline{gs}_G<0.1} \leq p^s$	$p^s_{\overline{gs}_G>0.5} \geq p^s$	$\text{avg}\,\overline{gs}^s_G \geq 0.1$	$\text{avg}\,\overline{gs}^u_G \leq 0.5$	$\text{avg}\,\overline{gs}^s_G \geq \text{avg}\,\overline{gs}^u_G$
Statistic value	–	–	41,518.5000	757.0000	13,501.0000
p-value	0.0001	0.0000	0.0000	0.0013	0.0000

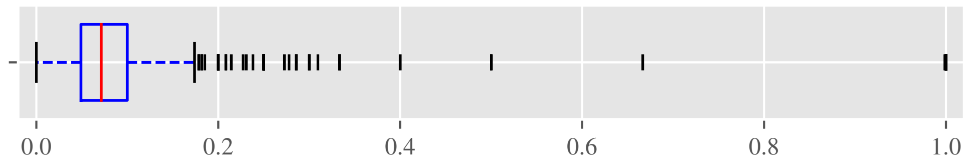
Table 7.3: Test-statistic and p-values[a] for five hypotheses. The two tests marked with § are performed using a binomial test, those marked with † by the one-sided Wilcoxon signed-rank test, and ‡ is tested by the Mann-Whitney-U-test. p^s is the probability that a partition is stable (determined from the data) and p^s_{cond} is the conditional probability of partition stability (“cond” stands for some condition). $\text{avg}\,\overline{gs}^s_G$ ($\text{avg}\,\overline{gs}^u_G$) is the average $\overline{gs}_G$ for graphs with stable (unstable) partitions. The statements that shall be supported by those tests are formulated as alternative hypotheses, the complementary events as null hypotheses. The p-values for all tests are below the 1 % confidence level ($p < 0.01$), thus all null hypotheses must be rejected in favor of the alternative hypotheses.

[a] For all performed tests, the tested null hypothesis must be rejected if the p-value is below a chosen confidence level. The null hypotheses are, therefore, formulated opposite to the desired result.

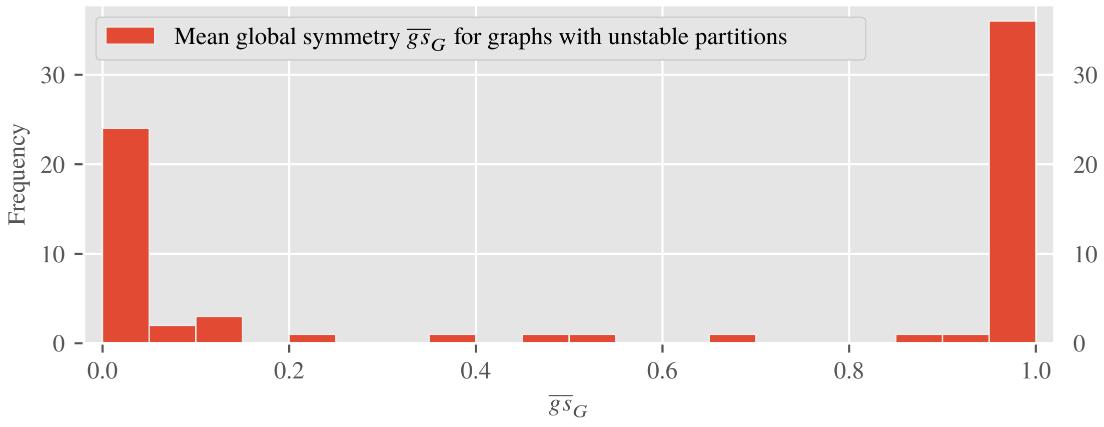
Both hypotheses are confirmed at a reasonably small confidence level and suggest that also H_2—the mean global symmetry for graphs with stable partitions is smaller than for unstable partitions—should be supported. Test results that support this proposition are also shown in Table 7.3. They are obtained by the Mann-Whitney-U-test (Mann and Whitney, 1947), which is preferred to the t-test, as it does not assume normally distributed data.

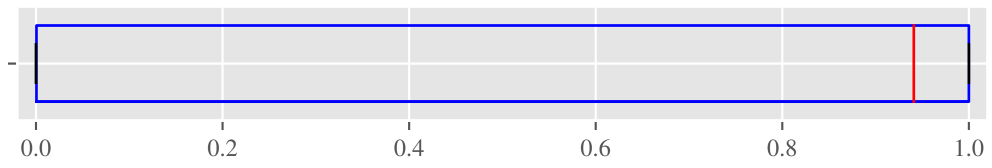
Hypotheses $H_{1.1}$ and $H_{1.2}$ are tested as well. They are more or less the opposite formulations of $H_{1.3}/H_{1.4}$, as they make assumptions about partition stability given a low or high mean global symmetry. Therefore, both hypotheses are checked using a binomial test, which compares the actual positive outcomes (i.e. stable partitions) against a hypothesized probability (i.e. the empirical probability of partition stability). The same thresholds as above for a low (0.1) and high (0.5) $\overline{gs}_G$ are used. Both hypotheses can be supported, i.e. low mean global symmetry has a positive effect on the expected partition stability, high mean global symmetry has a negative





(a) Histogram and boxplot of $\overline{gs}_G$ for graphs with stable partitions. All graphs have a relatively low mean global symmetry, but there are also seven exceptions. They are discussed in the text in Section 7.4.2.





(b) Histogram and boxplot of $\overline{gs}_G$ for graphs with unstable partitions. The situation is completely different than in (a) and needs further investigation. Most graphs have either very low global symmetry or very high global symmetry, intermediate values a scarce.

Figure 7.5: Comparison of the mean global symmetry $\overline{gs}_G$ for graphs with stable (a) and unstable (b) partitions.

effect. For all statistical tests, the implementations of the `scipy.stats` Python package[1] were used.

Summary Comparing the graphs with stable partitions to those with unstable partitions, and graphs with low mean global symmetry to those with high mean global symmetry, gives some interesting results. It can be shown that the symmetry of graphs with stable partitions is significantly more local than for graphs with unstable partitions. Conversely, graphs with a more local symmetry are more likely to have a stable partition and, the other way round, graphs with a more global symmetry are more likely to have an unstable partition. Nonetheless, there seem to exist two groups of graphs with unstable partitions: Those with a low mean global symmetry and those with a high mean global symmetry. This needs to be examined further. Another interesting finding is the relation of average cluster size and average support, which differs between stable and unstable partitions. An attempt to use this relation for the prediction of partition stability is employed in the next section.

7.4.3 Predicting Partition Stability: First Approach

We come back to the hypotheses $H_{3.1}$ and $H_{3.2}$ (Graphs with an average cluster size smaller than the average/maximum support are very likely to have an unstable partition) and employ the naïve classifiers described in Equations 7.7 and 7.8. As in the previous section, we apply a binomial test to check the hypotheses statistically. Table 7.4 shows the results for $H_{3.1}$/$H_{3.2}$ as well as their inverse hypotheses. In both directions, the thresholds for a low and high probability of stability are chosen to allow an error rate of about 10 %. It can be seen that both hypotheses can not be supported by the tests but their inverses can. This means the naïve classification works only well to determine stable partitions, but not the other way round.

Hypothesis	$H_{3.1}$	inv $H_{3.1}$	$H_{3.2}$	inv $H_{3.2}$
	$p^s_{(7.7)} < 0.1$	$p^s_{\neg(7.7)} > 0.9$	$p^s_{(7.8)} < 0.1$	$p^s_{\neg(7.8)} > 0.9$
Tested null hypothesis	$p^s_{(7.7)} \geq 0.1$	$p^s_{\neg(7.7)} \leq 0.9$	$p^s_{(7.8)} \geq 0.1$	$p^s_{\neg(7.8)} \leq 0.9$
p-value	0.4878	0.0000	1.0000	0.0000

Table 7.4: Results of binomial tests for hypotheses $H_{3.1}$ and $H_{3.2}$ as well as their inverses (prefixed by "inv"). The conditions from Equations 7.7 and 7.8 (both suggesting partition instability) are negated by changing inequalities "<" to "≥". p^s_{cond} is again the conditional probability of partition stability. So, e.g., $p^s_{\neg(7.7)}$ is the probability $\mathbb{P}\left(P \text{ stable} \mid \frac{n}{|P|} \geq \frac{|\text{supp}(Aut(G))|}{k}\right)$.

In Table 7.5 the confusion matrices for both classifications are shown: The true classes (stable or unstable) are described per row, the predicted classes per column. As the main goal is to identify instability, the values in the cell *(Unstable, Unstable)* are the true positives (TP), *(Stable, Stable)* are the true negatives (TN). The type I errors (false positives, FP) are *(Stable, Unstable)*, type II errors (false negatives, FN) are *(Unstable, Stable)*, i.e. we recognize a partition as stable that is actually unstable.

[1] `http://scipy.org/scipylib/` as of July 2018, release version 1.1.0 was used

		Prediction	
		Unstable	Stable
Truth	Unstable	43	29
	Stable	4	553

(a) Confusion matrix for the naïve prediction of partition stability using Equation 7.7: Precision for class “Unstable” is 0.9149, recall is 0.5972. Precision for class “Stable” is 0.9502, recall is 0.9928. The overall accuracy is 0.9475.

		Prediction	
		Unstable	Stable
Truth	Unstable	57	15
	Stable	21	536

(b) Confusion matrix for the naïve prediction of partition stability using Equation 7.8: Precision for class “Unstable” is 0.7308, recall is 0.7917. Precision for class “Stable” is 0.9728, recall is 0.9623. The overall accuracy is 0.9428.

Table 7.5: Confusion matrices for both naïve classifiers, which are given by Equations 7.7 and 7.8.

Although the overall *accuracy* $((TP + TN)/(TP + TN + FP + FN))$ for both classifiers is quite high (over 94 %), the test results of Table 7.4 show a high rate of type II errors (low recall for the class *Unstable*). *Precision* is the fraction of correctly classified partitions compared to all predictions of one class $(TP/(TP+FP)$ and $TN/(TN+FN))$; *recall* is the fraction of correctly classified partitions compared to the true classes $(TP/(TP+FN)$ and $TN/(TN+FP))$.

The high accuracy is due to the large number of negatives (i.e. stable partitions) that are classified correctly, which hides the fact of many type II errors in Table 7.5a and many type I and II errors in Table 7.5b. Not unexpected, the weaker classifier, in terms of the strength of the condition (using the maximum support), has a higher recall for unstable partitions, but at the cost of a lower precision for unstable and a lower recall for stable partitions.

Summary The hypotheses that graphs with an average cluster size smaller than the average or maximum support are very likely to have an unstable partition can not be supported, however their inverses can. This means that it is possible to predict partition stability with a very small statistical error, but it does not work for instability. Therefore, we can conclude that there must exist other causes, besides the relation of average cluster size and locality of the symmetric group, that influence partition instability. To investigate this matter further, we concentrate solely on the graphs with unstable partitions in the next section.

7.4.4 Graphs with Unstable Modularity Optimal Partitions

The naïve attempt to predict the stability of a graph clustering partition, in terms of the relation between local graph symmetry and average size of clusters, has turned out to be only half-successful in the previous Section 7.4.3: There exists a large number of graphs with unstable partitions although the instability conditions of Equations 7.7 and 7.8 are not met. Table 7.6 shows the values that are obtained directly by the analysis procedure described in Section 7.3 (except KSE) and the derived values that are described in Section 7.1.

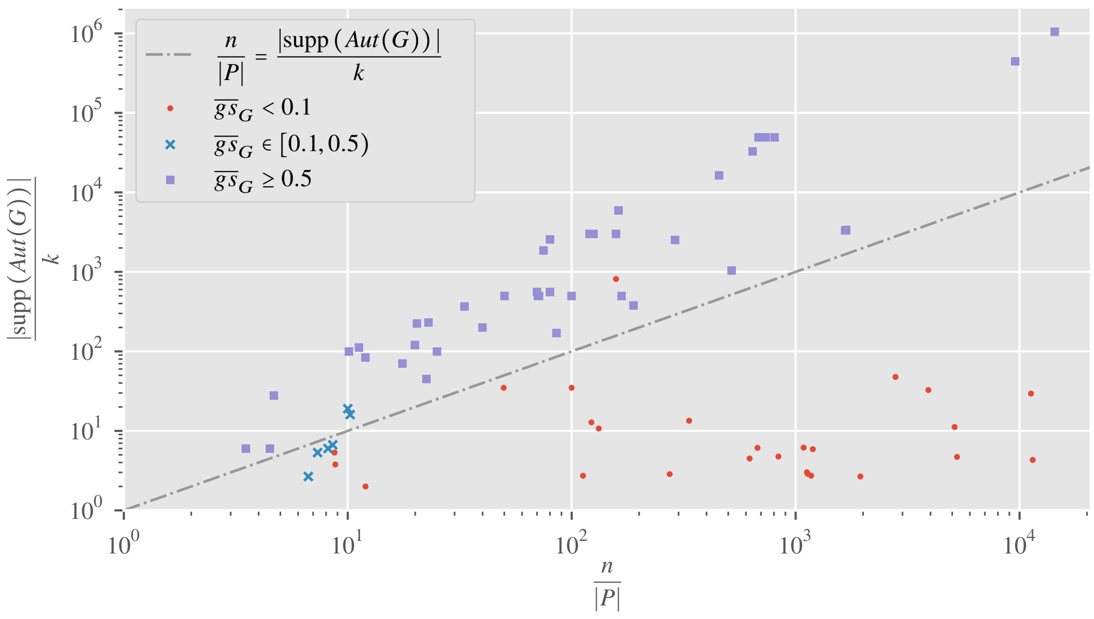
In contrast to Figure 7.4, where all graphs are visible, Figure 7.6 only shows graphs with unstable partitions and distinguishes them by their mean global symmetry $\overline{gs}_G$. Obviously, it can be seen in Figure 7.6a that most of the graphs for which Equation 7.7 does not hold (i.e. the average support is smaller than the average cluster size, which should suggest a stable partition) have a mean global symmetry smaller than 0.5 (i.e. a local or intermediately local symmetry).

The next step is to find a meaningful classification, which hopefully gives clues for an explanation what else are important graph properties that cause partition instability. Due to the exploratory manner of this analysis, we decided to use a clustering method. Figure 7.7 shows the clusters that result from utilizing DBSCAN (Ester et al., 1996) with a neighborhood distance parameter of $\epsilon = 0.1$. Using instead the famous k-means clustering with $k = 4$ results in very similar clusters. As either method is based on distances between data points, a normalization or standardization should be performed in advance. That is why the normalized versions of mean global symmetry ($\overline{ngs}_G$) and average cluster size are used.

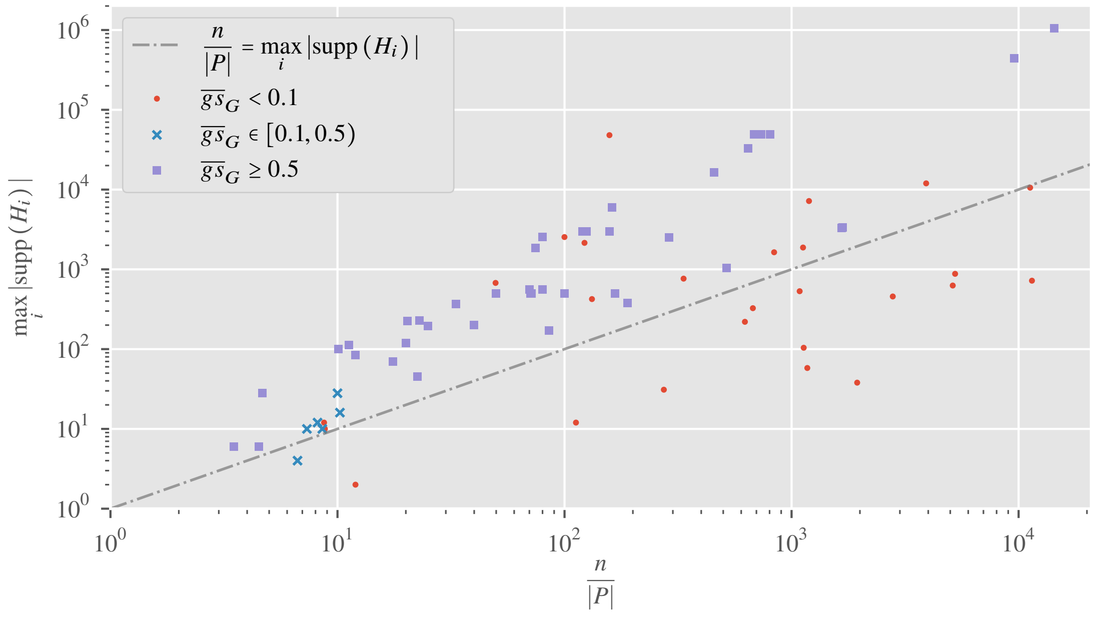
By inspecting the datapoints within the four clusters, there can be found several properties that distinguish them and, therefore, underline that a good separation of the data is found. Next, we describe these properties per cluster and then compare the clusters. Table 7.6 shows the properties of all graphs with unstable partitions. Descriptive statistics of the four clusters can be found in Table 7.7.

Cluster 0 This cluster contains the so called "noise points" of the DBSCAN method, i.e. these points form not an actual cluster in terms of the algorithm, as they lie too far apart of each other (DBSCAN forms clusters of data points that lie in the same ϵ-neighborhood of one data point; here, $\epsilon = 0.1$). There are seven graphs in this cluster and they all are quite small ($20 \le n \le 200$, $37 \le m \le 1534$) and have a notable clustering structure in terms of modularity. The mean global symmetry is strictly smaller than 1, which means that all automorphism groups can be decomposed. Similarly, $\frac{|\tilde{\Pi}_{inter}|}{|S|}$ is also strictly smaller than 1 and for each graph $|\tilde{\Pi}_{inter}| = 1$ holds. The KSE could be computed for all but one partition, and it is 1 for all graphs. Six graphs of this cluster are from the `chem` category; the last graph is called `c-fat200-1`. For five graphs the average classifier fails, and for only one graph the maximum classifier fails.

Cluster 1 This cluster contains 33 graphs. The sizes in terms of nodes and edges are diverse and range from small graphs (`ENZYMES-g509` with $n = 41$ and $m = 88$) to relatively large graphs (`debr` with $n = 1{,}048{,}576$ and $m = 2{,}097{,}149$). Modularity is reasonably high, however, there are five graphs with a (very) low modularity ($Q < 0.17$). All graphs have in common that their automorphism group is indecomposable ($k = 1$) and for 25 of them even $rs_G = \overline{gs}_G = 1$ holds (i.e. every node lies on a non-trivial orbit). For another six graphs, $rs_G = \overline{gs}_G > 0.91$ holds and the remaining two graphs have a rather low $\overline{gs}_G$. The values for $\frac{|\tilde{\Pi}_{inter}|}{|S|}$ are also very diverse: For 26 graphs $\frac{|\tilde{\Pi}_{inter}|}{|S|} = 1$ holds, for five graphs $\frac{|\tilde{\Pi}_{inter}|}{|S|} < 0.006$ holds (all named `c-fat...`), and for two graphs the values lie between these two extremes. Both naïve classifiers hold for all graphs except the one with the lowest mean global symmetry (`ENZYMES-g523` with $\overline{gs}_G = 0.0417$). The KSE could be computed for 27 graphs and the values range from 1 to 11.4551.



(a) For most graphs with low and intermediate mean global symmetry ($\overline{gs}_G < 0.5$) the relation from Equation 7.7 does not hold. Therefore, most of these data points lie below the drawn line that represents the separation of the naïve classifier (points above the line are classified as unstable, points below the line as stable).



(b) In contrast to (a), graphs with low mean global symmetry are much more scattered concerning Equation 7.8, which does not allow a clear distinction.

Figure 7.6: Relations between average cluster size and average (a) or maximal (b) support for graphs with unstable partitions only. All graphs are visually separated by low, high, or intermediate mean global symmetry.

Name	n	m	Q	$\|P\|$	$\|\operatorname{supp}(Aut(G))\|$	$\max_i \|\operatorname{supp}(H_i)\|$	k	$\tilde{\Pi}_{inter}$	$\tilde{\Pi}_{intra}$
`c-fat200-1`	200	1534	0.7133	8	200	194	2	1	162
`ENZYMES-g161`	22	42	0.4504	3	16	10	3	1	2
`ENZYMES-g293`	96	109	0.7693	11	16	12	3	1	2
`ENZYMES-g468`	20	37	0.4481	3	16	4	6	1	5
`ENZYMES-g531`	40	82	0.6401	4	38	28	2	1	1
`ENZYMES-g540`	49	92	0.6295	6	18	12	3	1	2
`ENZYMES-g578`	60	103	0.6734	7	20	10	3	1	2
`auto`	448,695	3,314,611	0.9308	47	443,120	443,120	1	1	4
`bfly`	49,152	98,304	0.8026	61	49,152	49,152	1	3	0
`c-fat200-2`	200	3235	0.6082	5	200	200	1	1	181
`c-fat500-1`	500	4459	0.8061	10	500	500	1	1	419
`c-fat500-10`	500	46,627	0.4049	3	500	500	1	1	491
`c-fat500-2`	500	9139	0.7342	7	500	500	1	1	459
`c-fat500-5`	500	23,191	0.5794	5	500	500	1	1	483
`cage`	366	2562	0.1692	11	366	366	1	8	0
`cca`	49,152	69,632	0.9264	72	49,152	49,152	1	12	0
`ccc`	49,152	73,728	0.8428	67	49,152	49,152	1	4	0
`debr`	1,048,576	2,097,149	0.8541	73	1,048,576	1,048,576	1	2	0
`diag`	2559	4092	0.9384	32	2558	2558	1	2	0
`ENZYMES-g509`	41	88	0.5648	4	16	16	1	1	0
`ENZYMES-g523`	48	111	0.5964	4	2	2	1	1	0
`EX1`	560	4368	0.4581	8	560	560	1	5	0
`EX2`	560	4368	0.4275	7	560	560	1	4	0
`EX4`	2600	35,880	0.4678	9	2520	2520	1	1	0
`fe-sphere`	16,386	49,152	0.9139	36	16,386	16,386	1	3	0
`G48`	3000	6000	0.8605	25	3000	3000	1	4	0
`G49`	3000	6000	0.8617	24	3000	3000	1	4	0
`G50`	3000	6000	0.8644	19	3000	3000	1	4	0
`GD06-theory`	101	190	0.3789	10	100	100	1	25	21
`GD97-a`	84	166	0.5660	7	84	84	1	2	0
`GD98-c`	112	168	0.5333	10	112	112	1	3	0
`grid1`	252	476	0.7308	11	230	230	1	1	0
`grid1-dual`	224	420	0.7311	11	224	224	1	1	0
`johnson16-2-4`	120	5460	0.0084	6	120	120	1	14	0
`johnson32-2-4`	496	107,880	0.0019	7	496	496	1	30	0
`johnson8-2-4`	28	210	0.0544	6	28	28	1	6	0
`johnson8-4-4`	70	1855	0.0312	4	70	70	1	6	0
`se`	32,768	49,150	0.8431	51	32,768	32,768	1	2	0
`ukerbe1`	5981	7852	0.9233	37	5936	5936	1	1	0
`ukerbe1-dual`	1866	3538	0.8985	25	1866	1866	1	1	0
`as-22july06`	22,963	48,436	0.6754	34	13,229	326	2163	2	10,978
`bio-dmela`	7393	25,569	0.4662	27	1665	31	583	1	1050
`ca-citeseer`	227,320	814,134	0.9038	201	110,078	104	38,128	1	70,022
`ca-dblp-2010`	226,413	716,460	0.8671	193	101,526	58	37,170	1	62,776
`com-dblp`	317,080	1,049,866	0.8354	163	134,021	38	50,168	1	81,680
`com-youtube`	140,959	276,042	0.6140	36	120,316	11,966	3689	1	116,624
`ENZYMES-g272`	44	78	0.6871	5	34	10	9	1	11
`power`	4941	6594	0.9392	44	823	12	302	1	413
`rt-islam`	4497	4616	0.6206	45	4252	2540	122	13	4065
`rt-retweet-crawl`	1,112,702	2,278,852	0.7078	217	764,347	627	68,414	4	695,037
`soc-buzznet`	101,163	2,763,066	0.3163	9	24,405	10,567	829	16	23,553
`soc-flickr`	513,969	3,190,452	0.6692	825	237,949	220	52,983	2	182,706
`soc-gowalla`	196,591	950,327	0.7172	175	32,917	1877	10,894	2	20,950
`soc-twitter-follows`	404,719	713,319	0.6924	145	324,947	457	6816	1	318,126
`soc-youtube`	495,957	1,936,748	0.6933	592	144,953	1637	30,483	2	110,324
`soc-youtube-snap`	1,134,890	2,987,624	0.7319	951	549,788	7184	93,492	10	435,964
`tech-as-skitter`	1,694,616	11,094,209	0.8565	148	443,488	721	102,827	2	319,735
`tech-internet-as`	40,164	85,123	0.6933	37	23,444	532	3809	36	19,440
`web-arabic-2005`	163,598	1,747,269	0.9967	1336	132,452	2148	10,354	19	111,420
`web-edu`	3031	6474	0.9522	61	2858	676	82	1	431
`web-indochina-2004`	11,358	47,606	0.9432	86	9681	424	904	6	5898
`web-sk-2005`	121,422	334,419	0.9907	363	100,724	765	7498	10	53,097
`web-uk-2005`	129,632	11,744,049	0.9976	822	129,491	48,090	159	1	127,007
`web-wikipedia2009`	1,864,433	4,507,315	0.8635	355	591,531	879	125,669	2	446,979
`ENZYMES-g352`	9	8	0.3047	2	6	6	1	1	0
`ENZYMES-g55`	7	12	0.2188	2	6	6	1	1	2
`keller4`	171	9435	0.0669	2	170	170	1	2	2
`keller6`	3361	4,619,898	0.0350	2	3360	3360	1	1	5
`MANN-a27`	378	70,551	0.0016	2	378	378	1	6	0
`MANN-a45`	1035	533,115	0.0006	2	1035	1035	1	3	0
`MANN-a81`	3321	5,506,380	0.0002	2	3321	3321	1	8	0
`MANN-a9`	45	918	0.0119	2	45	45	1	4	0

Table 7.6: Computed properties of simple graphs with unstable partitions. The horizontal lines divide the clusters 0 (top) to 3 (bottom) shown in Figure 7.7.

Name	$h_{Aut(G)}(P)$	rs_G	$\overline{gs}_G$	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$	$\frac{\lvert\text{supp}(Aut(G))\rvert}{k}$	$\frac{n}{\lvert P\rvert}$
c-fat200-1		1	0.5000	0.0061	100	25
ENZYMES-g161	1	0.7273	0.2424	0.3333	5.3333	7.3333
ENZYMES-g293	1	0.1667	0.0556	0.3333	5.3333	8.7273
ENZYMES-g468	1	0.8000	0.1333	0.1667	2.6667	6.6667
ENZYMES-g531	1	0.9500	0.4750	0.5000	19	10
ENZYMES-g540	1	0.3673	0.1224	0.3333	6	8.1667
ENZYMES-g578	1	0.3333	0.1111	0.3333	6.6667	8.5714
auto	1	0.9876	0.9876	0.2000	443,120	9546.7021
bfly	5.9205	1	1	1	49,152	805.7705
c-fat200-2		1	1	0.0055	200	40
c-fat500-1		1	1	0.0024	500	50
c-fat500-10		1	1	0.0020	500	166.6667
c-fat500-2		1	1	0.0022	500	71.4286
c-fat500-5		1	1	0.0021	500	100
cage	3.4246	1	1	1	366	33.2727
cca	11.4551	1	1	1	49,152	682.6667
ccc	5.9894	1	1	1	49,152	733.6119
debr	2	1	1	1	1,048,576	14,364.0548
diag	2	0.9996	0.9996	1	2558	79.9688
ENZYMES-g509	1	0.3902	0.3902	1	16	10.2500
ENZYMES-g523	1	0.0417	0.0417	1	2	12
EX1	5.7180	1	1	1	560	70
EX2	7.5850	1	1	1	560	80
EX4	1	0.9692	0.9692	1	2520	288.8889
fe-sphere	5.5850	1	1	1	16,386	455.1667
G48	4.6174	1	1	1	3000	120
G49	4.5558	1	1	1	3000	125
G50	4.1930	1	1	1	3000	157.8947
GD06-theory		0.9901	0.9901	0.5435	100	10.1000
GD97-a	3	1	1	1	84	12
GD98-c	6	1	1	1	112	11.2000
grid1	1	0.9127	0.9127	1	230	22.9091
grid1-dual	1	1	1	1	224	20.3636
johnson16-2-4	2.4887	1	1	1	120	20
johnson32-2-4	2.7881	1	1	1	496	70.8571
johnson8-2-4	2.5216	1	1	1	28	4.6667
johnson8-4-4	1.9948	1	1	1	70	17.5000
se	2	1	1	1	32,768	642.5098
ukerbe1	1	0.9925	0.9925	1	5936	161.6486
ukerbe1-dual	1	1	1	1	1866	74.6400
as-22july06	1.3228	0.5761	0.0003	0.0002	6.1160	675.3824
bio-dmela	1	0.2252	0.0004	0.0010	2.8559	273.8148
ca-citeseer	1	0.4842	1.2700×10^{-5}	1.4281×10^{-5}	2.8871	1130.9453
ca-dblp-2010	1	0.4484	1.2064×10^{-5}	1.5929×10^{-5}	2.7314	1173.1244
com-dblp	1	0.4227	8.4251×10^{-6}	1.2243×10^{-5}	2.6714	1945.2761
com-youtube	1	0.8536	0.0002	8.5745×10^{-6}	32.6148	3915.5278
ENZYMES-g272	1	0.7727	0.0859	0.0833	3.7778	8.8000
power	1	0.1666	0.0006	0.0024	2.7252	112.2955
rt-islam		0.9455	0.0078	0.0032	34.8525	99.9333
rt-retweet-crawl		0.6869	1.0041×10^{-5}	5.7551×10^{-6}	11.1724	5127.6590
soc-buzznet	0.9993	0.2412	0.0003	0.0007	29.4391	11,240.3333
soc-flickr	2	0.4630	8.7380×10^{-6}	1.0946×10^{-5}	4.4910	622.9927
soc-gowalla	2.3710	0.1674	1.5370×10^{-5}	9.5456×10^{-5}	3.0216	1123.3771
soc-twitter-follows	1	0.8029	0.0001	3.1434×10^{-6}	47.6741	2791.1655
soc-youtube	2.3710	0.2923	9.5879×10^{-6}	1.8128×10^{-5}	4.7552	837.7652
soc-youtube-snap		0.4844	5.1816×10^{-6}	2.2937×10^{-5}	5.8806	1193.3649
tech-as-skitter		0.2617	2.5451×10^{-6}	6.2551×10^{-6}	4.3130	11,450.1081
tech-internet-as	14.3800	0.5837	0.0002	0.0018	6.1549	1085.5135
web-arabic-2005		0.8096	7.8194×10^{-5}	0.0002	12.7924	122.4536
web-edu		0.9429	0.0115	0.0023	34.8537	49.6885
web-indochina-2004		0.8524	0.0009	0.0010	10.7091	132.0698
web-sk-2005		0.8295	0.0001	0.0002	13.4334	334.4959
web-uk-2005		0.9989	0.0063	7.8735×10^{-6}	814.4088	157.7032
web-wikipedia2009	2	0.3173	2.5247×10^{-6}	4.4745×10^{-6}	4.7071	5251.9239
ENZYMES-g352	1	0.6667	0.6667	1	6	4.5000
ENZYMES-g55	1	0.8571	0.8571	0.3333	6	3.5000
keller4	2.2296	0.9942	0.9942	0.5000	170	85.5000
keller6	2.5850	0.9997	0.9997	0.1667	3360	1680.5000
MANN-a27	1.3799	1	1	1	378	189
MANN-a45	5.2763	1	1	1	1035	517.5000
MANN-a81		1	1	1	3321	1660.5000
MANN-a9	1.3001	1	1	1	45	22.5000

Continued Table 7.6: Properties derived from the computed properties of simple graphs with unstable partitions.

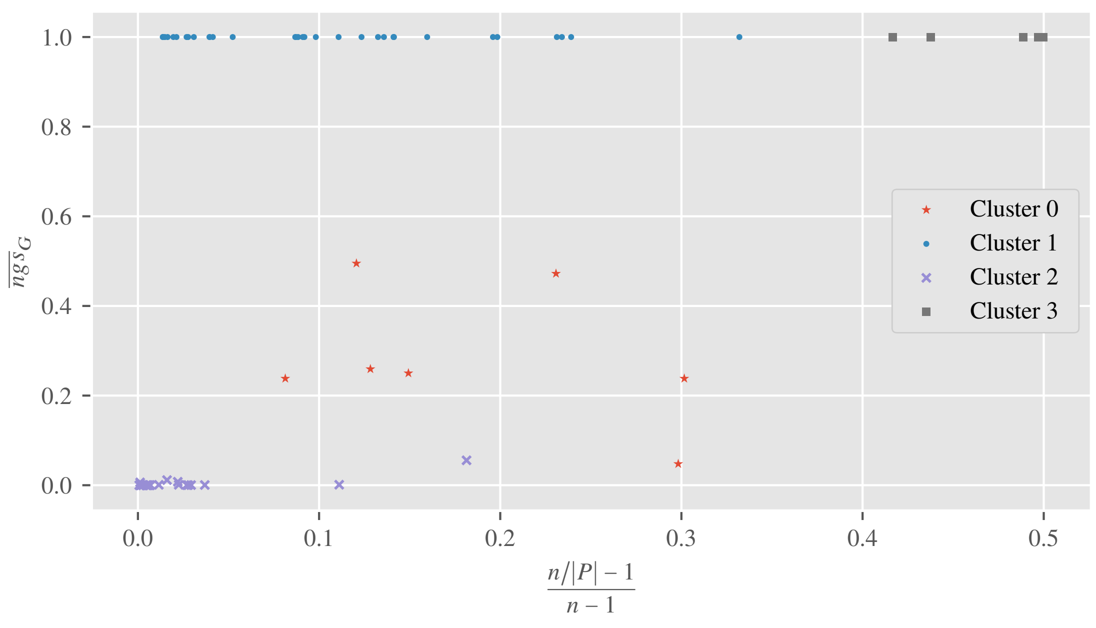


Figure 7.7: Clustering results for the graphs with unstable partitions. Cluster 0 contains the "noise points" of the DBSCAN procedure; the parameter was set to $\epsilon = 0.1$.

Cluster 2 The sizes of the 24 graphs are as diverse as in *Cluster 1* ($28 \leq n \leq 1{,}864{,}433$ and $78 \leq m \leq 11{,}744{,}049$) and modularity is relatively high as well, however, without any downward outliers as in *Cluster 1*. Nonetheless, the graphs are larger on average (the second smallest graph `rt-islam` has $n = 4497$ and $m = 4616$). The relative symmetries are distributed between 0.1666 and 0.9989; however, all automorphism groups are highly decomposable. As a consequence, the mean global symmetry is very small, with the largest value of only 0.0859. This means that the symmetry for all graphs is very local. The values of $\frac{|\tilde{\Pi}_{inter}|}{|S|}$ are also very low, i.e. only few generators from a large number of generators affect the modularity optimal partition. Additionally, the average classifier fails for all graphs but one; the maximum classifier fails for 13 graphs. For nine graphs we could not obtain the KSE, and for the other graphs the values are distributed between 0.9993 and 2.3710. Only one graph (`tech-internet-as`) has a much higher KSE of 14.3800.

Cluster 3 This cluster contains eight graphs of small to medium size. The most interesting commonality is that all their partitions consist of exactly two clusters. Two graphs are from the `chem` category and have a reasonably high modularity, the other six graphs have a very low modularity ($Q < 0.07$). All graphs have an indecomposable automorphism group. The relative and mean global symmetry of the two former graphs is relatively high (0.6667 and 0.8571), and for all the latter it is nearly or exactly 1. Besides their high degree of symmetry, all graph automorphism groups are generated by only a few generators ($|S| \leq 8$). $\frac{|\tilde{\Pi}_{inter}|}{|S|}$ is also quite high and the KSE could be computed for seven graphs. The values for the KSE range from 1 to 5.2763. Both classifiers correctly suggest partition instability for all graphs.

After this detailed description of the four clusters, we can see that clusters 0 and 2 and clusters 1 and 3, respectively, contain similar graphs in terms of their symmetry properties. Clusters 0 and 2 contain graphs with a decomposable automorphism group and only (very) few generators that actually cause the partition instability. In contrast to them, clusters 1 and 3 both contain only graphs with an indecomposable automorphism group ($k = 1$) and a large number of generators cause partition instability. It is interesting to see that the graphs in all clusters have a large degree of symmetry in terms of rs_G, but, based on the different possibilities of group decomposition, the graphs in clusters 0 and 2 are generally only locally symmetric, those in clusters 1 and 3 are mostly very globally symmetric. Another difference is the performance of the two naïve classifiers, especially the one of the stronger average classifier (Equation 7.7): It mostly fails for the graphs in clusters 0 and 2 and mostly predicts instability correctly for the graphs in clusters 1 and 3. The KSE is, of course, positive for all graphs it could be computed for; the actual values are relatively independent of the four clusters. Also, the modularity of most of the graphs (with only few exceptions) is reasonably high and, therefore, suggests the existence of a modular organization of the graphs.

When comparing not only quantitative properties of the graphs in the different clusters, it becomes apparent that there also exists a distinction in the underlying network models. Whereas clusters 0 and 2 mostly contain collaboration/social/retweet/web networks or graphs from the `chem` category, clusters 1 and 3 mainly contains very different graphs. Many of them are generated, either by a random model or by a deterministic one (e.g. Johnson graphs $J(n, k)$, which are determined by two parameters). Furthermore, several subgroups of similar graphs—recognizable by their naming scheme—are contained. Common schemes are `c-fat###-#` (e.g. `c-fat500-1`), `EX#` (e.g. `EX1`), `G#` (e.g. `G48`), `GD##-#` (e.g. `GD97-a`), `johnson##-#-#` (e.g. `johnson16-2-4`), and `MANN-a##` (e.g. `MANN-a27`). It is questionable in what way these graphs can be considered real-world networks. A summary of the comparison of the cluster properties is given in Table 7.8.

All in all, the largest differences between the clusters are the capability of correct classification of the naïve classifiers and the values of the fraction $\frac{|\tilde{\Pi}_{inter}|}{|S|}$. Because of this finding, we conjecture that the partitions of the graphs in clusters 1 and 3 are *certainly unstable*, whereas the partitions of the graphs in cluster 2 (and probably also in cluster 0) are *randomly* or *accidentally unstable*. The reason for the latter “type” of instability could be the very large number of generators paired with the randomized graph clustering heuristic, which makes it likely to accidentally get an unstable partition. Moreover, the graphs with randomly unstable partitions are mostly real-world networks, based on their names. Because of this fact, hypothesis H_4 (graphs with unstable partitions are no real-world networks) cannot be supported.

	n	m	Q	rs_G	$\overline{gs}_G$	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$	KSE
count	7	7	7	7	7	7	6
mean	69.5710	285.5700	0.6177	0.6207	0.2343	0.2866	1
std	63.0080	551.2300	0.1242	0.3288	0.1819	0.1567	0
min	20	37	0.4481	0.1667	0.0556	0.0061	1
25 %	31	62	0.5399	0.3503	0.1168	0.2500	1
50 %	49	92	0.6401	0.7273	0.1333	0.3333	1
75 %	78	106	0.6933	0.8750	0.3587	0.3333	1
max	200	1534	0.7693	1	0.5000	0.5000	1

(a) Statistics for *Cluster 0* (the noise points)

	n	m	Q	rs_G	$\overline{gs}_G$	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$	KSE
count	33	33	33	33	33	33	27
mean	52,156	1.8291×10^5	0.6156	0.9480	0.9480	0.8108	3.4014
std	1.9524×10^5	6.6899×10^5	0.2966	0.1944	0.1944	0.3808	2.5712
min	28	88	0.0019	0.0417	0.0417	0.0020	1
25 %	200	1855	0.4581	1	1	1	1
50 %	500	5460	0.7309	1	1	1	2.5216
75 %	3000	46,627	0.8605	1	1	1	5.1012
max	1.0486×10^6	3.3146×10^6	0.9384	1	1	1	11.4550

(b) Statistics for *Cluster 1*

	n	m	Q	rs_G	$\overline{gs}_G$	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$	KSE
count	24	24	24	24	24	24	15
mean	3.7249×10^5	1.9720×10^6	0.7680	0.5679	0.0048	0.0040	2.2296
std	5.3368×10^5	3.1665×10^6	0.1730	0.2726	0.0175	0.0169	3.4039
min	44	78	0.3163	0.1666	2.5247×10^{-6}	3.1434×10^{-6}	0.9993
25 %	20,062	48,228	0.6842	0.3110	9.9276×10^{-6}	1.0353×10^{-5}	1
50 %	1.5228×10^5	7.6530×10^5	0.7246	0.5303	0.0001	5.9197×10^{-5}	1
75 %	4.2753×10^5	2.3999×10^6	0.9127	0.8146	0.0004	0.0010	2
max	1.8644×10^6	1.1744×10^7	0.9976	0.9989	0.0859	0.0833	14.3800

(c) Statistics for *Cluster 2*

	n	m	Q	rs_G	$\overline{gs}_G$	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$	KSE
count	8	8	8	8	8	8	7
mean	1040.9000	1.3425×10^6	0.0800	0.9397	0.9397	0.7500	2.1101
std	1459.1000	2.3156×10^6	0.1167	0.1210	0.1210	0.3564	1.5231
min	7	8	0.0002	0.6667	0.6667	0.1667	1
25 %	36	691.5000	0.0014	0.9599	0.9599	0.4583	1.1500
50 %	274.5000	39,993	0.0234	0.9999	0.9999	1	1.3799
75 %	1606.5000	1.5548×10^6	0.1049	1	1	1	2.4073
max	3361	5.5064×10^6	0.3047	1	1	1	5.2763

(d) Statistics for *Cluster 3*

Table 7.7: Statistics of the clusters shown in Figure 7.7. Especially the differences of $\overline{gs}_G$ and $\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$ among them are notable. Other properties are discussed in the text.

Cluster	k	rs_G	$\overline{gs}_G$	$\frac{\lvert\tilde{\Pi}_{inter}\rvert}{\lvert S\rvert}$	Miscellaneous observations
1	$= 1$	> 0.9	$= rs_G$	mostly $= 1$	six outliers, described in the text
3	$= 1$	> 0.6	$= rs_G$	$\in (0.15, 1]$	$\lvert P\rvert = 2$
2	$\gg 1$	$\in (0.15, 1)$	$\ll 0.1$	$\ll 0.1$	mostly large collaboration/social/retweet/web networks
0	> 1	$\in (0.15, 1]$	$\in (0.05, 0.5)$	$\in (0.005, 0.5]$	noise points of DBSCAN, small graphs ($n \leq 200$), mostly `chem` networks

Table 7.8: Overview of the four clusters and its properties (see Figure 7.7). Clusters 1 and 3 contain the graphs with *certainly unstable* partitions, clusters 0 and 2 contain the graphs with *randomly unstable* partitions, as described in the text. Both pairs of clusters differ significantly in automorphism group locality, in the fraction of how many generators actually cause instability, and in the performance of the naïve classifiers.

Summary To investigate differences between graphs with unstable partitions a simple cluster analysis was employed. From the total of four clusters (including the "noise points" cluster as argued above) two pairs of clusters are found with only minor differences within the pairs but significant discrepancy between them. The main differences are the results of the two naïve classifiers defined in Equations 7.7 and 7.8, as well as the gap between the fraction of generators that affect stability. Thus, we conjecture that there exist graphs with certainly and randomly/ accidentally unstable partitions.

7.4.5 Predicting Partition Stability: Refined Approach

To emphasize our finding from the previous section, a refined naïve classification approach is presented. The only difference to Section 7.4.3 is that the 24 graphs with randomly unstable partitions, which are identified in cluster 2, are now removed from the data. The classification results can be seen in Table 7.9.

		Prediction	
		Unstable	Stable
Truth	Unstable	42 (43)	6 (29)
	Stable	4	553

(a) Confusion matrix for the naïve prediction of partition stability using Equation 7.7: Precision for class "Unstable" is 0.9130 (0.9149), recall is 0.8750 (0.5972). Precision for class "Stable" is 0.9893 (0.9502), recall is 0.9928 (0.9928). The overall accuracy is 0.9835 (0.9475).

		Prediction	
		Unstable	Stable
Truth	Unstable	46 (57)	2 (15)
	Stable	21	536

(b) Confusion matrix for the naïve prediction of partition stability using Equation 7.8: Precision for class "Unstable" is 0.6866 (0.7308), recall is 0.9583 (0.7917). Precision for class "Stable" is 0.9963 (0.9728), recall is 0.9623 (0.9623). The overall accuracy is 0.9620 (0.9428).

Table 7.9: Confusion matrices for both naïve classifiers given by Equations 7.7 and 7.8 after 24 graphs with randomly unstable partitions are removed. All the values in parentheses are those from Table 7.5 (the first classification).

The accuracy for both approaches increases from 0.9475 to 0.9835, and from 0.9428 to 0.9620, respectively. Especially the classifier that compares the average cluster size to the average support of the automorphism group benefits from the removal of the critical graphs: The number of falsely identified stable partitions decreases from 29 to 6, whereas only one of the correctly identified stable partitions vanishes (see Table 7.9a). As a result, the precision of the class “Unstable” only slightly decreases; for the class “Stable” there is an increase. The highest improvement is a significant increase of the recall (decrease of type II error) for unstable partitions.

As the second classifier already performed better for true negatives and false positives, but worse for false negatives, its precision for the class “Unstable” decreases (see Table 7.9b). Moreover, many graphs that were correctly predicted by the second classifier in the first attempt are now removed. This negative result, however, is not obvious by only comparing accuracy, precision, and recall.

7.4.6 Discussion

In the past sections we have presented the empirical analysis of a large number of symmetric simple graphs (629 in total) concerning an actual impact of graph symmetry on the modularity clustering results, in a similar manner as in Chapter 4. The modularity optimal partitions of 72 graphs are actually affected by the graphs’ symmetry.

In what followed, two naïve classifiers were presented that both perform relatively well. However, falsely identified unstable partitions are an issue. The most interesting part is the further inspection of graphs having unstable partitions, as there could be identified roughly two different subgroups, namely those with certainly and those with randomly unstable partitions. The latter are mostly actual real-world networks (e.g. social or collaboration networks), which are normally the main objective for exploratory analysis by graph clustering. These are, however, the ones with unstable partitions the naïve classifiers often fail to identify.

We want to point out that the purpose of trying to classify graphs, whether they have a stable or an unstable partition, is not a practical but a meta-analytic one. Our findings show that—given a graph partition and the corresponding symmetries that act thereon—it is not only a matter of the locality of the symmetry but also a matter of chance. This means if a very symmetric graph is analyzed with a graph clustering algorithm, the resulting partition may be affected by this symmetry simply by coincidence. The reasons for this phenomenon are diverse: Either the algorithm is not deterministic (as in our case), or the graph contains a symmetric structural motif that is spread over more than one cluster in the modularity optimal partition, or the (heuristic) algorithm is indeed deterministic, but deterministically yields a local optimum that is affected by symmetry. However, in an exploratory setting it is normally unknown (without explicitly testing) if a graph is symmetric or not.

Of course, we must limit these statements to the experimental conditions (modularity clustering using a non-deterministic algorithm), but we do not believe that there exists a simple solution (e.g. using another objective function or algorithm) to fix the issue of unstable partitions. For instance, for every algorithm that produces more and smaller clusters than modularity optimization methods (e.g. algorithms that allow the number of clusters as an external parameter), the effect of symmetry can be assumed to increase due to the ideas our naïve classifiers are based on. Nonetheless, we also could show that symmetry as such not necessarily leads to partition instability However, instability heavily depends on the locality of this symmetry.

From another point of view, our results could also be used to criticize modularity as a function for graph clustering partition quality. Fortunato and Castellano (2012, p. 491) says that "[c]ommunities are groups of vertices which probably share common properties and/or play similar roles within the graph". Garlaschelli et al. (2010, p. 1705) asserts that "[t]he modular structure of real networks can be therefore seen as a symmetry-breaking property". Clearly, both statements contradict the results shown in this chapter. However, it is not the goal of this thesis to criticize modularity, mainly because we believe that other goal functions or optimization procedures are also affected by graph symmetry.

8 Conclusion and Outlook

The title of this thesis captures its main goal: Does graph symmetry affect the partition that results from graph clustering? The road leading to an answer of this question is bumpy, as it involves numerous theoretical concepts, which are—once understood—not overly hard to get but are, nevertheless, necessary to bring together the ideas from different scientific disciplines.

Literature concerning graph symmetry, i.e. literature about graph isomorphisms and automorphisms, often comes from a purely mathematical community. As a downside for our purpose, most of the time only very theoretical considerations are in the focus of, e.g., the recognition of graph symmetry, and literature about symmetry in real-world graphs is scarce. In contrast to that, publications in the field of data analysis, more specifically, publications on graph clustering/community detection and everything in the closer neighborhood, do not consider graph symmetry at all. Additionally, possible symmetry effects in data analysis as a whole are mostly not taken into account at all, which is different to physics, where symmetry is an inherent part of the entire discipline.

This is why the main part of this thesis started with a very obvious and "practical" analysis, where the question is answered, if there even exists symmetry in real-world graphs. If the answer would have been "no", we could have come to an end, as every theoretical consideration that followed would have stayed what it is: A theoretical consideration with no practical impact. Fortunately (in the light of this thesis), it could be shown that many graphs involve a certain amount of symmetry, which made it worthwhile to further investigate the possible effects.

Before the climax of the thesis, in form of a second (again quite obvious) analysis, was reached, several theoretical considerations regarding, both, graph symmetry itself and partition stability in particular, were presented. One important part is the idea to break down the graph symmetry to the smallest independent "areas" within the graph that are actually affected by the symmetry. We call this the "locality" of symmetry and later used the idea in the analysis. Moreover, it is necessary to formally define partition stability concerning symmetry and we came up with several equivalent stability definitions that all allow slightly different views on the problem. In short, partition stability means that the graph symmetry does not affect the partition by exchanging nodes between different clusters. To quantify this effect, additionally an entropy-based measure was defined.

We finally analyzed a large sample of graphs concerning a symmetry impact and could come to the conclusion that for more than 10 % of the considered networks there actually exists an effect. One of the most important contributions of the whole thesis is that partition instability can happen counterintuitively, just because a graph is very symmetric and independent of the locality of symmetry.

To circumvent instability in a practical application context, our theoretical definitions and tools help to efficiently test a graph clustering partition. This is completely independent of the origin of the partition (i.e. which algorithm was used). Regarding the generalizability of our findings, we believe that, on the one hand, further research is needed (e.g. using other algorithms, other optimization goals etc.); on the other hand, we hope that others will incorporate the idea that symmetry as such should be considered in (graph) data analysis. Also, it could be worth trying to explain partition instability in terms of other graph properties—which make the actual computation of the symmetry group superfluous—or by partition sampling. Besides that, our experience in the practical application of graph symmetry algorithms (especially `saucy`) allows us to state that actually solving this problem is feasible, even for larger graphs.

Furthermore, it could be checked if the graphs with partitions identified as randomly unstable actually have unstable partitions by a repeated analysis. Another issue we have not addressed is how existing instability should be handled or interpreted. Ideas could be to simply ignore instability if the effect is small (measured by the Kolmogorov-Sinai Entropy), to break the symmetry before any further data analysis is performed, or to use some kind of fuzzy clustering that allows ambiguous cluster memberships. To summarize, there are roughly two paths of further research: One that concentrates on the handling and interpretation of symmetry effects, and one that tries to further refine the understanding of the conditions that cause partition instability. The latter also implies the need for a deepened understanding of when and why graph symmetry occurs, whether based on constructive graph models or based on the graphs' domain of origin. In a wider sense, also symmetry effects in other data analysis areas beyond graph clustering could be interesting to investigate.

Appendices

A Fraction of Affected Nodes for Fixed Normalized Redundancy

Theorem 18. *Given the normalized network redundancy $r'_G = \lambda$ (for $\lambda \in [0,1]$), approximately a fraction of λ to* $\min\{2\lambda; 1\}$ *nodes are affected by $Aut(G)$.*

Proof. Let $r'_G = 1 - \frac{|O_G|-1}{n-1} = \lambda$. The cardinality of the orbit partition $|O_G|$ can be written as the sum of the number of trivial orbits $k(l)$—the unaffected ones—and the number of non-trivial orbits l: $k(l) + l = |\{o \in O_G \mid |o| = 1\}| + |\{o \in O_G \mid |o| > 1\}|$. All nodes on non-trivial orbits are affected by $Aut(G)$. It follows that $(1-\lambda)n \approx k(l) + l$ with $l \geq 1$, as at least one non-trivial orbit must exist if $Aut(G)$ is not trivial. So for $l = 1$, about $k(l) \approx (1-\lambda)n$ nodes are unaffected, thus λn nodes are affected by $Aut(G)$.

As n is fixed, k decreases with increasing l, which means the number of affected nodes increases. Additionally, $n = k(l) + \sum_{i=1}^{l} |o_i|$ must hold with o_i being the non-trivial orbits. Combining both requirements yields $n = (1-\lambda)n - l + \sum_{i=1}^{l} |o_i|$, which can be simplified to $\lambda n = \sum_{i=1}^{l}(|o_i| - 1)$. Maximizing the factor l is possible up to $|o_i| = 2, \forall i$, as any shorter orbit would be trivial. It follows $\lambda n = l$, thus $k(l) = (1-\lambda)n - \lambda n = (1-2\lambda)n$ nodes are unaffected by $Aut(G)$ and $2\lambda n$ nodes are affected. □

Lemma 6. *If $r'_G = 0.5$, possibly all nodes are affected by $Aut(G)$.*

Proof. This is a direct consequence of the proof of Theorem 18. □

Example 59. *The graph in Figure A.1 has $r'_G \approx 0.5$ although all its nodes are affected by $Aut(G) = \{\mathbf{1}, (1\ 2)(3\ 4)(5\ 6)\cdots(n-1\ n)\}$. The orbit partition is $O_G = \{\{1,2\},\{3,4\},\{5,6\},\ldots,$ $\{n-1,n\}\}$, which implies $r'_G = 1 - \frac{n/2-1}{n-1} = \frac{n/2}{n-1} \approx 0.5$ (for large n).*

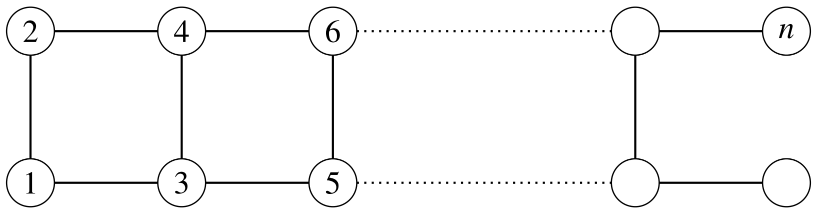


Figure A.1: A graph of an even number of nodes n for which the normalized network redundancy does not capture the high degree of symmetry, as no node is fixed by the automorphism group. The example shows that the (normalized) redundancy does not distinguish between the questions *how many redundancies exist* ($r'_G \approx 0.5$ means 50 % of the nodes have a redundant counterpart) and *how robust is an existing redundancy* ($r'_G \approx 0.5$ means 50 % are structurally identical, therefore redundant, and the other 50 % are not redundant).

B Graph Symmetry Analysis Procedure

This appendix gives additional and more detailed information concerning the first empirical analysis. It has the goal to check how many graphs actually contain symmetries, and the main results are presented in Chapter 4.

The second empirical analysis, which is described in Chapter 7, is based on the same graph datasets and the same algorithms (`saucy` and the RG family) are used. Therefore, many descriptions of this appendix are also related to this second analysis, especially Sections B.1.3–B.1.5 and, for a brief description of the implementations of the algorithms, also Section B.1.6.

B.1 Detailed Description of the Analysis Procedure

We describe how the analysis of network data from `networkrepository.com` was performed in detail. The idea was, according to MacArthur et al. (2008), to compute the symmetries of the graphs with `nauty`, but at a much larger scale. During the process, `nauty` was found to be too inefficient for larger graphs and we ended up by using `saucy` instead. Furthermore, the modularity Q (Newman and Girvan, 2004) should be computed to give a hint how much clustering structure a network has. As MacArthur et al. (2008), we present the automorphism group size $|Aut(G)|$ and the "normalized network redundancy" r'_G (see Equation 4.8).

Because of the large scale of the analysis, Python[1] scripts were written to automate the different sub-tasks:

- Retrieve graph metadata
- Download network data
- Compute properties of the graphs (including modularity and symmetry)
- Calculate basic statistics on the computed results

B.1.1 Graph Metadata

The graph metadata is simply the overview table of all available datasets[2] that already contains attributes extracted from the graphs. However, we do not use these information, as (i) it is often inaccurately aggregated (e.g. "1K" nodes) and (ii) we do not want to rely on them to eliminate possible errors. The only information used is (i) the download link to the dataset, (ii) the "Type" (category) of the dataset, (iii) the "Graph Name", and (iv) the approximate file size for the download selection. The result is saved as a comma-separated values (csv) file.

[1] `https://www.python.org/` as of March 2017, release version 2.7.13 was used
[2] `http://networkrepository.com/networks.php`, as of February 2018

B.1.2 Download of Network Data

Instead of downloading all available data from `networkrepository.com`, we decided to download only a subset. The only reason is that the network sizes differ in a wide range. Small networks only have a couple of nodes and edges and, therefore, have quite small file sizes. For the largest available networks the number of nodes and edges is sometimes not even reported, but their file sizes are often several gigabytes (GB). All data is already compressed via standard zip-archive compression, the graph data formats themselves are plain text. As computation of modularity and symmetry is very time (and space) consuming for huge graphs, we decided to download all graphs that have about 70 megabytes (MB) or less in their compressed form. This may sound quite small compared to the very large available data, but, nevertheless, the largest graph has about two million nodes already. In total, we downloaded 3015 datasets with a cumulative size of about 11.5 GB.

B.1.3 Parsing of Network Data

The downloaded datasets are compressed zip-archives, and each of them normally contains at least two files: First, a `readme.html` that unfortunately contains only general acknowledgment references for `networkrepository.com` (Rossi and Ahmed, 2015) and only rarely further information on the origin of the network data. Therefore, we did not try to extract any further information (e.g. references that describe the data) from this file. Also, the detail page of a dataset (which allows interactive exploration etc., see Section 4.2.1) provides only little additional information. Second, the remaining file(s) usually describe the network itself and we observed two (quite similar) data formats.

Matrix Market Exchange Format The first is the so called “Matrix Market Exchange Format”[3] (Boisvert et al., 1996), which is a sparse matrix ASCII file format (file extension `.mtx`) that allows several matrix structures. The first line of the text file looks like

```
%%MatrixMarket matrix <structure type> <field domain> <symmetry type>
```

where `<structure type>` is either `coordinate` (matrix entries a_{ij} are described by their coordinates i, j) or `array` (matrix entries are described by a sequence of values and the format $m \times n$). None of the datasets had `array` format, so we focus on `coordinate`.

The `<field domain>` provides the domain of the matrix entries (`real`, `complex`, `integer`, or `pattern`). `pattern` is of relevance for simple graphs, as only entries of ones are given; the zero-entries are implicit (this exactly describes an unweighted graph). All other domains describe weights, which can be dismissed if a graph is simplified.

The last modifier denotes symmetry properties of the matrix and we restrict this to be `symmetric` (undirected graph) if we want only simple graphs; `general` is allowed if a graph shall be simplified. To sum up, simple graphs have the format

[3] `http://math.nist.gov/MatrixMarket/formats.html#MMformat` as of February 2017

```
%%MatrixMarket matrix coordinate pattern symmetric
```

and the attribute sets for simplifiable graphs are shown in Table B.1.

Graph type	`<structure type>`	`<field domain>`	`<symmetry type>`
Simple graph	`coordinate`	`pattern`	`symmetric`
Simplifiable graph	`coordinate`	`real` or `complex` or `integer`	`symmetric` or `general`

Table B.1: Differences between the Matrix Market formats for simple graphs and those that can be simplified.

The second line of the text file for the `coordinate` format holds the information on the matrix dimensions ($m \times n$) and how many entries (l) the matrix has in the form

```
<m> <n> <l>
```

Here, $m = n$ holds, as we have adjacency matrices of graphs. Datasets with $m \neq n$ were not analyzed. Comments per line are also possible and start with `%`. The entry lines simply have the format

```
<i> <j>
```

in the `pattern` domain mode. If the matrix is determined to be symmetric, additionally $i \geq j$ holds, which means $a_{ij} = a_{ji}$, but only one entry is actually present in the file. If the domain mode is other than `pattern`, one line has the format

```
<i> <j> <w>
```

with w being the matrix entry $a_{ij} = w$ in the specified domain format. These values are neglected if a non-simple graph is simplified (i.e. w is set to 1).

The two routines `scipy.io.mminfo` and `scipy.io.mmread` were used to parse the Matrix Market Exchange format.[4] `scipy.io.mmread` returns a matrix that can then be used as input parameter for a `networkx.Graph`[5].

Edge List Format The second format was unfortunately not documented at all in a “standard” way. The network data archive file contained (besides the “readme” file) a plain text file with the extension `.edges`, in rare cases another file with the extension `.nodes` was present. No standard documentation on a graph format that uses these extensions could be found, but, of course, the semantic of the extensions and the syntax of the file content itself were helpful.

Sometimes the first and also second line contained some metainformation, for example

[4] `http://scipy.org/scipylib/` as of February 2017, release version 0.18.1 was used
[5] `http://networkx.github.io/` as of February 2017, release version 1.11 was used

```
% directed pos
% 312342 23132 23132
```

and the remaining lines were two-, three or four-tuples of values, separated by a delimiter (mainly comma or blank). After some further research, `KONECT` (The Koblenz Network Collection) was found (Kunegis, 2013), which is yet another online network repository. In the official handbook (Kunegis, 2014, chapter 9), a graph data format is described that mostly matches how the `.edges` files are structured in the first two lines (taken from Kunegis, 2014, p. 47):

```
% FORMAT WEIGHTS
% RELATIONSHIP-COUNT SUBJECT-COUNT OBJECT-COUNT
```

The `FORMAT` is either `sym` (symmetric adjacency matrix, undirected edges), `asym` (asymmetric adjacency matrix, directed edges) or `bip` (bipartite graph, undirected edges) and `WEIGHTS` are either `unweighted` or one of eight other possibilities, which we do not mention here, as they all imply weighted edges and/or multiple edges. The second line is optional and `RELATIONSHIP-COUNT` is the number of edges, `SUBJECT-COUNT` and `OBJECT-COUNT` are the number of nodes in the two node sets V_1 and V_2 of bipartite graphs or simply the total number of nodes for non-bipartite graphs.

The following lines contain the edges in the form

```
FROM_ID TO_ID WEIGHT TIMESTAMP
```

The last two columns are optional. The column delimiter can be "any sequence of whitespace" (Kunegis, 2014, p. 47). Dynamic networks (i.e. edges are added/removed over time) are modeled by giving weights -1 or 1 and usually the timestamp of change.

It appeared to be problematic that not every `.edges` file follows this format strictly. Furthermore, node ids for bipartite graphs are not unique due to a subsequent increase from 1 to $n = |V_1| + |V_2|$, instead they are numbered from 1 to $|V_1|$ (`FROM_ID`) and from 1 to $|V_2|$ (`TO_ID`). This had to be handled before creating the graph data structure.

If a `.nodes` file is present, the information that can be found in this file are used as well. Mostly datasets from the `chem` category have this file, which contains additional node labels/colors. This information is used as input for `saucy` as initial partition for the search procedure.

Format Issues As described above, sometimes it was not easy to correctly parse the graph data. The Matrix Market Exchange Format was relatively easy to parse, only sometimes files started with `%MatrixMarket` instead of `%%MatrixMarket`. We solved this issue by simply adding the missing character automatically before loading the data by the described functions `scipy.io.mminfo` and `scipy.io.mmread`.

Problems with the edge list format were much more substantial: Some files did not have a "header" (the first and sometimes second line) but began directly with the first edge. Other datasets used invalid (in terms of the KONECT graph format) modifiers like `bipartite` instead of `bip` or `undirected` instead of `sym`. Additionally, sometimes other column delimiters (mostly commas) instead of whitespace characters were used to separate the columns, or the third column contained timestamps (which is also not allowed in the KONECT graph format). To overcome these issues, a custom parser was written that only accepts files as simple graphs that either had

- no header and a simple (two column) edge list,
- a header with `FORMAT bip unweighted` or `bipartite unweighted` and a simple edge list,
- a header with `FORMAT sym unweighted`, `symmetric unweighted`, or `undirected unweighted` and a simple edge list,
- a header or not (according to the rules above) and a three column edge list, where the third column's values are timestamps, not weights.

For bipartite graphs we shifted all node ids in the `TO_ID` column by $|V_1|$ to have unique node ids in total. After this procedure, the nodes from the two disjoint node sets V_1 and V_2 are not distinguished anymore, even though for networks that are explicitly modeled bipartite in terms of the data format. Nodes from the two disjoint node sets normally represent different real-world entities (e.g. users and products for a recommendation network).

All these inaccuracies coming from the datasets led to the decision to write our own parser, which is capable to precisely distinguish graphs that are simple from those that are not, but can be simplified.

B.1.4 Data Selection

The decision which of the datasets to use for the actual analysis was not as easy as first thought. As mentioned in Section 2.1, we want to focus on simple graphs only. The naïve approach is to exclude every graph that is not simple, i.e. which is weighted, directed, contains multiple edges (or combinations of those) etc., and/or is disconnected. The disadvantage is that a lot of graphs would drop out of the analysis.

Another way to deal with a non-normal graph is to "normalize" it. If the adjacency matrix is asymmetric (i.e. the graph is directed), the "missing" entries on the right upper/left lower triangle can be filled in. If the graph is weighted, all non-zero weighted edges can be changed to have weight one; loops can be removed. All these procedures transform a non-simple graph into a simple one (so did, e.g., MacArthur et al., 2008). This can be interpreted as keeping the structure of the graph and relaxing the "additional properties" (see Section 3.2.1). If a network is disconnected, the largest connected component (the largest connected subgraph) can be extracted.

In terms of clustering, turning directed edges into undirected ones is the least invasive transformation (except for algorithms that explicitly work on directed graphs). The directions add no additional information on how dense a local area in the graph is. Therefore, clustering algorithms that have the maximization of local density as part of their procedure (implicitly or explicitly) are not influenced by this transformation.

Similarly, dropping the explicit information of the two node sets in bipartite graphs does not influence the clustering result at all. One might think that it is not a good idea to "mix" the different real-world entities represented by the disjoint sets. But in the end, (unsupervised) clustering is about finding a partition of the graph that separates the nodes in a certain way and the entity information can be used after the clustering for a proper interpretation of the results.

The relaxation of edge weights regarding graph clustering has much more influence. Some algorithms can directly deal with weights, others can normally be extended to take weights into account (e.g. the modularity definition can be modified for weighted (undirected) graphs). The weights are an essential property, as they express the strength of a relation. Relaxing them would highly influence the expected result.

All in all, we decided to split the analysis into two parts, one that only uses the simple graphs and another that uses the remaining connected non-simple graphs, which are simplified before the analysis is performed.

B.1.5 Duplicate Cleaning

It was quite difficult to recognize duplicated network datasets, as there are several sources of confusion. The following explanations are for the datasets that can be obtained directly from `networkrepository.com`.

1. The first class of duplicates are networks that are in more than one category but have the same name. The caveat with cleaning those is that the statistics of results grouped by category would be biased, as the "true" unique category is unknown.

2. Less obvious are independently duplicated datasets (probably resulting from different users contributing the same network without recognizing it already exists). These duplicates do not have the same name (e.g. "karate" and "soc-karate" or "movielens-10m-ui" and "rec-movielens-user-movies-10m") and may be in the same or in different categories. However, they can be recognized by comparing properties (such as node/edge count, automorphism group size, number of generators, etc.), which must have the same values. Again, the problem of biased statistics emerges if the duplicates are in different categories.

3. The third class contains (non-)duplicates that are even harder to recognize. They all seem to be "false positives", which means that they have similar but different names and the same properties. This phenomenon emerges from the fact that all the given properties are computed for undirected and unweighted graphs. Therefore, they are all exactly equal as for structurally isomorphic graphs, even if the graphs are not isomorphic when taking directions/weights/... into account. For instance, the two graphs "s3dkt3m2" and "s4dkt3m2" look like duplicates, but the first one is weighted; the second one is structurally the same, but unweighted.

Because of the described difficulties, we dropped networks using a semi-automatic procedure, which is described in Section 4.2.3, and we did this only for statistics on the complete dataset. We further grouped networks by their category/type (see next section) and dropped duplicates after the grouping and per group. This means duplicates in different categories are kept, duplicates in the same category are removed.

B.1.6 Analysis Procedure

For each network several properties were gathered. Trivial properties like the number of nodes n and the number of edges m are, of course, directly available "for free", quite similar to the density of the graph (see Equation 2.29). Furthermore, we use the name of a network (the name of the file containing the data) to identify graphs (or detect duplicates). And because we concentrate on graph clustering and symmetries in graphs, we compute the modularity value Q (see Equation 2.104) and use `saucy` to compute the automorphism group.

To get Q, we use the "original" C++ implementation of the RG algorithm (Ovelgönne and Geyer-Schulz, 2010) and pick the best result from a total of ten runs. For a better integration, a Python extension called `pycggcrg`[6] was developed, which allows to call the different algorithms (RG, CGGCRG, CGGCRGi) directly from Python.

`saucy`[7] is called through our own Python binding `pysaucy`[8], the result is a seven-tuple:

`group size base` and `group size exponent` As permutation groups can become quickly quite large, these two values represent the group order $|Aut(G)|$ as `group size base` $\times$ $10^{\texttt{group size exponent}}$. The value of `group size base` is a floating point number, `group size exponent` an integer.

`levels` Indicates the maximum depth of the search tree (see Section 3.4).

`nodes` The number of executed partition refinements.

`bads` The number of times `saucy` needed to actually backtrack.

`number of generators` The size of the generating set $|S|$.

`support` The sum of the support of all generators $\sum_{s \in S} \operatorname{supp}(s)$.

`orbits` A Python list `o` of length n, where `o[i]` is some integer that is the orbit id of node `i`. Nodes on the same orbit have, of course, the same id.

All return values, except the last one, are directly returned by `saucy` itself. `saucy` provides the possibility to call a user-defined function every time a new generator is found (i.e. a "callback function"). This mechanism is used to compute the orbit partition as list of orbit ids in an

[6] `https://github.com/KIT-IISM-EM/pycggcrg`, as of February 2018, release version 0.2.3 was used
[7] `http://vlsicad.eecs.umich.edu/BK/SAUCY/` as of February 2018, release version 3 was used
[8] `https://github.com/KIT-IISM-EM/pysaucy`, as of February 2018, release version 0.3.1 was used

iterative manner within pysaucy. From the returned values, only the group size and the length of the orbit partition is used for the further analysis.

All these properties are extracted by running a Python script, which is also responsible for loading the data (parsing) and selecting which networks to use. The result is a csv-file containing the described information (namely `name`, `n`, `m`, `density`, `modularity`, `aut_group_size`, `aut_group_size_exp`, `num_generators`, `num_orbits`; the first line is a column header, the following lines represent the analyzed data). This script is executed for the simple graphs and the non-simple graphs separately. The simplification is also performed by this script.

Another script was then used to compute statistics on the extracted data (see the next section). To recover the categories of the networks, we combine the data with the information that is directly obtained from `networkrepository.com` (these attributes were also used to create Table 4.1). For the data processing `pandas`[9] in combination with `numpy`[10] is used.

B.2 Detailed Results

This section provides some more detailed results for each individual category/type. Therefore, we provide for both analyses (on simple and simplified graphs) the tables of standard descriptive statistics. All the tables have the same format as Table 4.2 in Chapter 4. Some tables involve "nan" values ("not a number"), which emerge from the inability to compute the standard deviation for categories that contain only one analyzed graph.

B.2.1 Simple Graphs

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	36	36	36	36	36	36
mean	1042.5000	6.5212×10^5	0.8653	0.0071	0	1
std	620.5700	1.2043×10^6	0.0251	0.0022	0	0
min	450	83,151	0.8231	0.0028	0	1
25%	595	1.4887×10^5	0.8424	0.0054	0	1
50%	945	3.8748×10^5	0.8687	0.0066	0	1
75%	1272	7.1408×10^5	0.8834	0.0090	0	1
max	4000	7.4252×10^6	0.9284	0.0115	0	1

Table B.2: Analysis statistics for category "bhoslib" on `networkrepository.com`: 36 of the 36 analyzed graphs are asymmetric

[9] `http://pandas.pydata.org/` as of March 2017, release version 0.19.2 was used
[10] `http://www.numpy.org/` as of March 2017, release version 1.12.0 was used

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	1	1	1	1	1	1
mean	7393	25,569	0.0009	0.4196	0.1444	6.1562×10^{482}
std	nan	nan	nan	nan	nan	nan
min	7393	25,569	0.0009	0.4196	0.1444	6.1562×10^{482}
25%	7393	25,569	0.0009	0.4196	0.1444	6.1562×10^{482}
50%	7393	25,569	0.0009	0.4196	0.1444	6.1562×10^{482}
75%	7393	25,569	0.0009	0.4196	0.1444	6.1562×10^{482}
max	7393	25,569	0.0009	0.4196	0.1444	6.1562×10^{482}

Table B.3: Analysis statistics for category “bio” on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	5	5	5	5	5	5
mean	3.2880×10^5	3.7294×10^6	3.9911×10^{-5}	0.8351	0.3221	$3.4402 \times 10^{180,274}$
std	1.2820×10^5	6.4390×10^6	3.6608×10^{-5}	0.0404	0.1210	$7.6927 \times 10^{180,274}$
min	2.2641×10^5	7.1646×10^5	1.4829×10^{-5}	0.7864	0.2187	$2.2864 \times 10^{26,554}$
25%	2.2732×10^5	8.1413×10^5	2.0885×10^{-5}	0.8064	0.2637	$1.2879 \times 10^{29,719}$
50%	3.1708×10^5	8.2064×10^5	2.7953×10^{-5}	0.8458	0.2834	$4.7476 \times 10^{31,831}$
75%	3.3269×10^5	1.0499×10^6	3.1510×10^{-5}	0.8464	0.3155	$8.7666 \times 10^{33,586}$
max	5.4049×10^5	1.5246×10^7	0.0001	0.8906	0.5293	$1.7201 \times 10^{180,275}$

Table B.4: Analysis statistics for category “ca” on `networkrepository.com`: 0 of the 5 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	575	575	575	575	575	575
mean	32.3500	62.0260	0.1616	0.5733	0.1199	61.1510
std	15.0450	25.6430	0.1130	0.1128	0.1182	772.9400
min	2	1	0.0182	-2.7756×10^{-17}	0	1
25%	22	43	0.0975	0.5241	0.0303	2
50%	31	59	0.1307	0.5971	0.0930	4
75%	41	82	0.1917	0.6483	0.1739	16.0000
max	125	149	1	0.7677	1	18,432

Table B.5: Analysis statistics for category “chem” on `networkrepository.com`: 96 of the 575 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	78	78	78	78	78	78
mean	644.4100	3.2093×10^5	0.6648	0.0934	0.3023	3.8856×10^{689}
std	724.0100	9.3042×10^5	0.2425	0.1789	0.4566	3.4316×10^{690}
min	28	210	0.0357	9.6794×10^{-5}	0	1
25%	200	15,591	0.5012	0.0127	0	1
50%	400	57,775	0.7001	0.0228	0	1
75%	800	2.0746×10^5	0.8989	0.0471	0.9614	40,320
max	4000	5.5064×10^6	0.9988	0.8055	1	3.0308×10^{691}

Table B.6: Analysis statistics for category “dimacs” on `networkrepository.com`: 54 of the 78 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	45	45	45	45	45	50
mean	1.9897×10^6	4.0216×10^6	0.0054	0.9119	0.0702	$1.4723 \times 10^{198,709}$
std	3.2674×10^6	5.7674×10^6	0.0342	0.1168	0.2018	$1.0411 \times 10^{198,710}$
min	39	170	1.7801×10^{-7}	0.2598	0	1
25%	32,768	89,440	1.4305×10^{-6}	0.8880	0	1
50%	2.1476×10^5	8.1413×10^5	4.5772×10^{-5}	0.9149	0	1
75%	2.2167×10^6	7.0140×10^6	0.0002	0.9870	0.0069	48.0000
max	1.1951×10^7	2.5166×10^7	0.2294	0.9987	0.9772	$7.3617 \times 10^{198,710}$

Table B.7: Analysis statistics for category "dimacs10" on `networkrepository.com`: 30 of the 45 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	5	5	5	5	5	5
mean	31,908	1.2050×10^5	0.0376	0.4642	0.3517	$8.5050 \times 10^{41,955}$
std	37,362	1.5268×10^5	0.0794	0.1848	0.3159	$1.9018 \times 10^{41,956}$
min	32	89	8.5032×10^{-5}	0.2862	0.0387	4
25%	1266	6451	0.0001	0.3345	0.0645	3.0298×10^{18}
50%	32,430	54,397	0.0003	0.3867	0.4011	$3.8484 \times 10^{14,520}$
75%	33,696	1.8081×10^5	0.0081	0.5936	0.4469	$2.2605 \times 10^{35,904}$
max	92,117	3.6077×10^5	0.1794	0.7199	0.8070	$4.2525 \times 10^{41,956}$

Table B.8: Analysis statistics for category "ia" on `networkrepository.com`: 0 of the 5 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	4	4	4	4	4	5
mean	2.4340×10^6	2.8306×10^6	0.0001	0.9774	0.0350	$2.6048 \times 10^{11,227}$
std	2.9453×10^6	3.0079×10^6	0.0003	0.0305	0.0414	$5.8245 \times 10^{11,227}$
min	4941	6594	3.1376×10^{-7}	0.9319	0.0043	5.1851×10^{152}
25%	8.1691×10^5	1.1578×10^6	1.1595×10^{-6}	0.9742	0.0155	2.3742×10^{3804}
50%	1.5223×10^6	2.1510×10^6	2.0240×10^{-6}	0.9899	0.0198	2.3742×10^{3804}
75%	3.1394×10^6	3.8238×10^6	0.0001	0.9931	0.0392	1.4304×10^{6080}
max	6.6865×10^6	7.0140×10^6	0.0005	0.9977	0.0962	$1.3024 \times 10^{11,228}$

Table B.9: Analysis statistics for category "inf" on `networkrepository.com`: 0 of the 4 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	146	146	146	146	146	151
mean	5.8136×10^5	1.2644×10^6	0.0178	0.6722	0.1771	$3.8026 \times 10^{2,517,004}$
std	1.7977×10^6	3.5403×10^6	0.0377	0.2928	0.3220	$4.6728 \times 10^{2,517,005}$
min	23	27	1.8548×10^{-7}	0.0971	0	1
25%	1000	5910.2000	8.2713×10^{-5}	0.3782	0	1
50%	4941	19,990	0.0010	0.8112	0	1
75%	1.0813×10^5	3.4393×10^5	0.0139	0.9171	0.2084	480.0000
max	1.1549×10^7	2.5166×10^7	0.2294	0.9977	1	$5.7420 \times 10^{2,517,006}$

Table B.10: Analysis statistics for category "misc" on `networkrepository.com`: 81 of the 146 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	1	1	1	1	1	1
mean	91,813	1.2570×10^{5}	2.9825×10^{-5}	0.9895	0.0758	3.3631×10^{2037}
std	nan	nan	nan	nan	nan	nan
min	91,813	1.2570×10^{5}	2.9825×10^{-5}	0.9895	0.0758	3.3631×10^{2037}
25%	91,813	1.2570×10^{5}	2.9825×10^{-5}	0.9895	0.0758	3.3631×10^{2037}
50%	91,813	1.2570×10^{5}	2.9825×10^{-5}	0.9895	0.0758	3.3631×10^{2037}
75%	91,813	1.2570×10^{5}	2.9825×10^{-5}	0.9895	0.0758	3.3631×10^{2037}
max	91,813	1.2570×10^{5}	2.9825×10^{-5}	0.9895	0.0758	3.3631×10^{2037}

Table B.11: Analysis statistics for category "rec" on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	2	2	2	2	2	2
mean	5.5860×10^{5}	1.1417×10^{6}	0.0002	0.6449	0.7705	$1.6312 \times 10^{671,985}$
std	7.8362×10^{5}	1.6081×10^{6}	0.0003	0.0231	0.2054	$2.3068 \times 10^{671,985}$
min	4497	4616	3.6812×10^{-6}	0.6286	0.6253	2.8427×10^{9652}
25%	2.8155×10^{5}	5.7318×10^{5}	0.0001	0.6368	0.6979	2.8427×10^{9652}
50%	5.5860×10^{5}	1.1417×10^{6}	0.0002	0.6449	0.7705	$3.2623 \times 10^{671,985}$
75%	8.3565×10^{5}	1.7103×10^{6}	0.0003	0.6531	0.8431	$3.2623 \times 10^{671,985}$
max	1.1127×10^{6}	2.2789×10^{6}	0.0005	0.6613	0.9157	$3.2623 \times 10^{671,985}$

Table B.12: Analysis statistics for category "rt" on `networkrepository.com`: 0 of the 2 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	3	3	3	3	3	3
mean	2.8461×10^{6}	9.9127×10^{6}	0.0005	0.9157	0.4033	$1.1624 \times 10^{24,589}$
std	4.8000×10^{6}	1.3246×10^{7}	0.0005	0.0617	0.3543	$2.0134 \times 10^{24,589}$
min	54,870	1.3112×10^{6}	7.1525×10^{-7}	0.8781	0	1
25%	74,882	2.2861×10^{6}	0.0004	0.8801	0.2730	1
50%	94,893	3.2610×10^{6}	0.0007	0.8821	0.5459	$4.3346 \times 10^{12,764}$
75%	4.2418×10^{6}	1.4213×10^{7}	0.0008	0.9345	0.6050	$3.4873 \times 10^{24,589}$
max	8.3886×10^{6}	2.5166×10^{7}	0.0009	0.9870	0.6641	$3.4873 \times 10^{24,589}$

Table B.13: Analysis statistics for category "sc" on `networkrepository.com`: 1 of the 3 analyzed graphs is asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	18	18	18	18	18	18
mean	4.9906×10^{5}	2.2585×10^{6}	0.0125	0.5304	0.3340	$1.4911 \times 10^{1,777,025}$
std	6.2574×10^{5}	2.1664×10^{6}	0.0373	0.1550	0.2051	$6.3260 \times 10^{1,777,025}$
min	34	78	2.4873×10^{-6}	0.2795	0.0328	4
25%	91,879	4.4732×10^{5}	1.1066×10^{-5}	0.3783	0.2106	3.0142×10^{9994}
50%	3.0066×10^{5}	2.0150×10^{6}	2.5711×10^{-5}	0.5625	0.2955	$1.5698 \times 10^{110,447}$
75%	6.1329×10^{5}	3.1397×10^{6}	0.0003	0.6627	0.4320	$1.4463 \times 10^{400,685}$
max	2.5234×10^{6}	7.9188×10^{6}	0.1390	0.7327	0.7861	$2.6839 \times 10^{1,777,026}$

Table B.14: Analysis statistics for category "soc" on `networkrepository.com`: 0 of the 18 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	29	29	29	29	29	29
mean	15,239	5.4513×10^5	0.0134	0.4069	0.0360	1.4729×10^{6172}
std	15,144	4.8493×10^5	0.0135	0.0955	0.1832	7.9316×10^{6172}
min	1446	2981	0.0004	0.3028	0	1
25%	3068	1.1916×10^5	0.0030	0.3626	0.0004	8
50%	9885	4.8222×10^5	0.0085	0.3868	0.0011	5.8982×10^5
75%	22,900	8.3595×10^5	0.0220	0.4235	0.0019	4.8318×10^9
max	63,392	1.5907×10^6	0.0570	0.8087	0.9882	4.2714×10^{6173}

Table B.15: Analysis statistics for category “socfb” on `networkrepository.com`: 4 of the 29 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	6	6	6	6	6	6
mean	3.3703×10^5	2.0075×10^6	0.0004	0.6789	0.3019	$5.1937 \times 10^{258,834}$
std	6.6827×10^5	4.4566×10^6	0.0008	0.1475	0.1527	$1.2722 \times 10^{258,835}$
min	7476	53,381	7.7265×10^{-6}	0.4721	0.1501	4.6242×10^{655}
25%	29,897	63,988	4.3898×10^{-5}	0.5941	0.2014	4.8567×10^{7282}
50%	51,362	1.1650×10^5	9.0553×10^{-5}	0.6725	0.2386	$2.3575 \times 10^{19,475}$
75%	1.5883×10^5	4.9268×10^5	0.0001	0.7964	0.4313	$6.5455 \times 10^{20,197}$
max	1.6946×10^6	1.1094×10^7	0.0020	0.8507	0.4995	$3.1162 \times 10^{258,835}$

Table B.16: Analysis statistics for category “tech” on `networkrepository.com`: 0 of the 6 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	1	1	1	1	1	1
mean	6809	4.7145×10^6	0.2034	0.1424	0.2936	6.4900×10^{1052}
std	nan	nan	nan	nan	nan	nan
min	6809	4.7145×10^6	0.2034	0.1424	0.2936	6.4900×10^{1052}
25%	6809	4.7145×10^6	0.2034	0.1424	0.2936	6.4900×10^{1052}
50%	6809	4.7145×10^6	0.2034	0.1424	0.2936	6.4900×10^{1052}
75%	6809	4.7145×10^6	0.2034	0.1424	0.2936	6.4900×10^{1052}
max	6809	4.7145×10^6	0.2034	0.1424	0.2936	6.4900×10^{1052}

Table B.17: Analysis statistics for category “tscc” on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	9	9	9	9	9	9
mean	3.1374×10^5	2.8476×10^6	0.0008	0.9017	0.6526	$5.2232 \times 10^{406,573}$
std	6.0295×10^5	4.1827×10^6	0.0011	0.1686	0.2954	$1.5670 \times 10^{406,574}$
min	3031	6474	2.5933×10^{-6}	0.4743	0.0818	6.6973×10^{181}
25%	11,358	37,375	5.5341×10^{-5}	0.9349	0.6986	6.8272×10^{3909}
50%	1.2142×10^5	3.3442×10^5	0.0002	0.9522	0.7323	$7.4476 \times 10^{46,073}$
75%	1.6360×10^5	4.5073×10^6	0.0014	0.9948	0.7775	$3.9497 \times 10^{230,509}$
max	1.8644×10^6	1.1744×10^7	0.0033	0.9976	0.9970	$4.7009 \times 10^{406,574}$

Table B.18: Analysis statistics for category “web” on `networkrepository.com`: 0 of the 9 analyzed graphs are asymmetric

B.2.2 Simplified Graphs

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	1	1	1	1	1	1
mean	453	2025	0.0198	0.4308	0.0752	1.9327×10^{10}
std	nan	nan	nan	nan	nan	nan
min	453	2025	0.0198	0.4308	0.0752	1.9327×10^{10}
25%	453	2025	0.0198	0.4308	0.0752	1.9327×10^{10}
50%	453	2025	0.0198	0.4308	0.0752	1.9327×10^{10}
75%	453	2025	0.0198	0.4308	0.0752	1.9327×10^{10}
max	453	2025	0.0198	0.4308	0.0752	1.9327×10^{10}

Table B.19: Analysis statistics for category “bio” on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	2	2	2	2	2	2
mean	644.5000	53,450	0.4348	0.1419	0	1
std	610.2300	52,836	0.3929	0.1285	0	0
min	213	16,089	0.1570	0.0511	0	1
25%	428.7500	34,770	0.2959	0.0965	0	1
50%	644.5000	53,450	0.4348	0.1419	0	1
75%	860.2500	72,130	0.5737	0.1873	0	1
max	1076	90,811	0.7126	0.2328	0	1

Table B.20: Analysis statistics for category “bn” on `networkrepository.com`: 2 of the 2 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	5	5	5	5	5	5
mean	1.4280×10^{6}	2.4781×10^{6}	0.0008	0.8642	0.4657	$1.5677 \times 10^{175,224}$
std	2.9421×10^{6}	2.7727×10^{6}	0.0014	0.1099	0.2786	$3.5056 \times 10^{175,224}$
min	36,417	2.2451×10^{5}	3.1376×10^{-7}	0.6967	0.0043	3.1392×10^{882}
25%	41,731	2.5881×10^{5}	5.1685×10^{-5}	0.8561	0.4484	2.3742×10^{3804}
50%	49,989	2.1542×10^{6}	0.0002	0.8579	0.5106	6.0441×10^{6232}
75%	3.2556×10^{5}	2.7390×10^{6}	0.0003	0.9126	0.6609	1.9621×10^{9356}
max	6.6865×10^{6}	7.0140×10^{6}	0.0032	0.9977	0.7045	$7.8387 \times 10^{175,224}$

Table B.21: Analysis statistics for category “dimacs10” on `networkrepository.com`: 0 of the 5 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	6	6	6	6	6	6
mean	12,924	30,798	0.0999	0.3871	0.3001	$8.5315 \times 10^{17,077}$
std	19,979	45,456	0.1522	0.3086	0.3589	$2.0898 \times 10^{17,078}$
min	113	2196	8.9348×10^{-5}	0.0892	0	1
25%	350	4196.5000	0.0003	0.1371	0.0306	288.0000
50%	3854	7358	0.0089	0.3359	0.2219	1.6738×10^{7505}
75%	15,555	37,960	0.1802	0.5459	0.4033	1.3023×10^{9798}
max	51,083	1.1657×10^{5}	0.3470	0.8747	0.9260	$5.1189 \times 10^{17,078}$

Table B.22: Analysis statistics for category “dynamic” on `networkrepository.com`: 1 of the 6 analyzed graphs is asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	5	5	5	5	5	5
mean	95.2000	1371.4000	0.2889	0.1556	0.0220	21.2000
std	33.6850	762.4000	0.0545	0.0201	0.0204	24.6310
min	54	350	0.2446	0.1217	0	1
25%	69	880	0.2553	0.1578	0	1
50%	97	1446	0.2591	0.1583	0.0313	8
75%	128	2075	0.3106	0.1671	0.0394	48.0000
max	128	2106	0.3751	0.1733	0.0394	48.0000

Table B.23: Analysis statistics for category “eco” on `networkrepository.com`: 2 of the 5 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	1	1	1	1	1	1
mean	167	3250	0.2345	0.1167	0.0422	288.0000
std	nan	nan	nan	nan	nan	nan
min	167	3250	0.2345	0.1167	0.0422	288.0000
25%	167	3250	0.2345	0.1167	0.0422	288.0000
50%	167	3250	0.2345	0.1167	0.0422	288.0000
75%	167	3250	0.2345	0.1167	0.0422	288.0000
max	167	3250	0.2345	0.1167	0.0422	288.0000

Table B.24: Analysis statistics for category “ia” on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\lvert Aut(G)\rvert$
count	1	1	1	1	1	1
mean	6.6865×10^{6}	7.0140×10^{6}	3.1376×10^{-7}	0.9977	0.0043	2.3742×10^{3804}
std	nan	nan	nan	nan	nan	nan
min	6.6865×10^{6}	7.0140×10^{6}	3.1376×10^{-7}	0.9977	0.0043	2.3742×10^{3804}
25%	6.6865×10^{6}	7.0140×10^{6}	3.1376×10^{-7}	0.9977	0.0043	2.3742×10^{3804}
50%	6.6865×10^{6}	7.0140×10^{6}	3.1376×10^{-7}	0.9977	0.0043	2.3742×10^{3804}
75%	6.6865×10^{6}	7.0140×10^{6}	3.1376×10^{-7}	0.9977	0.0043	2.3742×10^{3804}
max	6.6865×10^{6}	7.0140×10^{6}	3.1376×10^{-7}	0.9977	0.0043	2.3742×10^{3804}

Table B.25: Analysis statistics for category “inf” on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	754	754	754	754	754	754
mean	63,715	6.9753×10^5	0.0155	0.7422	0.4205	$2.4614 \times 10^{7,237,014}$
std	1.7938×10^5	1.5414×10^6	0.0684	0.1927	0.3687	$6.7587 \times 10^{7,237,015}$
min	16	46	2.0407×10^{-6}	2.0579×10^{-14}	0	1
25%	2423.5000	14,092	0.0004	0.6636	0.0021	2
50%	11,233	1.0670×10^5	0.0017	0.7986	0.4886	1.2160×10^{17}
75%	45,150	5.6556×10^5	0.0074	0.8814	0.7600	1.0694×10^{1593}
max	1.7548×10^6	1.6138×10^7	1	0.9963	1	$1.8559 \times 10^{7,237,017}$

Table B.26: Analysis statistics for category “misc” on `networkrepository.com`: 159 of the 754 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	3	3	3	3	3	3
mean	1.1770×10^5	1.0012×10^7	0.0021	0.3151	0.0547	$1.9422 \times 10^{10,036}$
std	89,524	7.2109×10^6	0.0018	0.0726	0.0563	$3.3639 \times 10^{10,036}$
min	61,989	2.8115×10^6	0.0007	0.2340	0.0010	2.2218×10^{38}
25%	66,072	6.4014×10^6	0.0011	0.2855	0.0254	2.2218×10^{38}
50%	70,155	9.9913×10^6	0.0015	0.3371	0.0498	3.3507×10^{9619}
75%	1.4556×10^5	1.3612×10^7	0.0028	0.3556	0.0816	$5.8267 \times 10^{10,036}$
max	2.2097×10^5	1.7233×10^7	0.0041	0.3741	0.1134	$5.8267 \times 10^{10,036}$

Table B.27: Analysis statistics for category “rec” on `networkrepository.com`: 0 of the 3 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	29	29	29	29	29	29
mean	6067	7893.9000	0.0005	0.7069	0.7268	$3.8621 \times 10^{29,975}$
std	3318.4000	8060.6000	0.0003	0.1953	0.1744	$2.0798 \times 10^{29,976}$
min	2139	2464	0.0002	0.1993	0.4039	6.4446×10^{1286}
25%	3698	4435	0.0003	0.6459	0.5897	9.8489×10^{2883}
50%	4904	6385	0.0005	0.7808	0.7244	1.6607×10^{4250}
75%	7974	8534	0.0007	0.8348	0.8954	$5.2621 \times 10^{12,323}$
max	18,470	48,053	0.0012	0.9053	0.9808	$1.1200 \times 10^{29,977}$

Table B.28: Analysis statistics for category “rt” on `networkrepository.com`: 0 of the 29 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	6	6	6	6	6	6
mean	1.1939×10^5	3.8734×10^5	0.0807	0.3587	0.3110	$6.7128 \times 10^{670,798}$
std	1.7313×10^5	3.1315×10^5	0.1972	0.1646	0.2735	$1.6443 \times 10^{670,799}$
min	16	58	7.7094×10^{-6}	0.1547	0	1
25%	28,824	1.5382×10^5	0.0001	0.3087	0.2056	$2.2670 \times 10^{10,415}$
50%	78,240	4.6852×10^5	0.0002	0.3147	0.2088	$2.2324 \times 10^{11,246}$
75%	81,406	4.9547×10^5	0.0005	0.3879	0.3971	$1.0998 \times 10^{12,189}$
max	4.6502×10^5	8.3354×10^5	0.4833	0.6494	0.7838	$4.0277 \times 10^{670,799}$

Table B.29: Analysis statistics for category “soc” on `networkrepository.com`: 1 of the 6 analyzed graphs is asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	1	1	1	1	1	1
mean	10,680	24,316	0.0004	0.8780	0.2562	4.4963×10^{1251}
std	nan	nan	nan	nan	nan	nan
min	10,680	24,316	0.0004	0.8780	0.2562	4.4963×10^{1251}
25%	10,680	24,316	0.0004	0.8780	0.2562	4.4963×10^{1251}
50%	10,680	24,316	0.0004	0.8780	0.2562	4.4963×10^{1251}
75%	10,680	24,316	0.0004	0.8780	0.2562	4.4963×10^{1251}
max	10,680	24,316	0.0004	0.8780	0.2562	4.4963×10^{1251}

Table B.30: Analysis statistics for category “tech” on `networkrepository.com`: 0 of the 1 analyzed graphs are asymmetric

	n	m	ρ	Q	r'_G	$\|Aut(G)\|$
count	2	2	2	2	2	2
mean	3.2564×10^{5}	1.9145×10^{6}	3.6117×10^{-5}	0.9246	0.5787	$1.9909 \times 10^{246,936}$
std	121.6200	1.1659×10^{6}	2.2017×10^{-5}	0.0170	0.0964	$2.8156 \times 10^{246,936}$
min	3.2556×10^{5}	1.0901×10^{6}	2.0549×10^{-5}	0.9126	0.5106	$7.8387 \times 10^{175,224}$
25%	3.2560×10^{5}	1.5023×10^{6}	2.8333×10^{-5}	0.9186	0.5446	$7.8387 \times 10^{175,224}$
50%	3.2564×10^{5}	1.9145×10^{6}	3.6117×10^{-5}	0.9246	0.5787	$3.9818 \times 10^{246,936}$
75%	3.2569×10^{5}	2.3268×10^{6}	4.3901×10^{-5}	0.9306	0.6128	$3.9818 \times 10^{246,936}$
max	3.2573×10^{5}	2.7390×10^{6}	5.1685×10^{-5}	0.9367	0.6468	$3.9818 \times 10^{246,936}$

Table B.31: Analysis statistics for category “web” on `networkrepository.com`: 0 of the 2 analyzed graphs are asymmetric

C Dependency of Modularity and Density

Theorem 19 (Maximum modularity in dependence of the density)**.** *The maximum modularity of a graph is bounded by its density in the form $Q \leq 1 - \rho$.*

Proof. Fortunato and Barthélemy (2007) show that a graph with maximum modularity can be constructed as a cycle of $k \geq 3$ connected and equal sized cliques. Such a graph has $m = k \cdot \binom{n'}{2} + k$ edges, where $n' \geq 2$ is the number of nodes per clique. $n' = 1$ and $k = 1$ are not valid solutions (see Section 2.5.1.1). Modularity yields

$$\begin{aligned} Q(n',k) &:= k \cdot \left[\frac{\binom{n'}{2}}{m} - \left(\frac{2\binom{n'}{2} + 2}{2m} \right)^2 \right] \\ &= \frac{k\binom{n'}{2}}{k\binom{n'}{2} + k} - k \cdot \left(\frac{\binom{n'}{2} + 1}{k(\binom{n'}{2} + 1)} \right)^2 \\ &= \frac{\binom{n'}{2}}{\binom{n'}{2} + 1} - \frac{1}{k}. \end{aligned} \tag{C.1}$$

For $k = 2$ one would find $Q_2(n') := \frac{\binom{n'}{2}}{\binom{n'}{2} + \frac{1}{2}} - \frac{1}{2}$, as $m = 2\binom{n'}{2} + 1$. With $n = k \cdot n'$ we have

$$\rho = \frac{m}{\binom{n}{2}} = \frac{k\binom{n'}{2} + k}{\frac{(kn')\cdot(kn'-1)}{2}} = \frac{\frac{n'(n'-1)}{2} + 1}{\frac{n'(kn'-1)}{2}} = \frac{n'(n'-1) + 2}{n'(kn'-1)} \tag{C.2}$$

for $k \geq 3$ and

$$\rho_2 = \frac{m}{\binom{n}{2}} = \frac{2\binom{n'}{2} + 1}{\frac{(2n')\cdot(2n'-1)}{2}} = \frac{n'(n'-1) + 1}{n'(2n'-1)} \tag{C.3}$$

for $k = 2$, respectively.

Clearly, $Q(n',k) < 1 - \frac{1}{k} =: Q'(k)$, thus $Q' \leq 1 - \rho \implies Q \leq 1 - \rho$ and if follows

$$1 - \frac{1}{k} \leq 1 - \frac{n'(n'-1) + 2}{n'(kn'-1)} \tag{C.4}$$

$$\iff \quad \frac{1}{k} \geq \frac{n'(n'-1) + 2}{n'(kn'-1)} \tag{C.5}$$

$$\iff \quad \frac{kn'^2 - n'}{k} \geq n'^2 - n' + 2 \tag{C.6}$$

$$\iff \quad -\frac{n'}{k} \geq -n' + 2 \tag{C.7}$$

$$\iff \quad \frac{n'}{n'-2} \leq k \tag{C.8}$$

which is true for $n' \geq 3$, as $\frac{n'}{n'-2} \leq 3 \leq k$.

For $n' = 2$ follows

$$Q(2,k) = \frac{1}{2} - \frac{1}{k} \leq 1 - \frac{2}{2k-1} \tag{C.9}$$

$$\iff \quad \frac{2}{2k-1} - \frac{1}{k} \leq \frac{1}{2} \tag{C.10}$$

$$\iff \quad \frac{4}{2k-1} \leq \frac{k+2}{k} \tag{C.11}$$

$$\iff \quad 4k \leq (k+2)(2k-1) \tag{C.12}$$

$$\iff \quad 0 \leq 2k^2 - k - 2, \tag{C.13}$$

which holds for $k \geq 3$.

Finally, for $k = 2$ follows (again, $Q_2(n') < 1 - \frac{1}{k} = \frac{1}{2}$)

$$Q_2(n') \leq 1 - \rho_2 \tag{C.14}$$

$$\iff \quad \frac{1}{2} \leq 1 - \frac{n'(n'-1)+1}{n'(2n'-1)} \tag{C.15}$$

$$\iff \quad \frac{n'(n'-1)+1}{n'(2n'-1)} \leq \frac{1}{2} \tag{C.16}$$

$$\iff \quad 2n'(n'-1)+2 \leq n'(2n'-1) \tag{C.17}$$

$$\iff \quad 2(n')^2 - 2n' + 2 \leq 2(n')^2 - n' \tag{C.18}$$

$$\iff \quad 2 \leq n' \tag{C.19}$$

which is true, as we argued above. □

Figure C.1 exemplarily shows the proved relation between modularity and density. However, this is not a very tight upper bound, as there are many networks for which $Q \ll 1 - \rho$.

D Algorithms

In Algorithms 3 and 4 a HASHMAP is used, which is a surjective mapping of a set of keys to their corresponding values. For a HASHMAP H, $H[k]$ returns the value v the key k maps to. Due to surjectivity, it is possible that $\exists k_1 \neq k_2 : H[k_1] = H[k_2]$. The operation $k \in H$ tests if the key k maps to some value in H.

D.1 Create Orbit Partition from Generators

Algorithm 2 shows an efficient implementation of the computation of the orbit partition from a set of generators. The idea is to color all nodes that are on the same orbit by successively iterating over the cycles of each generator, starting from a given node. Nodes on the same orbit are used as starting points in the next iteration. This procedure follows a breath-first search strategy. Coloring simply means to set a *cluster id* for each node; all nodes on the same orbit have the same color.

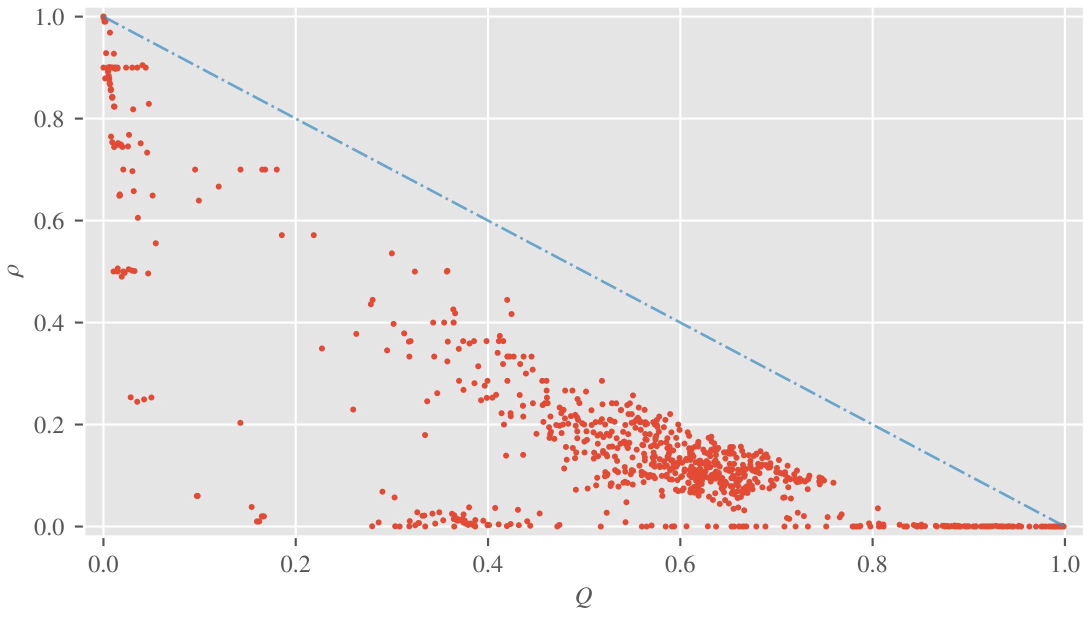


Figure C.1: Scatterplot between the modularity Q and density ρ of the simple graphs from `network-repository.com` that are analyzed. The drawn line represents the relation $\rho = 1 - Q$ for $Q \in [0, 1]$ and the points $(Q(G), \rho(G))$ lie below or onto this line for all graphs G, as a consequence of Theorem 19.

When a cycle of a permutation g is explored, for each node i on that cycle the tuple (i, g) is added to the set U to prevent that the same cycle is repeatedly explored. This allows us to give an upper bound for the time complexity of Algorithm 2.

1. The outer loop is in $\mathcal{O}(n)$ (Algorithm 2, lines 4–27).
2. Initializations of O, c, N, and U are all in $\mathcal{O}(1)$ or $\mathcal{O}(n)$ (Algorithm 2, lines 2, 3, 6, 7).
    - If a node is already colored, no work is performed (Algorithm 2, line 5).
    - If a node is uncolored, all equivalent nodes are determined by a successive search (Algorithm 2, lines 8–27, the inner loop).
3. The search starts from a given node j and iterates over all $s := |S|$ generators. It is, therefore, in $\mathcal{O}(s)$ (Algorithm 2, lines 13–25).
    - If the cycle of a permutation was already explored, it is skipped (set insertion/lookup is normally in $\mathcal{O}(1)$) (Algorithm 2, line 14).
    - Otherwise, the cycle is explored; i.e. each uncolored node is colored and added to the set of new starting points for the search (Algorithm 2, lines 15–25).
4. A permutation can consist of at most $n/2$ cycles and each is guaranteed to be explored only once, independent of the starting node i.

Combining all the facts results in an overall worst case complexity of $\mathcal{O}(s \cdot \frac{n}{2}) = \mathcal{O}(sn)$, as over time each cycle is explored at most once. This is an upper bound, as an early exit may be possible (Algorithm 2, line 26) if all nodes are already colored, even if not every cycle needed to be explored.

Algorithm 2 Compute the orbit partition O from a set of generators

Require: S is a set of generators for $Aut(G)$, i.e. $\langle S \rangle = Aut(G)$; n is the number of nodes of G

```
 1: function computeOrbitPartition($S$, $n$)
 2:     $O \leftarrow [-1, \ldots, -1]$                ▷ Initialize “empty” orbit partition array of length $n$
 3:     $c \leftarrow 0$                               ▷ Variable to count the number of colored nodes
 4:     for $i \leftarrow 0, \ldots, n-1$ do
 5:         if $O[i] \neq -1$ then continue                       ▷ Skip already colored nodes
 6:         $N \leftarrow \{i\}$                       ▷ Set to save visited nodes as new starting points
 7:         $U \leftarrow \emptyset$                   ▷ Set to save which cycles were already explored
 8:         repeat
 9:             $j \leftarrow$ pop($N$)                ▷ pop($N$) removes an arbitrary element from $N$
10:             if $O[j] = -1$ then
11:                 $O[j] \leftarrow i$                                    ▷ Use $i$ as color
12:                 $c \leftarrow c + 1$                   ▷ Increase the number of colored nodes $c$
13:             for $g \in S$ do
14:                 if $(j, g) \in U$ then continue        ▷ Do not search a cycle more than once
15:                 $U \leftarrow U \cup \{(j, g)\}$       ▷ Remember which cycle was already explored
16:                 $k \leftarrow g[j]$
17:                 while $k \neq j$ do                            ▷ Color all nodes on the cycle
18:                     if $O[k] = -1$ then
19:                         $O[k] \leftarrow i$
20:                         $c \leftarrow c + 1$
21:                         $N \leftarrow N \cup \{k\}$        ▷ Add explored node to set of starting points
22:                         $U \leftarrow U \cup \{(k, g)\}$
23:                     else
24:                         assert $O[k] = i$      ▷ All nodes on an orbit are successively colored!
25:                     $k \leftarrow g[k]$
26:             if $c = n$ then return $O$             ▷ Early exit if all nodes are already colored
27:         until $N = \emptyset$                  ▷ Repeat, as long as there are new starting points
28:     return $O$
```

It is important to understand that in Algorithm 2 all colored nodes are successively explored (lines 8–27) until no new nodes were colored. This implies that for a given node (e.g. $i = 0$ in the beginning) its complete orbit is explored and colored in this inner loop before i is increased by one in the outer loop.

D.2 Test Coarser-or-Equal for Partitions

Algorithm 3 is an efficient implementation to check if a partition P is coarser than or equal to partition Q as defined in Section 6.2.3. As the *cluster ids* are arbitrary, a hash map is used to keep track of the mappings of *cluster ids*. If the *cluster ids* also come from the domain $\{0, \ldots, n-1\}$, an array can be used instead of a hash map. The time complexity of the comparison is $\mathcal{O}(n)$, as inserting and reading from a hash map is $\mathcal{O}(1)$.

Algorithm 3 Test $P \geq Q$ for two partitions P and Q

Require: Two partitions P and Q in array representation and of the same length

```
 1: function GEQ($P$, $Q$)
 2:     $M \leftarrow$ HASHMAP( )                    ▷ A hash map to save mappings of cluster ids
 3:     $n \leftarrow |P|$
 4:     for $i \leftarrow 0, \ldots, n-1$ do
 5:         if $Q[i] \in M$ then                     ▷ Tests if there exists a value for the given key
 6:             $c \leftarrow M[Q[i]]$                         ▷ Get the already saved mapping
 7:         else
 8:             $c \leftarrow P[i]$
 9:             $M[Q[i]] \leftarrow c$
10:         if $P[i] \neq c$ then return False
11:     return True
```

D.3 Decompose Set of Generators into Support Disjoint Sets

Algorithm 4 can be used to decompose a set of generators for a permutation group into support disjoint sets of permutations. Each of these subsets generates a normal subgroup, and the composition of all these subgroups yields the input group. There are three different cases, which are discussed below.

The procedure successively builds subsets of the generating set S, which are support disjoint after each iteration of the outer loop (Algorithm 4, lines 3–17). The (intermediary) results are put into a hash map, which saves for each node the set of permutations whose support contains this node (Algorithm 4, line 2). This mapping is surjective, i.e. all nodes that are part of the same support set map to the same set of permutations. For each generator g is tested if there already exist subsets of permutations whose support overlaps $\operatorname{supp}(g)$ (Algorithm 4, line 4; also recall that we defined the support for single permutations and sets of permutations). Three different cases can occur:

Case 1 If no subsets exist, a new one that contains g is created (Algorithm 4, lines 5 and 6).

Case 2 If exactly one subset exists, g is added to it (Algorithm 4, lines 7 and 8).

Case 3 This case captures the situation that there are multiple subsets of S that have an overlapping support with g. To resolve this situation, g is added to an arbitrary subset S_i (Algorithm 4, lines 10 and 11), and for the remaining subsets, N_S is updated (Algorithm 4, lines 12–14). Then, S_i is updated to be the union of all these subsets (including S_i itself; Algorithm 4, line 15).

Algorithm 4 Decompose the set of generators S of a permutation group $\langle S \rangle = H$ into disjoint sets S_i that generate support disjoint normal subgroups $\langle S_i \rangle = H_i$, as presented by MacArthur et al. (2008).

Require: A set of generators S that is essential (MacArthur et al., 2008, p. 3527)

```
1:  function DECOMPOSE($S$)
2:      $N_S \leftarrow$ HASHMAP( )    ▷ A hash map to save mappings of nodes to sets of permutations
3:      for $g \in S$ do
4:          $\tilde{S} \leftarrow \{N_S[u] \mid u \in \operatorname{supp}(g) \wedge u \in N_S\}$    ▷ Get all sets of permutations whose support
        overlaps $\operatorname{supp}(g)$
5:          if $\tilde{S} = \varnothing$ then    ▷ Case 1
6:              $S_i \leftarrow \{g\}$
7:          else if $\tilde{S} = \{S_i\}$ then    ▷ Case 2
8:              $S_i \leftarrow S_i \cup \{g\}$
9:          else    ▷ Case 3
10:             $S_i \leftarrow$ PICK($\tilde{S}$)    ▷ PICK($\tilde{S}$) returns a random element of $\tilde{S}$
11:             $S_i \leftarrow S_i \cup \{g\}$
12:             for $S_j \in \tilde{S} \setminus S_i$ do    ▷ Update the mappings for all subsets
13:                 for $u \in \operatorname{supp}(S_j)$ do
14:                     $N_S[u] \leftarrow S_i$
15:             $S_i \leftarrow \bigcup_{S_j \in \tilde{S}} S_j$    ▷ Merge permutation sets that are not support disjoint anymore
16:         for $u \in \operatorname{supp}(g)$ do
17:             $N_S[u] \leftarrow S_i$    ▷ Set the mappings for $g$
18:     return $\{N_S[u] \mid u \in N_S\}$
```

If this last case occurs, support disjoint subsets of permutations (up to this iteration in the procedure) are recognized to be *not* support disjoint, thus they are merged.

In a last step, the mapping N_S is updated for all nodes that are part of the support of g (Algorithm 4, lines 16 and 17). When the algorithm terminates, it returns a partition of S into support disjoint subsets $S_1, \ldots, S_k$ of permutations.

E Karush-Kuhn-Tucker Conditions for Non-Linear Constrained Optimization

The Karush-Kuhn-Tucker conditions are a system of equations and inequations that must hold for every optimal point of a non-linear constrained optimization problem of the form (we loosely follow Nickel et al., 2011, pp. 269 ff.):

$$\begin{aligned} & \min f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0 \qquad \forall i \\ & h_j(\boldsymbol{x}) = 0 \qquad \forall j \end{aligned} \tag{E.1}$$

The system that needs to be solved is:

$$\begin{aligned}\nabla f(\boldsymbol{x}) + \sum_i \lambda_i \nabla g_i(\boldsymbol{x}) + \sum_j \mu_j \nabla h_j(\boldsymbol{x}) = 0 & \\ \lambda_i \geq 0 \qquad & \forall i \\ g_i(\boldsymbol{x}) \leq 0 \qquad & \forall i \\ \lambda_i g_i(\boldsymbol{x}) = 0 \qquad & \forall i \\ h_j(\boldsymbol{x}) = 0 \qquad & \forall j\end{aligned} \tag{E.2}$$

which is equivalent to

$$\begin{aligned}\nabla f(\boldsymbol{x}) + \sum_i \lambda_i \nabla g_i(\boldsymbol{x}) + \sum_j \mu_j \nabla h_j(\boldsymbol{x}) = 0 & \\ \lambda_i \geq 0 \qquad & i \in I(\boldsymbol{x}) \\ g_i(\boldsymbol{x}) = 0 \qquad & i \in I(\boldsymbol{x}) \\ g_i(\boldsymbol{x}) < 0 \qquad & i \notin I(\boldsymbol{x}) \\ h_j(\boldsymbol{x}) = 0 \qquad & \forall j\end{aligned} \tag{E.3}$$

with $I(\boldsymbol{x}) := \{i \mid g_i(\boldsymbol{x}) = 0\}$. As solutions for the two equivalent systems rely on the linearization of the non-linear problem, the solution set must satisfy regularity conditions, also known as constraint qualifications (CQ). We only mention the linearity constraint qualification (LCQ), which holds if all functions g_i/h_j, which describe the solution set, are linear.

An optimization problem is convex, if the goal function f and all constraints g_i are convex; all h_j must be linear (that implies convexity). The theorem of Karush-Kuhn-Tucker for convex optimization problems states that if there exists a point $\boldsymbol{x}$, which is a local minimum of a convex and continuously differentiable problem, and there exists another point for which all constraints g_i are strictly less than zero, then $\boldsymbol{x}$ is a KKT-point of the problem. Furthermore, if $\boldsymbol{x}$ is a KKT-point of such a problem, it is a global minimum.

List of Figures

1.1 Example of symmetric engravings on an ancient vase 2
1.2 Introductory example of a symmetry group 3
1.3 Examples for symmetry of religious symbols 6

2.1 Example of a simple graph 16
2.2 Example of a directed graph 16
2.3 Example of a weighted graph 16
2.4 Example of a multigraph 17
2.5 Example of a labeled graph 18
2.6 Example of a circular graph drawing layout 24
2.7 Example of subgraphs 26
2.8 Example of a complement graph 26
2.9 Example for a node degree histogram (PGP network) 30
2.10 Example of a Hasse diagram for partitions of four objects 37
2.11 Visualization of the λ-μ search space of a cluster's optimal modularity 53
2.12 A cluster's modularity contribution in dependence of λ and μ 54
2.13 The network `rt_obama` from `networkrepository.com` as example for clustering . 56

3.1 Example of isomorphic graphs 63
3.2 Example for an automorphic graph 65
3.3 Example for graph simplification 67
3.4 Example of an asymmetric graph with the minimum number of edges 70
3.5 Example of the embedding of K_5 onto the surface of a torus 74
3.6 The symmetric butterfly graph B 80
3.7 Example of a search tree to find automorphisms of the butterfly graph 81
3.8 Example of search tree pruning while finding automorphisms 83
3.9 Example of a graph with pseudo-similar nodes 90

4.1 Graph analysis: Histogram and box-plots for n and m 101
4.2 Graph analysis: Histogram and box-plots for ρ 102
4.3 Graph analysis: Histogram and box-plots for Q 103
4.4 Graph analysis: Histogram and box-plot for r'_G 104
4.5 Graph analysis: Scatter-plot for r'_G and the network type/category 105
4.6 Graph analysis (simplifed): Histogram and box-plots for n and m 107

4.7 Graph analysis (simplified): Histogram and box-plots for ρ 108
4.8 Graph analysis (simplified): Histogram and box-plots for Q 108
4.9 Graph analysis (simplified): Histogram and box-plot for r'_G 109
4.10 Graph analysis (simplified): Scatter-plot for r'_G and the network type/category 110

5.1 Example for symmetry by chance in the Karate network 116
5.2 Example of subgraphs induced by indecomposable subgroups 119
5.3 Example for local graph symmetry 121
5.4 Example for globality of graph symmetry 121
5.5 Example of a partition induced subgraph 124

6.1 Example graph to illustrate the space of partition equivalence classes 126
6.2 Example diagram of the symmetry induced partition equivalence classes 126
6.3 Example of partition stability using the refinement lattice definition 130
6.4 Illustration of a star graph as part of a larger network 134
6.5 Illustration of a complete bipartite graph as part of a larger network 136
6.6 The modularity optimal partition of the Karate network 142
6.7 A suboptimal partition of the Karate network that is affected by symmetry 143
6.8 Example for the KSE of a graph partition 151
6.9 Example for the consideration of dependent orbits for the KSE 153
6.10 Example of a dynamical system with continuous state-space 156
6.11 Example for the simplified computation of KSE if only one orbit exists 158
6.12 Example for the simplified computation of KSE if two orbits exists 159
6.13 Example for the simplified computation of KSE of the Petersen graph 160

7.1 Comparison of the distributions of rs_G and $\overline{gs}_G$ 168
7.2 Symmetry impact: Histogram and box-plot for KSE 169
7.3 Symmetry impact: Histogram and box-plot for $\frac{|\tilde{\Pi}_{inter}|}{|S|}$ 169
7.4 Relations between average cluster size and average/max support 171
7.5 Symmetry impact: Histogram and box-plot for $\overline{gs}_G$, distinguished by stability 173
7.6 Relations between average cluster size and average/max support (unstable partitions) 177
7.7 DBSCAN clustering result for graphs with unstable partitions 180

A.1 Example for a highly symmetric graph with intermediate r'_G 189
C.1 Scatterplot of Q and ρ to illustrate the relation $Q \leq 1 - \rho$ 208

List of Tables

2.1 Permutation catenation example 43
2.2 Cayley table example 43

3.1 The first ten values for the number of asymmetric trees having n nodes 69
3.2 Three different problem manifestations of GI and GA 72

4.1 Number of datasets for different network types on `networkrepository.com` 98
4.2 Statistics for the `networkrepository.com` datasets 100
4.3 Analysis statistics for `networkrepository.com` datasets 106

6.1 Blocks that occur with positive probability for certain k for the given symbol stream. 146
6.2 Observed entropy for a simulated symbol stream on two symbols 146
6.3 Theoretical k-block probabilities of a random binary stream 147

7.1 Overview of different symmetry impact measures 165
7.2 Partition stability statistics for `networkrepository.com` datasets 167
7.3 Test results of hypotheses $H_{1.1}$ to $H_{1.4}$ and H_2 172
7.4 Test results of hypotheses $H_{3.1}$ and $H_{3.2}$ and their inverses 174
7.5 Confusion matrices for both naïve classifiers 175
7.6 Computed properties of simple graphs with unstable partitions 178
7.7 Statistics per cluster (DBSCAN) for unstable partitions 182
7.8 Overview of the four clusters of unstable partitions 183
7.9 Confusion matrices for both refined naïve classifiers 183

B.1 Matrix Market exchange format overview 192
B.2 Analysis statistics for category “bhoslib” on `networkrepository.com` (simple) . . 197
B.3 Analysis statistics for category “bio” on `networkrepository.com` (simple) 198
B.4 Analysis statistics for category “ca” on `networkrepository.com` (simple) 198
B.5 Analysis statistics for category “chem” on `networkrepository.com` (simple) . . . 198
B.6 Analysis statistics for category “dimacs” on `networkrepository.com` (simple) . . 198
B.7 Analysis statistics for category “dimacs10” on `networkrepository.com` (simple) . 199
B.8 Analysis statistics for category “ia” on `networkrepository.com` (simple) 199
B.9 Analysis statistics for category “inf” on `networkrepository.com` (simple) 199
B.10 Analysis statistics for category “misc” on `networkrepository.com` (simple) . . . 199
B.11 Analysis statistics for category “rec” on `networkrepository.com` (simple) 200

B.12 Analysis statistics for category “rt” on `networkrepository.com` (simple) 200
B.13 Analysis statistics for category “sc” on `networkrepository.com` (simple) 200
B.14 Analysis statistics for category “soc” on `networkrepository.com` (simple) 200
B.15 Analysis statistics for category “socfb” on `networkrepository.com` (simple) . . . 201
B.16 Analysis statistics for category “tech” on `networkrepository.com` (simple) 201
B.17 Analysis statistics for category “tscc” on `networkrepository.com` (simple) 201
B.18 Analysis statistics for category “web” on `networkrepository.com` (simple) 201
B.19 Analysis statistics for category “bio” on `networkrepository.com` (simplified) . . 202
B.20 Analysis statistics for category “bn” on `networkrepository.com` (simplified) . . . 202
B.21 Analysis statistics for category “dimacs10” on `networkrepository.com` (simplified)202
B.22 Analysis statistics for category “dynamic” on `networkrepository.com` (simplified)203
B.23 Analysis statistics for category “eco” on `networkrepository.com` (simplified) . . 203
B.24 Analysis statistics for category “ia” on `networkrepository.com` (simplified) . . . 203
B.25 Analysis statistics for category “inf” on `networkrepository.com` (simplified) . . . 203
B.26 Analysis statistics for category “misc” on `networkrepository.com` (simplified) . . 204
B.27 Analysis statistics for category “rec” on `networkrepository.com` (simplified) . . 204
B.28 Analysis statistics for category “rt” on `networkrepository.com` (simplified) . . . 204
B.29 Analysis statistics for category “soc” on `networkrepository.com` (simplified) . . 204
B.30 Analysis statistics for category “tech” on `networkrepository.com` (simplified) . . 205
B.31 Analysis statistics for category “web” on `networkrepository.com` (simplified) . . 205

Bibliography

Aaronson, S., Kuperberg, G., and Granade, C. (2017) Complexity zoo. Accessed May 11, 2017, URL `https://complexityzoo.uwaterloo.ca`.

Adai, A.T., Date, S.V., Wieland, S., and Marcotte, E.M. (2004) LGL: Creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal of Molecular Biology*, 340(1), pp. 179–190, doi:10.1016/j.jmb.2004.04.047.

Agrawal, R. and Srikant, R. (1994) Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Databases Conference*, VLDB ’94, pp. 487–499, Morgan Kaufmann Publishers Inc., San Francisco, URL `http://www.vldb.org/conf/1994/P487.PDF`.

Aho, A.V., Hopcroft, J.E., and Ullmann, J.D. (1974) *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, first edition.

Albert, R. and Barabási, A.L. (2002) Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1), pp. 47–97, doi:10.1103/RevModPhys.74.47.

Anand, K. and Bianconi, G. (2009) Entropy measures for networks: Toward an information theory of complex topologies. *Physical Review E*, 80(4), pp. 045102:1–4, doi:10.1103/PhysRevE.80.045102.

Andrews, G.E. (1998) Partitions in combinatorics. In *The Theory of Partitions*, Cambridge Mathematical Library, pp. 212–229, Cambridge University Press, New York, doi:10.1017/CBO9780511608650.016.

Aschbacher, M. (2004) The status of the classification of the finite simple groups. *Notices of the American Mathematical Society*, 51(7), pp. 736–740, URL `https://www.ams.org/journals/notices/200407/fea-aschbacher.pdf`.

Babai, L. (1979) Monte-Carlo algorithms in graph isomorphism testing. Technical Report 79–10, Université de Montréal Technical Report, DMS, Montreal, URL `http://cs.uchicago.edu/~laci/lasvegas79.pdf`.

Babai, L. (2016a) Graph isomorphism in quasipolynomial time. *arXiv:1512.03547v2 [cs, math]*, `1512.03547v2`, URL `https://arxiv.org/abs/1512.03547v2`.

Babai, L. (2016b) Graph isomorphism in quasipolynomial time [Extended abstract]. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pp. 684–697, ACM, New York, NY, USA, doi:10.1145/2897518.2897542.

Babai, L. and Luks, E.M. (1983) Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pp. 171–183, ACM, New York, NY, USA, doi:10.1145/800061.808746.

Bader, D.A., Meyerhenke, H., Sanders, P., Schulz, C., Kappes, A., and Wagner, D. (2014) Benchmarking for graph clustering and partitioning. In Alhajj, R. and Rokne, J. (editors) *Encyclopedia of Social Network Analysis and Mining*, pp. 73–82, Springer, New York, doi:10.1007/978-1-4614-6170-8_23.

Bailey, D.H. and Borwein, J. (2016) Pi day is upon us again and we still do not know if pi is normal (2014). In *Pi: The Next Generation*, pp. 425–442, Springer, Cham, doi:10.1007/978-3-319-32377-0_23.

Bairagi, V.K. (2015) Symmetry-based biomedical image compression. *Journal of Digital Imaging*, 28(6), pp. 718–726, doi:10.1007/s10278-015-9779-3.

Balaban, A.T. (1985) Applications of graph theory in chemistry. *Journal of Chemical Information and Computer Sciences*, 25(3), pp. 334–343, doi:10.1021/ci00047a033.

Ball, F. and Geyer-Schulz, A. (2017) Weak invariants of actions of the automorphism group of a graph. *Archives of Data Science, Series A*, 2(1), pp. 123–144, doi:10.5445/KSP/1000058749/02.

Ball, F. and Geyer-Schulz, A. (2018a) How symmetric are real-world graphs? A large-scale study. *Symmetry*, 10(1), pp. 29:1–17, doi:10.3390/sym10010029.

Ball, F. and Geyer-Schulz, A. (2018b) Invariant graph partition comparison measures. *Symmetry*, 10(10), pp. 504:1–24, doi:10.3390/sym10100504.

Ball, F. and Geyer-Schulz, A. (2018c) Symmetry-based graph clustering partition stability. *Archives of Datascience, Series A*, 4(1), pp. 1–21, doi:10.5445/KSP/1000085951/01.

Barabási, A.L. and Albert, R. (1999) Emergence of scaling in random networks. *Science*, 286(5439), pp. 509–512, doi:10.1126/science.286.5439.509.

Basak, S.C., Niemi, G.J., and Veith, G.D. (1991) Predicting properties of molecules using graph invariants. *Journal of Mathematical Chemistry*, 7(1), pp. 243–272, doi:10.1007/BF01200826.

Bastian, M., Heymann, S., and Jacomy, M. (2009) Gephi: An open source software for exploring and manipulating networks. In *Third International AAAI Conference on Weblogs and Social Media*, pp. 361–362, URL `http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154`.

Ben-David, S., von Luxburg, U., and Pál, D. (2006) A sober look at clustering stability. In *Learning Theory*, pp. 5–19, Springer, Berlin, Heidelberg, doi:10.1007/11776420_4.

Ben-Hur, A., Elisseeff, A., and Guyon, I. (2001) A stability based method for discovering structure in clustered data. In *Biocomputing 2002*, pp. 6–17, WORLD SCIENTIFIC, doi:10.1142/9789812799623_0002.

Beth, T., Jungnickel, D., and Lenz, H. (1993) Groups and designs. In *Design Theory*, pp. 142–195, Cambridge University Press, Cambridge, first edition, doi:10.1017/CBO9780511549533.003, reprint, first published 1985.

Bianconi, G. (2009) Entropy of network ensembles. *Physical Review E*, 79(3), pp. 036114:1–10, doi:10.1103/PhysRevE.79.036114.

Biemann, C. (2006) Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, pp. 73–80, Association for Computational Linguistics, Stroudsburg, PA, USA, URL `http://dl.acm.org/citation.cfm?id=1654774`.

Biggs, N.L. (1993) *Algebraic Graph Theory*. Cambridge Mathematic Library, Cambridge University Press, Cambridge, second edition, doi:10.1017/CBO9780511608704.

Birkhoff, G.D. (1931) Proof of the ergodic theorem. *Proceedings of the National Academy of Sciences of the United States of America*, 17, pp. 656–660, doi:10.1073/pnas.17.2.656.

Blondel, V.D., Guillaume, J.L., Lambiotte, R., and Lefebvre, E. (2008) Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), pp. 1008:1–12, doi:10.1088/1742-5468/2008/10/P10008.

Boguñá, M., Pastor-Satorras, R., Díaz-Guilera, A., and Arenas, A. (2004) Models of social networks based on social distance attachment. *Physical Review E*, 70(5), pp. 056122:1–8, doi:10.1103/PhysRevE.70.056122.

Boisvert, R.F., Pozo, R., and Remington, K.A. (1996) The matrix market exchange formats: Initial design. Technical Report NISTIR 5935, National Institute of Standards and Technology, Applied and Computational Mathematics Division, Gaithersburg, MD, URL `https://math.nist.gov/MatrixMarket/reports/MMformat.ps`.

Boltzmann, L. (1896) *Vorlesungen über Gastheorie*, volume 1. Barth, Leipzig, first edition, URL `http://www.deutschestextarchiv.de/book/show/boltzmann_gastheorie01_1896`.

Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D., and Ferro, A. (2013) A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 14(Suppl 7), pp. S13:1–13, doi:10.1186/1471-2105-14-S7-S13.

Booth, K.S. and Colbourn, C.J. (1979) Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, University of Waterloo, Departement of Computer Science, Waterloo, Ontario, Canada, URL `https://cs.uwaterloo.ca/research/tr/1977/CS-77-04.pdf`.

Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S.V.N., Smola, A.J., and Kriegel, H.P. (2005) Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1), pp. i47–i56, doi:10.1093/bioinformatics/bti1007.

Brading, K. and Castellani, E. (2007) Symmetries and invariances in classical physics. In Butterfield, J. and Earman, J. (editors) *Philosophy of Physics*, Handbook of the Philosophy of Science, pp. 1331–1367, North-Holland, Amsterdam, doi:10.1016/B978-044451560-5/50016-6.

Brading, K., Castellani, E., and Teh, N. (2003) Symmetry and symmetry breaking. Accessed April 19, 2018, URL `https://plato.stanford.edu/archives/win2017/entries/symmetry-breaking/`.

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008) On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2), pp. 172–188, doi:10.1109/TKDE.2007.190689.

Brandstädt, A., Le, V.B., and Spinrad, J.P. (1999) *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, PA, USA, doi:10.1137/1.9780898719796.

Brouwer, A.E. and Haemers, W.H. (2012) *Spectra of Graphs*. Universitext, Springer, New York, doi:10.1007/978-1-4614-1939-6.

Brualdi, R.A. (1988) Some applications of doubly stochastic matrices. *Linear Algebra and its Applications*, 107, pp. 77–100, doi:10.1016/0024-3795(88)90239-X.

Butler, G. (1991) Schreier-Sims method. In *Fundamental Algorithms for Permutation Groups*, *Lecture Notes in Computer Science*, volume 559, pp. 129–142, Springer, Berlin, Heidelberg, doi:10.1007/3-540-54955-2_32.

Cameron, P.J. (1983) Automorphism groups of graphs. In *Selected Topics in Graph Theory*, volume 2, pp. 89–127, Academic Press, London.

Campbell, D.M. and Radford, D. (1991) Tree isomorphism algorithms: Speed vs. clarity. *Mathematics Magazine*, 64(4), pp. 252–261, doi:10.2307/2690833.

Cayley, A. and Forsyth, A.R. (1889) *The Collected Mathematical Papers of Arthur Cayley*, volume 2. Cambridge University Press, Cambridge, URL `http://archive.org/details/collmathpapers02caylrich`.

Chee, Y.M. and Ling, S. (2002) Highly symmetric expanders. *Finite Fields and Their Applications*, 8(3), pp. 294–310, doi:10.1006/ffta.2001.0341.

Chen, J. (1992) A linear time algorithm for isomorphism of graphs of bounded average genus. In *Graph-Theoretic Concepts in Computer Science*, *Lecture Notes in Computer Science*, volume 657, pp. 103–113, Springer, Berlin, Heidelberg, doi:10.1007/3-540-56402-0_40.

Cherepnalkoski, D. and Mozetič, I. (2016) Retweet networks of the European Parliament: Evaluation of the community structure. *Applied Network Science*, 1(1), pp. 2:1–20, doi:10.1007/s41109-016-0001-4.

Choe, G.H. and Kim, D.H. (2000) Entropy and the randomness of the digits of pi. *Communications of the Korean Mathematical Society*, 15(4), pp. 683–689, URL `http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=DBSHCJ_2000_v15n4_683`.

Christodoulakis, T., Karagiorgos, A., and Zampeli, A. (2018) Symmetries in classical and quantum treatment of Einstein's cosmological equations and mini-superspace actions. *Symmetry*, 10(3), pp. 70:1–17, doi:10.3390/sym10030070.

Chung, F., Lu, L., Dewey, T.G., and Galas, D.J. (2003) Duplication models for biological networks. *Journal of Computational Biology*, 10(5), pp. 677–687, doi:10.1089/106652703322539024.

Chung, F.R.K. (1997) *Spectral Graph Theory*. Number 92 in Regional Conference Series in Mathematics, American Mathematical Society Press, Providence, first edition, doi:10.1090/cbms/092.

Clauset, A., Shalizi, C., and Newman, M. (2009) Power-law distributions in empirical data. *SIAM Review*, 51(4), pp. 661–703, doi:10.1137/070710111.

Clausius, R. (1867) *The Mechanical Theory of Heat: With Its Applications to the Steam-Engine and to the Physical Properties of Bodies*. John Van Voorst, London, URL `https://babel.hathitrust.org/cgi/pt?id=hvd.32044009771759`.

Colbourn, C.J. (1981) On testing isomorphism of permutation graphs. *Networks*, 11(1), pp. 13–21, doi:10.1002/net.3230110103.

Conte, D., Foggia, P., Sansone, C., and Vento, M. (2003) Graph matching applications in pattern recognition and image processing. In *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, volume 2, pp. II–21–24, doi:10.1109/ICIP.2003.1246606.

Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004) Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(03), pp. 265–298, doi:10.1142/S0218001404003228.

Cordella, L.P., Foggia, P., Sansone, C., and Vento, M. (1999) Performance evaluation of the VF graph matching algorithm. In *Proceedings 10th International Conference on Image Analysis and Processing*, pp. 1172–1177, doi:10.1109/ICIAP.1999.797762.

Cordella, L.P., Foggia, P., Sansone, C., and Vento, M. (2001) An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-Based Representations in Pattern Recognition*, pp. 149–159, Cuen.

Cordella, L.P., Foggia, P., Sansone, C., and Vento, M. (2004) A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10), pp. 1367–1372, doi:10.1109/TPAMI.2004.75.

Corneil, D. and Kirkpatrick, D. (1980) A theoretical analysis of various heuristics for the graph isomorphism problem. *SIAM Journal on Computing*, 9(2), pp. 281–297, doi:10.1137/0209025.

Corneil, D.G. and Gotlieb, C.C. (1970) An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17(1), pp. 51–64, doi:10.1145/321556.321562.

Coudène, Y. (2016) *Ergodic Theory and Dynamical Systems*. Universitext, Springer, London, doi:10.1007/978-1-4471-7287-1.

Darga, P.T., Liffiton, M.H., Sakallah, K.A., and Markov, I.L. (2004) Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pp. 530–534, ACM, San Diego, CA, USA, doi:10.1145/996566.996712.

Darga, P.T., Sakallah, K.A., and Markov, I.L. (2008) Faster symmetry discovery using sparsity of symmetries. In *2008 45th ACM/IEEE Design Automation Conference*, pp. 149–154, doi:10.1145/1391469.1391509.

Darvas, G. (2017) Interdisciplinary application of symmetry phenomena. In *Aesthetics of Interdisciplinarity: Art and Mathematics*, pp. 81–101, Birkhäuser, Cham, doi:10.1007/978-3-319-57259-8_5.

Dehmer, M., Grabner, M., Mowshowitz, A., and Emmert-Streib, F. (2013) An efficient heuristic approach to detecting graph isomorphism based on combinations of highly discriminating invariants. *Advances in Computational Mathematics*, 39(2), pp. 311–325, doi:10.1007/s10444-012-9281-0.

Dehmer, M. and Mowshowitz, A. (2011a) Generalized graph entropies. *Complexity*, 17(2), pp. 45–50, doi:10.1002/cplx.20379.

Dehmer, M. and Mowshowitz, A. (2011b) A history of graph entropy measures. *Information Sciences*, 181(1), pp. 57–78, doi:10.1016/j.ins.2010.08.041.

Dixon, J.D. and Mortimer, B. (1996) *Permutation Groups*. Number 163 in Graduate Texts in Mathematics, Springer, New York.

van Dongen, S. (2018) Human bodily asymmetry relates to behavioral lateralization and may not reliably reflect developmental instability. *Symmetry*, 10(4), pp. 117:1–7, doi:10.3390/sym10040117.

Einsiedler, M. and Ward, T. (2011) *Ergodic Theory*. Number 259 in Graduate Texts in Mathematics, Springer, London, doi:10.1007/978-0-85729-021-2.

Elmsallati, A., Clark, C., and Kalita, J. (2016) Global alignment of protein-protein interaction networks: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 13(4), pp. 689–705, doi:10.1109/TCBB.2015.2474391.

Emerson, E.A. and Trefler, R.J. (1999) From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Correct Hardware Design and Verification Methods*, Lecture Notes in Computer Science, pp. 142–157, Springer, Berlin, Heidelberg, doi:10.1007/3-540-48153-2_12.

Erdős, P. and Rényi, A. (1957) On random graphs I. *Publicationes Mathematicae Debrecen*, 6, pp. 290–297, URL `https://users.renyi.hu/~p_erdos/1959-11.pdf`.

Erdős, P. and Rényi, A. (1960) On the evolution of random graphs. *Publications of the Hungarian Academy of Sciences*, 5(1), pp. 17–61, URL `https://users.renyi.hu/~p_erdos/1960-10.pdf`.

Erdős, P. and Rényi, A. (1963) Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3), pp. 295–315, doi:10.1007/BF01895716.

Ester, M., Kriegel, H.P., Sander, J., and Xu, X. (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, AAAI Press, Portland, OR.

Evdokimov, S. and Ponomarenko, I. (2004) Circulant graphs: Recognizing and isomorphism testing in polynomial time. *St. Petersburg Mathematical Journal*, 15(6), pp. 813–835, doi:10.1090/S1061-0022-04-00833-7.

Fan, W. (2012) Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT ’12, pp. 8–21, ACM, New York, NY, USA, doi:10.1145/2274576.2274578.

Feynman, R.P., Leighton, R.B., and Sands, M. (1963) *Mainly Mechanics, Radiation and Heat, The Feynman Lectures on Physics.*, volume 1. Addison-Wesley, Reading.

Foggia, P., Sansone, C., and Vento, M. (2001) A performance comparison of five algorithms for graph isomorphism. In *In Proceedings of the 3rd IAPR TC-15 Workshop on Graph-Based Representations in Pattern Recognition*, pp. 188–199.

Fortin, S. (1996) The graph isomorphism problem. Technical Report TR 96-20, University of Alberta, Edmonton, Alberta, Canada, doi:10.7939/R3SX64C5K.

Fortunato, S. (2010) Community detection in graphs. *Physics Reports*, 486(3–5), pp. 75–174, doi:10.1016/j.physrep.2009.11.002.

Fortunato, S. and Barthélemy, M. (2007) Resolution limit in community detection. *Proceedings of the National Academy of Sciences of the United States of America*, 104(1), pp. 36–41, doi:10.1073/pnas.0605965104.

Fortunato, S. and Castellano, C. (2012) Community structure in graphs. In Meyers, R.A. (editor) *Computational Complexity*, pp. 490–512, Springer, New York, NY, doi:10.1007/978-1-4614-1800-9_33.

van Fraassen, B.C. (1989) *Laws and Symmetry*. Oxford University Press, Oxford, doi:10.1093/0198248601.001.0001.

Frigg, R. (2004) In what sense is the Kolmogorov-Sinai entropy a measure for chaotic behaviour? Bridging the gap between dynamical systems theory and communication theory. *The British Journal for the Philosophy of Science*, 55(3), pp. 411–434, doi:10.1093/bjps/55.3.411.

Frucht, R. (1939) Herstellung von Graphen mit vorgegebener abstrakter Gruppe. *Compositio Mathematica*, 6, pp. 239–250, URL `http://eudml.org/doc/88709`.

Fruchterman, T.M.J. and Reingold, E.M. (1991) Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11), pp. 1129–1164, doi:10.1002/spe.4380211102.

Garlaschelli, D., Ruzzenenti, F., and Basosi, R. (2010) Complex networks and symmetry I: A review. *Symmetry*, 2(3), pp. 1683–1709, doi:10.3390/sym2031683.

Garrido, A. (2011) Symmetry in complex networks. *Symmetry*, 3(1), pp. 1–15, doi:10.3390/sym3010001.

Gary, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

Geyer-Schulz, A. (1989) Memo. *APL Quote Quad*, 20(2), pp. 12–27, doi:10.1145/379209.379211.

Geyer-Schulz, A. and Ovelgönne, M. (2014) The randomized greedy modularity clustering algorithm and the core groups graph clustering scheme. In Gaul, W., Geyer-Schulz, A., Baba, Y., and Okada, A. (editors) *German-Japanese Interchange of Data Analysis Results*, Studies in Classification, Data Analysis, and Knowledge Organization, pp. 17–36, Springer, Cham, doi:10.1007/978-3-319-01264-3_2.

Gfeller, D., Chappelier, J.C., and De Los Rios, P. (2005) Finding instabilities in the community structure of complex networks. *Physical Review E*, 72(5), pp. 056135:1–6, doi:10.1103/PhysRevE.72.056135.

Gibbs, J.W. (1873) A method of geometrical representation of the thermodynamic properties of substances by means of surfaces. *Transactions of the Connecticut Academy of Arts and Sciences*, II, pp. 382–404, URL `https://babel.hathitrust.org/cgi/pt?id=hvd.32044106253321`.

Godsil, C.D. and Royle, G. (2001) *Algebraic Graph Theory*. Number 207 in Graduate Texts in Mathematics, Springer, New York, doi:10.1007/978-1-4613-0163-9.

Grochow, J.A. and Kellis, M. (2007) Network motif discovery using subgraph enumeration and symmetry-breaking. In *Research in Computational Molecular Biology*, *Lecture Notes in Computer Science*, volume 4453, pp. 92–106, Springer, Berlin, Heidelberg, doi:10.1007/978-3-540-71681-5_7.

Grohe, M. (2010) Fixed-point definability and polynomial time on graphs with excluded minors. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*, pp. 179–188, IEEE Computer Society, doi:10.1109/LICS.2010.22.

Gross, D.J. (1996) The role of symmetry in fundamental physics. *Proceedings of the National Academy of Sciences*, 93(25), pp. 14256–14259, doi:10.1073/pnas.93.25.14256.

Gross, J.L. and Tucker, T.W. (2001) *Topological Graph Theory*. Dover Publication, New York, reprint of the original version of 1987 (Wiley, New York).

Harary, F. and Palmer, E. (1966) On similar points of a graph. *Journal of Mathematics and Mechanics*, 15(4), pp. 623–630, doi:10.1512/iumj.1966.15.15042.

Harary, F. and Prins, G. (1959) The number of homeomorphically irreducible trees, and other species. *Acta Mathematica*, 101(1-2), pp. 141–162, doi:10.1007/BF02559543.

Harenberg, S., Bello, G., Gjeltema, L., Ranshous, S., Harlalka, J., Seay, R., Padmanabhan, K., and Samatova, N. (2014) Community detection in large-scale networks: A survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6), pp. 426–439, doi:10.1002/wics.1319.

Hartke, S.G. and Radcliffe, A.J. (2009) Mckay's canonical graph labeling algorithm. In *Communicating Mathematics*, number 479 in Contemporary Mathematics, pp. 99–111, American Mathematical Society, Providence, Rhode Island, URL `http://www.math.unl.edu/%7Earadcliffe1/Papers/Canonical.pdf`.

Hasselberg, J., Pardalos, P.M., and Vairaktarakis, G. (1993) Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization*, 3(4), pp. 463–482, doi:10.1007/BF01096415.

Helmberg, C. and Rendl, F. (2000) A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3), pp. 673–696, doi:10.1137/S1052623497328987.

Holt, D.F., Eick, B., and O'Brien, E.A. (2005) *Handbook of Computational Group Theory*. Discrete Mathematics and Its Application, Chapman & Hall/CRC, Boca Raton.

Hon, G. and Goldstein, B.R. (2006) Unpacking "for reasons of symmetry": Two categories of symmetry arguments. *Philosophy of Science*, 73(4), pp. 419–439, doi:10.1086/516811.

Hoory, S., Linial, N., and Wigderson, A. (2006) Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4), pp. 439–561.

Hopcroft, J.E. and Wong, J.K. (1974) Linear Time Algorithm for Isomorphism of Planar Graphs (preliminary Report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pp. 172–184, ACM, New York, NY, USA, doi:10.1145/800119.803896.

Hu, C., Fakhri, G.E., and Li, Q. (2014) Evaluating structural symmetry of weighted brain networks via graph matching. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2014*, Lecture Notes in Computer Science, pp. 733–740, Springer, Cham, doi:10.1007/978-3-319-10470-6_91.

Itai, A. and Rodeh, M. (1990) Symmetry breaking in distributed networks. *Information and Computation*, 88(1), pp. 60–87, doi:10.1016/0890-5401(90)90004-2.

Jabbour, S., Khiari, M., Sais, L., Salhi, Y., and Tabia, K. (2013) Symmetry-based pruning in itemset mining. In *Proceedings of the 25th International Conference on Tools with Artificial Intelligence*, pp. 483–490, IEEE Computer Society, doi:10.1109/ICTAI.2013.78.

Jain, B.J. and Wysotzki, F. (2005) Solving inexact graph isomorphism problems using neural networks. *Neurocomputing*, 63, pp. 45–67, doi:10.1016/j.neucom.2004.01.189.

Jiang, K.Z., Zheng, Z.T., and Li, L. (2017) Topological structure matching measure between two graphs. *Computer-Aided Civil and Infrastructure Engineering*, 32(6), pp. 515–524, doi:10.1111/mice.12270.

Johnson, D.S. (1990) A catalog of complexity classes. In *Algorithms and Complexity*, *Handbook of Theoretical Computer Science*, volume A, pp. 67–161, Elsevier, Amsterdam, New York, Oxford, Tokyo, first edition.

Junttila, T. and Kaski, P. (2007) Engineering an efficient canonical labeling tool for large and sparse graphs. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 135–149, SIAM, doi:10.1137/1.9781611972870.13.

Junttila, T. and Kaski, P. (2011) Conflict propagation and component recursion for canonical labeling. In *Theory and Practice of Algorithms in (Computer) Systems*, *Lecture Notes in Computer Science*, volume 6595, pp. 151–162, Springer, Berlin, Heidelberg, doi:10.1007/978-3-642-19754-3_16.

Kamada, T. and Kawai, S. (1989) An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1), pp. 7–15, doi:10.1016/0020-0190(89)90102-6.

Kang, M. and Petrášek, Z. (2014) Random graphs: Theory and applications from nature to society to the brain. *Internationale Mathematische Nachrichten*, (227), pp. 1–24.

Karrer, B., Levina, E., and Newman, M.E.J. (2008) Robustness of community structure in networks. *Physical Review E*, 77(4), p. 46119, doi:10.1103/PhysRevE.77.046119.

Karush, W. (2014) Minima of functions of several variables with inequalities as side conditions. In *Traces and Emergence of Nonlinear Programming*, pp. 217–245, Birkhäuser, Basel, doi:10.1007/978-3-0348-0439-4_10.

Katebi, H., Sakallah, K.A., and Markov, I.L. (2012) Conflict anticipation in the search for graph automorphisms. In Bjørner, N. and Voronkov, A. (editors) *Logic for Programming, Artificial Intelligence, and Reasoning*, number 7180 in Lecture Notes in Computer Science, pp. 243–257, Springer, Berlin, Heidelberg, doi:10.1007/978-3-642-28717-6_20.

Kehagias, A. and Pitsoulis, L. (2013) Bad communities with high modularity. *The European Physical Journal B*, 86(7), pp. 330:1–11, doi:10.1140/epjb/e2013-40169-1.

Khalifeh, M.H., Yousefi-Azari, H., Ashrafi, A.R., and Wagner, S.G. (2009) Some new results on distance-based graph invariants. *European Journal of Combinatorics*, 30(5), pp. 1149–1163, doi:10.1016/j.ejc.2008.09.019.

Kim, J. and Wilhelm, T. (2008) What is a complex graph? *Physica A: Statistical Mechanics and its Applications*, 387(11), pp. 2637–2652, doi:10.1016/j.physa.2008.01.015.

Kimble, R.J., Schwenk, A.J., and Stockmeyer, P.K. (1981) Pseudosimilar vertices in a graph. *Journal of Graph Theory*, 5(2), pp. 171–181, doi:10.1002/jgt.3190050207.

Knauer, U. (2011) *Algebraic Graph Theory: Morphisms, Monoids and Matrices*. Number 41 in De Gruyter Studies in Mathematics, De Gruyter, Berlin, URL `http://lib.myilibrary.com/Open.aspx?id=340044`.

Knuth, D.E. (1997) *Seminumerical Algorithms*, *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, third edition.

Kuhn, H.W. and Tucker, A.W. (1951) Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492, University of California Press, Berkeley, CA, URL `http://projecteuclid.org/euclid.bsmsp/1200500249`.

Kukluk, J.P., Holder, L.B., and Cook, D.J. (2004) Algorithm and experiments in testing planar graphs for isomorphism. *Journal of Graph Algorithms and Applications*, 8(3), pp. 313–356, doi:10.7155/jgaa.00094.

Kunegis, J. (2013) KONECT: The Koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pp. 1343–1350, ACM, Rio de Janeiro, Brazil, doi:10.1145/2487788.2488173.

Kunegis, J. (2014) Handbook of network analysis [KONECT - the Koblenz network collection]. *arXiv:1402.5500v3 [cs.SI]*, `1402.5500v3`, URL `http://arxiv.org/abs/1402.5500v3`.

van Laarhoven, T. and Marchiori, E. (2013) Graph clustering with local search optimization: The resolution bias of the objective function matters most. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, 87(1), pp. 012812:1–11, doi:10.1103/PhysRevE.87.012812.

Lakshmivarahan, S., Jwo, J.S., and Dhall, S.K. (1993) Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey. *Parallel Computing*, 19(4), pp. 361–407, doi:10.1016/0167-8191(93)90054-O.

Lancichinetti, A. and Fortunato, S. (2011) Limits of modularity maximization in community detection. *Physical Review E*, 84(6), pp. 066122:1–8, doi:10.1103/PhysRevE.84.066122.

Lauri, J. and Scapellato, R. (2016) *Topics in Graph Automorphisms and Reconstruction*. Number 432 in London Mathematical Society Lecture Notes Series, Cambridge University Press, Cambridge, second edition, doi:10.1017/CBO9781316669846.

Leskovec, J. and Krevl, A. (2014) SNAP datasets: Stanford large network dataset collection. Accessed August 21, 2018, URL `http://snap.stanford.edu/data`.

Li, F., Shang, H., and Woo, P.Y. (2008) Determination of isomorphism and its applications for arbitrary graphs based on circuit simulation. *Circuits, Systems & Signal Processing*, 27(5), pp. 749–761, doi:10.1007/s00034-008-9054-7.

Liu, J. (1997) Hidden symmetry in molecular graphs. *Journal of the Chemical Society, Faraday Transactions*, 93(1), pp. 5–9, doi:10.1039/A602071B.

López-Presa, J.L. (2009) *Efficient Algorithms for Graph Isomorphism Testing*. PhD, University Rey Juan Carlos, Madrid, URL `http://www.diatel.upm.es/jllopez/tesis/thesis.pdf`.

López-Presa, J.L. and Anta, A.F. (2009) Fast algorithm for graph isomorphism testing. In *Experimental Algorithms*, number 5526 in Lecture Notes in Computer Science, pp. 221–232, Springer, Berlin, Heidelberg, doi:10.1007/978-3-642-02011-7_21.

López-Presa, J.L., Chiroque, L.F., and Fernández Anta, A. (2014) Novel techniques to speed up the computation of the automorphism group of a graph. *Journal of Applied Mathematics*, 2014, p. 934637, doi:10.1155/2014/934637.

Lorrain, F. and White, H.C. (1977) Structural equivalence of individuals in social networks. In Leinhardt, S. (editor) *Social Networks*, pp. 67–98, Academic Press, New York, doi:10.1016/B978-0-12-442450-0.50012-2.

Lu, G. (2017) Generalized degree-based graph entropies. *Symmetry*, 9(3), p. 29, doi:10.3390/sym9030029.

Lubiw, A. (1981) Some NP-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1), pp. 11–21, doi:10.1137/0210002.

Lucchini, A., Menegazzo, F., and Morigi, M. (2004) Generating permutation groups. *Communications in Algebra*, 32(5), pp. 1729–1746, doi:10.1081/AGB-120029899.

Lueker, G.S. and Booth, K.S. (1979) A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM*, 26(2), pp. 183–195, doi:10.1145/322123.322125.

Luks, E.M. (1982) Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1), pp. 42–65, doi:10.1016/0022-0000(82)90009-5.

von Luxburg, U. (2007) A tutorial on spectral clustering. *Statistics and Computing*, 17(4), pp. 395–416, doi:10.1007/s11222-007-9033-z.

von Luxburg, U. (2010) Clustering stability: An overview. *Foundations and Trends in Machine Learning*, 2(3), pp. 235–274, doi:10.1561/2200000008.

MacArthur, B.D., Sánchez-García, R.J., and Anderson, J.W. (2008) Symmetry in complex networks. *Discrete Applied Mathematics*, 156(18), pp. 3525–3531, doi:10.1016/j.dam.2008.04.008.

Magner, A., Janson, S., Kollias, G., and Szpankowski, W. (2014) On symmetry of uniform and preferential attachment graphs. *The Electronic Journal of Combinatorics*, 21(3), pp. 3–32, URL `http://www.combinatorics.org/ojs/index.php/eljc/article/view/v21i3p32`.

Mann, H.B. and Whitney, D.R. (1947) On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1), pp. 50–60, doi:10.1214/aoms/1177730491.

Mansour, M. (2017) A message-passing algorithm for graph isomorphism. *arXiv:1704.00395 [cs]*, `1704.00395`, URL `http://arxiv.org/abs/1704.00395`.

Martin, S., Brown, W.M., Klavans, R., and Boyack, K.W. (2011) OpenOrd: An open-source toolbox for large graph layout. In *Visualization and Data Analysis 2011*, *Proceedings of SPIE-IS&T Electronic Imaging*, volume 7868, p. 786806, SPIE, Bellingham, WA, doi:10.1117/12.871402.

Mathon, R. (1979) A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3), pp. 131–136, doi:10.1016/0020-0190(79)90004-8.

McKay, B.D. (1981) Practical graph isomorphism. *Congressus Numerantium*, 30, pp. 45–87, URL `http://users.cecs.anu.edu.au/~bdm/papers/pgi.pdf`.

McKay, B.D. and Piperno, A. (2014) Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60, pp. 94–112, doi:10.1016/j.jsc.2013.09.003.

de Melo, V.A., Boaventura-Netto, P.O., and Bahiense, L. (2013) QAPV: A polynomial invariant for graph isomorphism testing. *Pesquisa Operacional*, 33(2), pp. 163–184, doi:10.1590/S0101-74382013000200002.

Milgram, S. (1967) The small-world problem. *Psychology Today*, 1(1), pp. 61–67.

Miller, G. (1980) Isomorphism testing for graphs of bounded genus. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC ’80, pp. 225–235, ACM, New York, NY, USA, doi:10.1145/800141.804670.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002) Network motifs: Simple building blocks of complex networks. *Science*, 298(5594), pp. 824–827, doi:10.1126/science.298.5594.824.

Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., and Bhattacharjee, B. (2007) Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC ’07, pp. 29–42, ACM, New York, NY, USA, doi:10.1145/1298306.1298311.

Miyazaki, T. (1997) The complexity of McKay's canonical labeling algorithm. In *Groups and Computation II*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 28, pp. 239–256, American Mathematical Society, Providence, RI, URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.3532&rep=rep1&type=pdf`.

Mohar, B. (2006) A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1), pp. 6–26, doi:10.1137/S089548019529248X.

Mortici, C. (2009) An ultimate extremely accurate formula for approximation of the factorial function. *Archiv der Mathematik*, 93(1), pp. 37–45, doi:10.1007/s00013-009-0008-5.

Murtagh, F. (2009) Symmetry in data mining and analysis: A unifying view based on hierarchy. *Proceedings of the Steklov Institute of Mathematics*, 265(1), pp. 177–198, doi:10.1134/S0081543809020175.

Muzychuk, M. (2004) A solution of the isomorphism problem for circulant graphs. *Proceedings of the London Mathematical Society*, 88(1), pp. 1–41, doi:10.1112/S0024611503014412.

Myrvold, W. and Kocay, W. (2011) Errors in graph embedding algorithms. *Journal of Computer and System Sciences*, 77(2), pp. 430–438, doi:10.1016/j.jcss.2010.06.002.

von Neumann, J. (1996) *Mathematische Grundlagen der Quantenmechanik*. Springer, Berlin, second edition, doi:10.1007/978-3-642-61409-5, reprint of the first edition of 1932.

Newman, M.E.J. (2003) The structure and function of complex networks. *SIAM Review*, 45(2), pp. 167–256, doi:10.1137/S003614450342480.

Newman, M.E.J. (2004) Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, pp. 066133:1–5, doi:10.1103/PhysRevE.69.066133.

Newman, M.E.J. (2010) *Networks: An Introduction*. Oxford University Press, Oxford, doi:10.1093/acprof:oso/9780199206650.001.0001.

Newman, M.E.J. (2017) Power-law distribution. *Significance*, 14(4), pp. 10–11, doi:10.1111/j.1740-9713.2017.01050.x.

Newman, M.E.J. and Girvan, M. (2004) Finding and evaluating community structure in networks. *Physical Review E*, 69(2), pp. 026113:1–15, doi:10.1103/PhysRevE.69.026113.

Newman, M.E.J. and Watts, D.J. (1999) Scaling and percolation in the small-world network model. *Physical Review E*, 60(6), pp. 7332–7342, doi:10.1103/PhysRevE.60.7332.

Nickel, S., Stein, O., and Waldmann, K.H. (2011) *Operations Research*. Springer, Berlin, Heidelberg.

OEIS Foundation Inc. (2017) The on-line encyclopedia of integer sequences. Accessed September 5, 2017, URL `http://oeis.org/`.

Ovelgönne, M. (2011) *Scalable Algorithms for Community Detection in Very Large Graphs*. PhD, Karlsruhe Institute of Technology, Karlsruhe, URL `https://publikationen.bibliothek.kit.edu/1000024924/1911550`.

Ovelgönne, M. and Geyer-Schulz, A. (2010) Cluster cores and modularity maximization. In Fan, W., Hsu, W., Webb, G.I., Liu, B., Zhang, C., Gunopulos, D., and Wu, X. (editors) *Proceeding of the 10th IEEE International Conference on Data Mining Workshops*, pp. 1204–1213, IEEE Computer Society, Los Alamitos, doi:10.1109/ICDMW.2010.63.

Ovelgönne, M. and Geyer-Schulz, A. (2013) An ensemble learning strategy for graph clustering. In Bader, D.A., Meyerhenke, H., Sanders, P., and Wagner, D. (editors) *Graph Partitioning and Graph Clustering*, number 588 in Contemporary Mathematics, pp. 187–205, American Mathematical Society, Providence, doi:10.1090/conm/588/11701.

Ovelgönne, M., Geyer-Schulz, A., and Stein, M. (2010) Randomized greedy modularity optimization for group detection in huge social networks. In *SNA-KDD'10: Proceedings of the 4th Workshop on Social Network Mining and Analysis*, pp. 1–9, ACM, New York, NY, USA, URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.614&rep=rep1&type=pdf`.

Park, J.H. (2001) Analysis and synthesis in architectural designs: A study in symmetry. *Nexus Network Journal*, 3(1), pp. 85–98, doi:10.1007/s00004-000-0007-0.

Park, M., Lee, S., Chen, P.C., Kashyap, S., Butt, A.A., and Liu, Y. (2008) Performance evaluation of state-of-the-art discrete symmetry detection algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE Computer Society, Anchorage, doi:10.1109/CVPR.2008.4587824.

Perrett, D.I., Burt, D.M., Penton-Voak, I.S., Lee, K.J., Rowland, D.A., and Edwards, R. (1999) Symmetry and human facial attractiveness. *Evolution and Human Behavior*, 20(5), pp. 295–307, doi:10.1016/S1090-5138(99)00014-8.

Petersen, K. (1983) Entropy. In *Ergodic Theory*, *Cambridge Studies in Advanced Mathematics*, volume 2, pp. 227–248, Cambridge University Press, Cambridge, doi:10.1017/CBO9780511608728.007.

Piperno, A. (2008) Search space contraction in canonical labeling of graphs. *arXiv:0804.4881 [cs]*, `0804.4881`, URL `http://arxiv.org/abs/0804.4881`.

Plummer, D. (1982) Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. RFC 826, Internet Engineering Taskforce, URL `https://tools.ietf.org/html/rfc826`.

Pólya, G. (1937) Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta mathematica*, 68(1), pp. 145–254, doi:10.1007/BF02546665.

Price, D.J.d.S. (1965) Networks of scientific papers. *Science*, 149(3683), pp. 510–515, doi:10.2307/1716232.

Price, D.J.d.S. (1976) A general theory of bibliometric and other cumulative advantage processes. *Journal of the American Society for Information Science*, 27(5), pp. 292–306, doi:10.1002/asi.4630270505.

Quintas, L.V. (1967) Extrema concerning asymmetric graphs. *Journal of Combinatorial Theory*, 3(1), pp. 57–82, doi:10.1016/S0021-9800(67)80018-8.

Raghavan, U.N., Albert, R., and Kumara, S. (2007) Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3), pp. 036106:1–11, doi:10.1103/PhysRevE.76.036106.

Rashevsky, N. (1955) Life, information theory, and topology. *The bulletin of mathematical biophysics*, 17(3), pp. 229–235, doi:10.1007/BF02477860.

Raussen, M. and Skau, C. (2015) Interview with Yakov Sinai. *Notices of the American Mathematical Society.*, 62(2), pp. 152–160, doi:10.1090/noti1210.

Ravasz, E. and Barabási, A.L. (2003) Hierarchical organization in complex networks. *Physical Review E*, 67(2), pp. 026112:1–7, doi:10.1103/PhysRevE.67.026112.

Raychaudhury, C., Ray, S.K., Ghosh, J.J., Roy, A.B., and Basak, S.C. (1984) Discrimination of isomeric structures using information theoretic topological indices. *Journal of Computational Chemistry*, 5(6), pp. 581–588, doi:10.1002/jcc.540050612.

Read, R.C. and Corneil, D.G. (1977) The graph isomorphism disease. *Journal of Graph Theory*, 1(4), pp. 339–363, doi:10.1002/jgt.3190010410.

Reber, R. (2002) Reasons for the preference for symmetry. *Behavioral and Brain Sciences*, 25(3), pp. 415–416, doi:10.1017/S0140525X02350076.

Reichardt, J. and Bornholdt, S. (2006) When are networks truly modular? *Physica D: Nonlinear Phenomena*, 224(1–2), pp. 20–26, doi:10.1016/j.physd.2006.09.009.

Remak, R. (1911) Über die Zerlegung der endlichen Gruppen in direkte unzerlegbare Faktoren. *Journal für die reine und angewandte Mathematik*, 139, pp. 293–308, URL `https://eudml.org/doc/149350`.

Rensink, A. (2007) Isomorphism checking in GROOVE. *Electronic Communications of the EASST*, 1, doi:10.14279/tuj.eceasst.1.77.

de Ridder, H.N. et al. (2010) Information system on graph classes and their inclusions (ISGCI). Accessed January 30, 2017, URL `http://www.graphclasses.org`.

Rossi, R.A. and Ahmed, N.K. (2015) The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 4292–4293, AAAI Press, Austin, Texas, URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9553`.

Rudinger, K., Gamble, J.K., Bach, E., Friesen, M., Joynt, R., and Coppersmith, S.N. (2013) Comparing algorithms for graph isomorphism using discrete- and continuous-time quantum random walks. *Journal of Computational and Theoretical Nanoscience*, 10(7), pp. 1653–1661, doi:10.1166/jctn.2013.3105.

Ruzzenenti, F., Garlaschelli, D., and Basosi, R. (2010) Complex networks and symmetry II: Reciprocity and evolution of world trade. *Symmetry*, 2(3), pp. 1710–1744, doi:10.3390/sym2031710.

Saari, D.G. (1988) Symmetry, voting, and social choice. *The Mathematical Intelligencer*, 10(3), pp. 32–42, doi:10.1007/BF03026639.

Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., and Vergés, J. (2002) Graph-based representations and techniques for image processing and image analysis. *Pattern Recognition*, 35(3), pp. 639–650, doi:10.1016/S0031-3203(01)00066-8.

Sardiu, M.E. and Washburn, M.P. (2011) Building protein-protein interaction networks with proteomics and informatics tools. *The Journal of Biological Chemistry*, 286(27), pp. 23645–23651, doi:10.1074/jbc.R110.174052.

Schaeffer, S.E. (2007) Graph clustering. *Computer Science Review*, 1(1), pp. 27–64, doi:10.1016/j.cosrev.2007.05.001.

Schaub, M.T., Delvenne, J.C., Rosvall, M., and Lambiotte, R. (2017) The many facets of community detection in complex networks. *Applied Network Science*, 2(1), pp. 4:1–13, doi:10.1007/s41109-017-0023-6.

Scheib, J.E., Gangestad, S.W., and Thornhill, R. (1999) Facial attractiveness, symmetry and cues of good genes. *Proceedings of the Royal Society of London B: Biological Sciences*, 266(1431), pp. 1913–1917, doi:10.1098/rspb.1999.0866.

Schieber, T.A., Carpi, L., Díaz-Guilera, A., Pardalos, P.M., Masoller, C., and Ravetti, M.G. (2017) Quantification of network structural dissimilarities. *Nature Communications*, 8, pp. 13928:1–10, doi:10.1038/ncomms13928.

Schmidt, D.C. and Druffel, L.E. (1976) A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of the ACM*, 23(3), pp. 433–445, doi:10.1145/321958.321963.

Seress, A. (2003) *Permutation Group Algorithms*. Number 152 in Cambridge Tracts in Mathematics, Cambridge University Press, Cambridge, doi:10.1017/CBO9780511546549.

Shannon, C.E. (1948) A mathematical theory of communication. *The Bell System Technical Journal*, 27, pp. 379–423, doi:10.1145/584091.584093.

Sharma, H., Alekseychuk, A., Leskovsky, P., Hellwich, O., Anand, R., Zerbe, N., and Hufnagl, P. (2012) Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics. *Diagnostic Pathology*, 7, pp. 134:1–20, doi:10.1186/1746-1596-7-134.

Shi, P., Zheng, X., Ratkowsky, D.A., Li, Y., Wang, P., and Cheng, L. (2018) A simple method for measuring the bilateral symmetry of leaves. *Symmetry*, 10(4), pp. 118:1–10, doi:10.3390/sym10040118.

Sims, C.C. (1970) Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra*, pp. 169–183, Pergamon Press, Oxford, doi:10.1016/B978-0-08-012975-4.50020-5.

Sinai, Y.G. (1959) On the notion of entropy of a dynamical system. *Doklady Akademiia Nauk SSSR*, 125(5), pp. 768–771.

Sinai, Y.G. (2010) On the notion of entropy of a dynamical system. In *Ergodic Theory and Dynamical Systems*, *Selecta*, volume 1, pp. 3–14, Springer, Dordrecht, doi:10.1007/978-0-387-87870-6_1.

Sloane, N.J.A. (1973) *A Handbook of Integer Sequences*. Academic Press, New York, doi:10.1016/C2013-0-11509-7.

Solomonoff, R. and Rapoport, A. (1951) Connectivity of random nets. *The bulletin of mathematical biophysics*, 13(2), pp. 107–117, doi:10.1007/BF02478357.

Song, C., Havlin, S., and Makse, H.A. (2005) Self-similarity of complex networks. *Nature*, 433, pp. 392–395, doi:10.1038/nature03248.

Sorlin, S. and Solnon, C. (2004) A global constraint for graph isomorphism problems. In Régin, J.C. and Rueher, M. (editors) *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, number 3011 in Lecture Notes in Computer Science, pp. 287–301, Springer, Berlin, Heidelberg, doi:10.1007/978-3-540-24664-0_20.

Staudt, C.L. and Meyerhenke, H. (2016) Engineering parallel algorithms for community detection in massive networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(1), pp. 171–184, doi:10.1109/TPDS.2015.2390633.

Stein, O. (2012) Twice differentiable characterizations of convexity notions for functions on full dimensional convex sets. *Schedae Informaticae*, 21, pp. 55–63, doi:10.4467/20838476SI.12.004.0814.

Stelzl, U., Worm, U., Lalowski, M., Haenig, C., Brembeck, F.H., Goehler, H., Stroedicke, M., Zenkner, M., Schoenherr, A., Koeppen, S., Timm, J., Mintzlaff, S., Abraham, C., Bock, N., Kietzmann, S., Goedde, A., Toksöz, E., Droege, A., Krobitsch, S., Korn, B., Birchmeier, W., Lehrach, H., and Wanker, E.E. (2005) A human protein-protein interaction network: A resource for annotating the proteome. *Cell*, 122(6), pp. 957–968, doi:10.1016/j.cell.2005.08.029.

Stoichev, S.D. (2010) Vsep-new heuristic and exact algorithms for graph automorphism group computation. *arXiv:1007.1726 [cs, math]*, `1007.1726`, URL `http://arxiv.org/abs/1007.1726`.

Tener, G.D. (2009) *Attacks on Difficult Instances of Graph Isomorphism: Sequential and Parallel Algorithms*. PhD, University of Central Florida, Orlando, URL `http://etd.fcla.edu/CF/CFE0002894/Tener_Greg_D_200912_PhD.pdf`.

Tibshirani, R. and Walther, G. (2005) Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14(3), pp. 511–528, doi:10.1198/106186005X59243.

Tinhofer, G. (1991) A note on compact graphs. *Discrete Applied Mathematics*, 30(2), pp. 253–264, doi:10.1016/0166-218X(91)90049-3.

Todeschini, R. and Consonni, V. (2008) *Handbook of Molecular Descriptors*. Wiley-VCH, Weinheim, doi:10.1002/9783527613106.

Traag, V.A., Van Dooren, P., and Nesterov, Y. (2011) Narrow scope for resolution-limit-free community detection. *Physical Review E*, 84(1), pp. 016114:1–9, doi:10.1103/PhysRevE.84.016114.

Trucco, E. (1956a) A note on the information content of graphs. *The bulletin of mathematical biophysics*, 18(2), pp. 129–135, doi:10.1007/BF02477836.

Trucco, E. (1956b) On the information content of graphs: Compound symbols; different states for each point. *The bulletin of mathematical biophysics*, 18(3), pp. 237–253, doi:10.1007/BF02481859.

Ugander, J. and Backstrom, L. (2013) Balanced label propagation for partitioning massive graphs. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pp. 507–516, ACM, doi:10.1145/2433396.2433461.

Ullmann, J.R. (1976) An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1), pp. 31–42, doi:10.1145/321921.321925.

Viana, M.A.G. (2005) Symmetry studies. Lecture notes, Eurandom, Eindhoven, URL `https://research.tue.nl/en/publications/symmetry-studies-lecture-notes`.

Viana, M.A.G. (2006) Symmetry studies and decompositions of entropy. *Entropy*, 8(2), pp. 88–109, doi:10.3390/e8020088.

Viana, M.A.G. (2007) Symmetry studies for data analysis. *Methodology and Computing in Applied Probability*, 9(2), pp. 325–341, doi:10.1007/s11009-007-9022-x.

Wang, H., Yan, G., and Xiao, Y. (2009) Symmetry in world trade network. *Journal of Systems Science and Complexity*, 22(2), pp. 280–290, doi:10.1007/s11424-009-9163-9.

Wang, J., Huang, Y., Wu, F.X., and Pan, Y. (2012) Symmetry compression method for discovering network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(6), pp. 1776–1789, doi:10.1109/TCBB.2012.119.

Watts, D.J. and Strogatz, S.H. (1998) Collective dynamics of 'small-world' dynamics. *Nature*, 393, pp. 440–442, doi:10.1038/30918.

Weisfeiler, B. (1976) *On Construction and Identification of Graphs*. Number 558 in Lecture Notes in Mathematics, Springer, Berlin, doi:10.1007/BFb0089374.

Welch, E. and Kobourov, S. (2017) Measuring symmetry in drawings of graphs. *Computer Graphics Forum*, 36(3), pp. 341–351, doi:10.1111/cgf.13192.

Wertheimer, M. (1922) Untersuchungen zur Lehre von der Gestalt. *Psychologische Forschung*, 1(1), pp. 47–58, doi:10.1007/BF00410385.

Wertheimer, M. (1923) Untersuchungen zur Lehre von der Gestalt. II. *Psychologische Forschung*, 4(1), pp. 301–350, doi:10.1007/BF00410640.

Weyl, H. (1952) *Symmetry*. Princeton University Press, Princeton.

Wielandt, H. (1964) *Finite Permutation Groups*. Academic Press, New York, doi:10.1016/C2013-0-11702-3.

Wilcoxon, F. (1945) Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6), pp. 80–83, doi:10.2307/3001968.

Wonka, P., Wimmer, M., Sillion, F., and Ribarsky, W. (2003) Instant architecture. *ACM Transactions on Graphics*, 22(3), pp. 669–677, doi:10.1145/1201775.882324.

Wu, P. and Antsaklis, P.J. (2010) Symmetry in the design of large-scale complex control systems: Some initial results using dissipativity and Lyapunov stability. In *Proceedings of the 18th Mediterranean Conference on Control & Automation*, pp. 197–202, IEEE Computer Society, doi:10.1109/MED.2010.5547667.

Wu, W., Xiao, Y., Wang, W., He, Z., and Wang, Z. (2010) K-symmetry model for identity anonymization in social networks. In *Proceedings of the 13th International Conference on Extending Database Technology*, pp. 111–122, ACM, New York, NY, USA, doi:10.1145/1739041.1739058.

Wynn, T. (2002) Archaeology and cognitive evolution. *Behavioral and Brain Sciences*, 25(3), pp. 389–402, doi:10.1017/S0140525X02000079.

Xiao, Y., MacArthur, B.D., Wang, H., Xiong, M., and Wang, W. (2008a) Network quotients: Structural skeletons of complex systems. *Physical Review E*, 78(4), pp. 046102:1–7, doi:10.1103/PhysRevE.78.046102.

Xiao, Y., Xiong, M., Wang, W., and Wang, H. (2008b) Emergence of symmetry in complex networks. *Physical Review E*, 77(6), pp. 066108:1–10, doi:10.1103/PhysRevE.77.066108.

Xu, K. and Li, W. (2006) Many hard examples in exact phase transitions. *Theoretical Computer Science*, 355(3), pp. 291–302, doi:10.1016/j.tcs.2006.01.001.

Zachary, W.W. (1977) An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4), pp. 452–473, doi:10.1086/jar.33.4.3629752.

Zenil, H., Soler-Toscano, F., Dingle, K., and Louis, A.A. (2014) Correlation of automorphism group size and topological properties with program-size complexity evaluations of graphs and complex networks. *Physica A: Statistical Mechanics and its Applications*, 404, pp. 341–358, doi:10.1016/j.physa.2014.02.060.

Zhang, B.D., Wu, J.J., Tang, Y.H., and Zhou, J. (2012) Scale-free tree graphs are more symmetrical. *Physica Scripta*, 86(2), pp. 025006:1–7, doi:10.1088/0031-8949/86/02/025006.

Zhao, C. and Parhami, B. (2018) Symmetric agency graphs facilitate and improve the quality of virtual network embedding. *Symmetry*, 10(3), pp. 63:1–17, doi:10.3390/sym10030063.

Zhao, Y. (2017) A survey on theoretical advances of community detection in networks. *Wiley Interdisciplinary Reviews: Computational Statistics*, 9(5), pp. e1403:1–13, doi:10.1002/wics.1403.