On Algorithmic Certification of Graph Structures

Oliver Bachtler

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Vom Fachbereich Mathematik
der Rheinland-Pfälzischen Technischen Universität
Kaiserslautern-Landau
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte Dissertation

Gutachter:	Prof. Dr. Sven Oliver Krumke
	Prof. Dr. Ir. Arie M.C.A. Koster
Datum der Disputation:	30.03.2023

DE-386

Abstract

Many open problems in graph theory aim to verify that a specific class of graphs has a certain property. One example, which we study extensively in this thesis, is the *3-decomposition conjecture*. It states that every cubic graph can be decomposed into a spanning tree, cycles, and a matching. Our most noteworthy contributions to this conjecture are a proof that graphs which are *star-like* satisfy the conjecture and that several small graphs, which we call *forbidden subgraphs*, cannot be part of minimal counterexamples. These star-like graphs are a natural generalisation of Hamiltonian graphs in this context and encompass an infinite family of graphs for which the conjecture was not known previously. Moreover, we use the forbidden subgraphs we determined to deduce that 3-connected cubic graphs of path-width at most 4 satisfy the 3-decomposition conjecture: we do this by showing that the path-width restriction causes one of these forbidden subgraphs to appear.

In the second part of this thesis, we delve deeper into two steps of the proof that 3-connected cubic graphs of path-width 4 satisfy the conjecture. These steps involve a significant amount of case distinctions and, as such, are impractical to extend to larger path-width values. We show how to formalise the techniques used in such a way that they can be implemented and solved algorithmically. As a result, only the work that is ‘interesting’ to do remains and the many ‘straightforward’ parts can now be done by a computer. While one step is specific to the 3-decomposition conjecture, we derive a general algorithm for the other. This algorithm takes a class of graphs $\mathcal{G}$ as an input, together with a set of graphs $\mathcal{U}$, and a path-width bound k. It then attempts to answer the following question: does any graph in $\mathcal{G}$ that has path-width at most k contain a subgraph in $\mathcal{U}$? We show that this problem is undecidable in general, so our algorithm does not always terminate, but we also provide a general criterion that guarantees termination.

In the final part of this thesis we investigate two connectivity problems on directed graphs. We prove that verifying the existence of an *st*-path in a local certification setting, cannot be achieved with a constant number of bits. More precisely, we show that a *proof labelling scheme* needs $\Theta(\log \Delta)$ many bits, where Δ denotes the maximum degree. Furthermore, we investigate the complexity of the *separating by forbidden pairs* problem, which asks for the smallest number of arc pairs that are needed such that any *st*-path completely contains at least one such pair. We show that the corresponding decision problem in $\Sigma_2\mathsf{P}$-complete.

Zusammenfassung

Viele offene Probleme in der Graphentheorie beschäftigen sich mit dem Nachweis einer speziellen Eigenschaft für eine bestimmte Klasse von Graphen. Ein Beispiel, welches wir in dieser Arbeit detailliert untersuchen, ist die 3-Zerlegungsvermutung. Diese besagt, dass jeder kubische Graph in einen Spannbaum, Kreise und ein Matching zerlegt werden kann. Unsere wichtigsten Beiträge zu dieser Vermutung sind ein Beweis, dass *sternförmige* Graphen die Vermutung erfüllen und dass mehrere kleine Graphen, die wir *verbotene Subgraphen* nennen, nicht Teil minimaler Gegenbeispiele sind. Dabei sind sternförmige Graphen eine natürliche Verallgemeinerung von Graphen die einem Hamiltonkeis besitzen und es gibt eine unendliche Familie von sternförmigen Graphen, für die die Vermutung bisher nicht geklärt war. Darüber hinaus verwenden wir die verbotenen Subgraphen, um zu beweisen, dass 3-zusammenhängende kubische Graphen mit Pfadweite höchstens 4 die 3-Zerlegungsvermutung erfüllen: Dazu zeigen wir, dass die Beschränkung der Pfadweite dazu führt, dass einer der verbotenen Subgraphen auftauchen muss.

Im zweiten Teil dieser Arbeit schauen wir uns zwei Schritte des Beweises, dass 3-zusammenhängende kubische Graphen mit Pfadweite höchstens 4 die Vermutung erfüllen, genauer an. Diese Schritte sind für den Großteil der Fallunterscheidungen verantwortlich und führen dazu, dass der Beweis sich schlecht auf größere Werte für die Pfadweite fortsetzt. Deshalb formalisieren wir die verwendeten Techniken so, dass man sie implementieren und diese Schritte algorithmisch lösen kann. Dies führt dazu, dass man sich bei den Beweisen auf die „interessanten“ Teile fokussieren kann und die vielen „geradlinigen“ Schritte überprüft ein Computer. Ersterer der beiden Schritte ist spezifisch für die 3-Zerlegungsvermutung. Den zweiten behandeln wir jedoch allgemeiner und entwickeln dafür einen Algorithmus, der als Eingabe eine Klasse von Graphen $\mathcal{G}$, eine Menge von Graphen $\mathcal{U}$ und eine Pfadweitebeschränkung k nimmt. Dieser Algorithmus versucht nun, folgende Frage zu beantworten: Enthält jeder Graph in $\mathcal{G}$ der Pfadweite höchstens k einen Subgraph in $\mathcal{U}$? Wir zeigen, dass dieses Problem im Allgemeinen unentscheidbar ist, womit der Algorithmus im Allgemeinen nicht terminieren muss, aber wir beweisen auch ein Terminierungskriterium.

Im letzten Teil behandeln wir zwei Zusammenhangsprobleme auf gerichteten Graphen. Wir zeigen, dass eine konstante Anzahl Bits nicht ausreicht, um lokal zu überprüfen, ob ein st-Pfad in einem Graph existiert. Genauer gesagt, beweisen wir, dass ein *Proof Labelling Scheme* $\Theta(\log \Delta)$ Bits für diese Aufgabe benötigt, wobei Δ der Maximalgrad ist. Weiterhin analysieren wir die Komplexität des *Separating by Forbidden Pairs* Problems, das die geringste Anzahl an Kantenpaaren sucht, sodass jeder st-Pfad mindestens ein Paar komplett enthält. Wir zeigen, dass dieses Problem $\Sigma_2\mathsf{P}$-vollständig ist.

Acknowledgements

I cannot begin to express my gratitude to my supervisor Sven O. Krumke for everything he has done for me. Aside from giving me free rein to pursue my research interests, he was (and still is) always there to encourage and support me. I am also extremely thankful to Irene Heinrich and Pascal Schweitzer for going out of their way to assist me with helpful suggestions, advice, and insights.

Special thanks go to my co-authors Tim Bergner, Irene Heinrich, and Sven O. Krumke, who all provided invaluable contributions to this work. It is a joy to work with all of you. I also wish to thank my external examiner Prof. Dr. Ir. Arie M.C.A. Koster for agreeing to read and evaluate this thesis.

I very much appreciate the time and effort my proofreaders put into reading this thesis. Thank you, Tim Bergner, Sebastian Blauth, Jan Böckmann, Tobias Dietz, Christoph Geis, Nils Hausbrandt, Irene Heinrich, Stephan Helfrich, Helena Petri, Kathrin Prinz, Adrian Rettich, and Eva Schmidt for all your useful feedback.

Many thanks also go to all my colleagues, current and former, in the optimisation group for all the fruitful discussions and nice work atmosphere that make coming to work every day both beneficial and enjoyable.

Finally, I am grateful to my family for their unwavering support.

Contents

Chapter 1.

Introduction

Many results and conjectures in graph theory are of the following (very generic) form, where $\mathcal{G}$ is a class of graphs and π is some property.

Generic Conjecture. Every graph in $\mathcal{G}$ satisfies π. $\lhd$

Examples are Dirac's theorem [Dir52], which states that every graph G whose minimum degree is at least $\frac{|VG|}{2}$ is Hamiltonian, and the four colour theorem [AH77, AHK77], which states that every loopless planar graph has chromatic number at most 4. The conjecture we extensively study is called the 3-decomposition conjecture [Hof11] which is of the form above.

3-Decomposition Conjecture. Every finite connected cubic graph has a decomposition consisting of a spanning tree, a 2-regular subgraph, and a matching. $\lhd$

Such decompositions are called 3-decompositions and Figure 1.1 depicts four examples.

To keep the 3-decomposition conjecture company, we add two more well-known conjectures to the mix: the cycle double cover conjecture [Sze73, Sey79] states that every bridgeless graph has a family of cycles such that every edge is covered exactly twice and the Berge-Fulkerson conjecture [Ful71] claims that every bridgeless cubic graph has a family of six perfect matchings such that every edge is covered exactly twice.

A common approach to studying such conjectures is to derive properties satisfied by every minimum counterexample, in case counterexamples exist. Here, a minimum counterexample is just a counterexample that is minimal in some sense, usually with respect to the number of vertices. For all the stated conjectures, such results exist. For

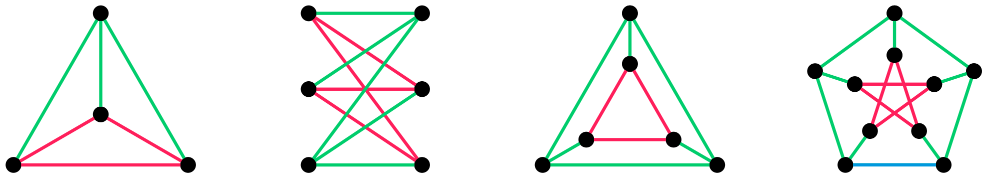


Figure 1.1.: Examples of 3-decompositions for four small cubic graphs. The green edges are part of the spanning tree, the red ones make up the 2-regular subgraph, and the blue edges form a matching.

example, no minimum counterexample to the 3-decomposition conjecture contains a triangle [Bac15, Hei20]. The cycle double cover conjecture can be reduced to snarks of girth at least 12 [Jae85, Huc00]. (Snarks are cubic, cyclically 4-edge-connected graphs that are not 3-edge-colourable.) Similarly, the Berge-Fulkerson conjecture needs a cyclically 5-edge-connected snark as a minimum counterexample [MM20].

Such properties are generally obtained as follows: let G be a minimum counterexample. Then, using some operation, a smaller graph G' is generated from G, which is not a counterexample and, therefore, satisfies π. This graph is then used to show that G also satisfies π, contradicting that it is a counterexample. The last step heavily depends on how G' was generated from G and this has to be done in a way that is compatible with the property π. Usually, one can use this method to determine structures that cannot be part of a minimum counterexample, like the triangle for the 3-decomposition conjecture or certain edge-cuts for the cycle double cover and Berge-Fulkerson conjectures.

Even if one cannot prove the conjecture in its entirety this way, it is oftentimes possible to rule out certain graphs, for example, those of small path- or tree-width. This works by showing that the additional width restriction is sufficient to make one of those structures appear that may not be part of a minimum counterexample. We call such structures *forbidden* and conclude that no minimum counterexample to the conjecture can have small width. This does not yet show that all graphs of small width satisfy the conjecture since the smaller graph G' constructed from G could have larger width. However, if we can additionally construct the smaller graph G' from G is such a way that its width does not increase, then we actually get a proof for all graphs of small width. To see this, simply repeat the above arguments when starting with a minimum counterexample of small width.

This serves as motivation for the first two parts of this thesis. In Part I we study the 3-decomposition conjecture. In particular, we look at it in the way described above and show that it holds for all graphs of path-width at most 4. In Part II we revisit the two proof techniques we used in this process that are responsible for a majority of case distinctions that are needed. We show how these can be automated algorithmically and discuss how they can be made practically useful, despite the corresponding problems being hard in general. In addition, the techniques we develop to prove the algorithms' correctness are also independently helpful and we use them to obtain bounds on the girth of cubic graphs.

A paradigm shift occurs in the transition to Part III, where we consider two connectivity problems on directed graphs with a given source s and sink t. Since graphs represent how objects are connected, it is unsurprising that connectivity problems are well-studied. Numerous generalisations of the 'standard' notion of connectivity exist, like vertex- and edge-connectivity. These have been studied in detail and are closely related to flows and cuts. One of the main results in this area is Menger's theorem [Men27, cf. Die16] or the closely related max-flow min-cut theorem [DF55, cf. AMO93], which both describe duality. Menger's theorem shows that edge-separators are dual to disjoint paths and the max-flow min-cut theorem establishes the duality of st-flows and -cuts. Connectivity problems have also been investigated algorithmically, and fast algorithms have been developed, see [AMO93]. Some typical examples are algorithms computing connected components, shortest paths, or maximum flows.

The first connectivity problem we look at in this thesis is the standard *st*-reachability problem, where we wish to determine the existence of an *st*-path. We look at this problem in the setting of *local certification* (which we intuitively describe below and formally define later). The second problem we study is the separating by forbidden pairs problem. In it, we want to find the fewest pairs of arcs such that every *st*-path contains at least one of these pairs completely. This problem appears as the dual of the almost disjoint paths problem in which the goal is to find as many *st*-paths as possible that share at most one arc. This is a very natural generalisation of the disjoint paths problem where vertex- or edge-cuts are the dual concept. For the almost disjoint paths and the separating by forbidden pairs problem, we get weak duality: any path in a set of almost disjoint paths must contain a pair and none can share one. Unlike in the disjoint case however, where Menger's theorem yields strong duality, the optimal solution values here need not coincide.

We now go into a bit more detail by describing what happens in the upcoming chapters and what our contributions are.

Part I. The 3-Decomposition Conjecture

The first part of this thesis consists of three chapters. In Chapter 3 we discuss known results for the 3-decomposition conjecture and present several preliminary results of our own. We investigate a relaxation in which the matching is extended to allow for paths of length at most 2 and, in this context, study non-separating cycles in graphs. Next, in Chapter 4, we look at a special class of graphs, which are 'star-like' in some sense and naturally generalise Hamiltonian graphs in this setting, and show that the conjecture holds for the graphs in this class. We also construct an infinite family of star-like graphs that are in none of the classes for which the conjecture is already known to hold. Lastly, in Chapter 5, we follow the steps from our motivation to determine six graphs that cannot be part of a minimum counterexample to the conjecture and use these to conclude that it holds for all graphs of path-width at most 4. The former step, when done manually, contains a large amount of easy cases, which we have relegated to Appendix A.

Contributions. In this part, we obtain several new results for the 3-decomposition conjecture by using both a local and a global approach. We already mentioned that we prove the conjecture for star-like graphs and for those of path-width at most 4. The former uses the imposed global structure and we define certain sub-decompositions of which we hope they are both reusable and extendable, such that this result can potentially be generalised. A natural candidate for such an extension are 'tree-like' graphs. For the latter, we determine and use forbidden structures, making it a local approach. From these structures we also derive some additional properties of minimum counterexamples. Moreover, we describe a new conjecture that is equivalent to the 3-decomposition conjecture and which would yield an alternative method to Wormald's theorem (see Theorem 2.12) for constructing all cubic graphs in case it is true.

Furthermore, we look at a generalisation of the 3-decomposition conjecture, where the matching condition is relaxed to allow for paths of length at most 2. It is known that this guarantees the existence of such decompositions, and we describe an efficient way of computing them. This is done by depth-first search essentially, but requires a non-separating cycle to be known beforehand. We show that such a cycle can be found in linear time on cubic graphs, despite the problem of deciding whether a non-separating cycle exists being NP-complete on subcubic graphs.

Part II. Algorithmic Proof Support

In this part, we show how to largely automate the two proof techniques from Part I that result in the majority of the case distinctions needed there. More precisely, we implement the two key steps of the proof that graphs of path-width at most 4 satisfy the conjecture. These are:

(1) determining whether a certain graph is not a subgraph of a minimum counterexample and
(2) determining whether every cubic graph of small path-width contains a subgraph in a given set.

These steps are automated in Chapters 6 and 7, respectively. The latter step is actually handled in more generality and is not restricted to cubic graphs and the 3-decomposition conjecture. The techniques we develop to prove the resulting algorithm's correctness are used in Chapter 8 where we show general bounds on the girth of cubic graphs of small path-width.

Contributions. For Step (1), the method we use in Chapter 5 is to replace the graph S in question by a smaller graph R and to check whether the possible behaviours of a 3-decomposition on R can be extended back to S. In many cases, this can be done in a 'straightforward' manner, making it easy but lengthy to do since R can admit many possible behaviours that need to be checked. We formalise what a *straightforward extension* is and show how they can be checked algorithmically. We also prove that this problem is NP-complete and computationally verify that all cubic graphs of order at most 20 satisfy the 3-decomposition conjecture.

To deal with Step (2), we formalise the procedure we used in Chapter 5 to show that the path-width bound guarantees the existence of a certain subgraph in a more general setting. Doing so, we obtain an algorithm that tries to answer the question whether every graph in a class $\mathcal{G}$ that has path-width at most k contains a subgraph in a finite set $\mathcal{U}$. In case the answer is negative, it returns a smallest counterexample. The reason we wrote that the algorithm 'tries' to answer the question is that it does not necessarily terminate. We show that this problem is undecidable, making this behaviour unavoidable. However, we obtain a termination criterion as well: if the graphs in $\mathcal{U}$ are connected, the class $\mathcal{G}$ has bounded maximum degree, and containment in $\mathcal{G}$ can be checked locally, in some sense, the algorithm can be adjusted in a way that it does terminate. To speed up this algorithm, we checked symmetries and specifically tailored it to cubic graphs since these are our main use case.

Lastly, we prove two upper bounds on the girth of a cubic graph of path-width at most k, determining the largest possible values precisely for all $k \leq 10$. Moreover, we present a new constructive characterisation of the cubic graphs of path-width 3 and girth 4.

Part III. Two Graph Connectivity Problems

The final part of this thesis is only comprised of two chapters. In Chapter 9 we study the *st*-reachability problem in the context of local certification. Roughly speaking, we want to determine whether an *st*-path exists, while only seeing what the graph looks like locally around every vertex. For this to be possible for interesting classes, local certification allows passing additional information to every vertex that can be used to check whether each local view is consistent. For example, bipartiteness becomes a property that is easy to check locally once one tells all vertices which side of the bipartition they reside on [GS16]. The quality of the local certification is measured by the amount of information that needs to be passed to the vertices of the graph, the less the better. Several concepts for local certification exist, which differ in their definition of a local view and a common one is the concept of proof labelling schemes.

In Chapter 10 we investigate a second connectivity problem: the separating by forbidden pairs problem. Here, the goal is to find few pairs of arcs such that any *st*-path contains both arcs of at least one chosen pair. This is an extension of the path avoiding forbidden pairs problem where a set of arc pairs is given and the objective is to find a path that does not contain any pair. It is also dual to the almost disjoint paths problem, which looks for a largest set of *st*-paths such that every pair of paths in this set share at most one arc.

Contributions. It is known that certifying *st*-reachability in undirected graphs can be done using a single bit of information [GS16], but the directed case has not yet been answered. We show that, for proof labelling schemes, a constant number of bits are insufficient in the directed case. In fact, $\Theta(\log \Delta)$ bits are necessary, where Δ denotes the maximum degree. Concerning the separating by forbidden pairs problem, we show that it is $\Sigma_2\mathsf{P}$-complete.

A Note on Publications

Chapters 4 and 7 to 9 of this thesis have been published in the following peer-reviewed articles:

[BBK22] O. Bachtler, T. Bergner, and S. O. Krumke. ‘Local Certification of Reachability’. *Proceedings of the 10th International Network Optimization Conference.* INOC 2022. OpenProceedings.org, 2022, pages 40–44. DOI: `10.48786/inoc.2022.08`.

[BH23] O. Bachtler and I. Heinrich. ‘Automated testing and interactive construction of unavoidable sets for graph classes of small path-width’. *Journal of Graph Theory* early view (2023). DOI: `https://doi.org/10.1002/jgt.22964`.

[BK22] O. Bachtler and S. O. Krumke. ‘Towards Obtaining a 3-Decomposition from a Perfect Matching’. *The Electronic Journal of Combinatorics* 29.4 (2022). DOI: `10.37236/11128`.

To be precise, [BK22] corresponds to Chapter 4, [BH23] to Chapters 7 and 8, and [BBK22] to Chapter 9.

Additionally, the following two preprints also contains parts of this thesis.

[BBK22] O. Bachtler, T. Bergner, and S. O. Krumke. *Almost Disjoint Paths and Separating by Forbidden Pairs.* Version 1. 2022. arXiv: `2202.10090 [math.CO]`.

[BH21] O. Bachtler and I. Heinrich. *Reductions for the 3-Decomposition Conjecture.* Version 2. 2021. arXiv: `2104.15113 [math.CO]`.

More precisely, [BH21] contains the majority of Chapters 3, 5, and 6 and has been submitted to the *Latin-American Algorithms, Graphs and Optimization Symposium.* Chapter 10 is part of [BBK22] and is also included in the dissertation of Tim Bergner. It has been submitted to *Theoretical Computer Science.*

Chapter 2.

Preliminaries

In this chapter, we introduce the notation used throughout this thesis. We shall refrain from introducing everything in detail, assuming that certain basic concepts and common notation are known. In case this assumption is incorrect, we also refer to literature (textbooks, for the most part) that defines the parts we leave out.

2.1. Graph theory

The graph-theoretic notation used here is mainly based on [Die16], to which we refer the reader unfamiliar with a concept we do not explicitly introduce. Other notable mentions, from which we have taken some notation, are [Wes01, KN12, Sch03, BM08] and to all of which we add several personal preferences.

Basics. For a set X, we write $\binom{X}{k}$ for the set of all k-element subsets of X.

We now introduce graphs by starting with the general definition that allows loops and parallel edges (defined momentarily), before simplifying it for easier use in the cases where no problems occur when doing so.

An *undirected graph*, or just *graph*, G is a triple $(V G, E G, \gamma)$ consisting of a *vertex set $V(G)$ or $V G$*, an *edge set $E(G)$ or $E G$*, and a function $\gamma\colon E G \to \binom{VG}{1} \cup \binom{VG}{2}$. The sets $V G$ and $E G$ are assumed to be disjoint and finite. Note that we omit the parentheses whenever this is possible without harming clarity. Edges e with $\gamma\, e = \{v\}$ are called *loops* and edges e, e' with $\gamma\, e = \gamma\, e'$ are *parallels*. We write $e = uv$ for an edge $e \in E G$ with $\gamma\, e = \{u, v\}$ if the edge has no parallels, or if it does not matter which parallel is used.

Similarly, a *directed graph* or *digraph* D is a quadruple $(V D, A D, \alpha, \omega)$ consisting of a vertex set $V D$, an *arc set* $A D$, and two functions α, $\omega\colon A D \to V D$. The sets $V D$ and $A D$ are disjoint and finite. Loops are arcs a with $\alpha\, a = \omega\, a$ and parallels are arcs a, a' with $\alpha\, a = \alpha\, a'$ and $\omega\, a = \omega\, a'$. Additionally, arcs a, a' with $\alpha\, a = \omega\, a'$ and $\omega\, a = \alpha\, a'$

are *anti-parallels*. Again, we write $a = uv$ for an arc $a \in A\,D$ with $\alpha\,a = u$ and $\omega\,a = v$. The *underlying graph* of D is the undirected graph G with vertex set $V D$ and edge set $A\,D$ where $\gamma\,a = \{\alpha\,a, \omega\,a\}$.

For the remainder of this section, let G and H be graphs, but most definitions we go through immediately translate to the directed case as well (usually by replacing the word edge by arc or using the underlying graph). As we announced before the definition, we now simplify our notions: we forget about the functions γ, α, and ω that graphs come with, instead identifying an edge with the representation by its ends. This makes it impossible to distinguish a specific parallel arc, but this is not something we shall need. Moreover, we write $v \in G$ and $e \in G$ and read it as $v \in VG$ and $e \in E\,G$.

The *order* of G is $|G| := |VG|$ and its *size* is $\|G\| := |E\,G|$. The graph G is called *simple* if it has neither loops nor parallels.

We mostly deal with undirected graphs, with the only major exception being Part III. For these, we primarily assume that they are simple, making a note in all places where we explicitly allow parallels and loops. Since this is important to remember, let us store it in an environment.

2.1 Assumption. Unless explicitly stated otherwise, all undirected graphs considered in this thesis are simple. ◁

When considering graphs that need not be simple, we refer to them as *multigraphs*.

Two more useful bits of notation are the following: for $X \subseteq VG$, we define $NX = N_G\,X$ as the set of *neighbours of* X, that is,

$$NX := \{v \in VG \setminus X \colon\ v \text{ is adjacent to some } x \in X\}\,.$$

Here, and in all other definitions, we drop braces for single-element sets, letting us write Nv. For $X, Y \subseteq VG$ we write $E(X,Y) = E_G(X,Y)$ for the set of *edges between* X *and* Y, that is,

$$E(X,Y) := \{uv \in E\,G\colon\ u \in X \text{ and } v \in Y\}\,.$$

We also use the shorthand $E\,X$ for $E(X, VG \setminus X)$. The *degree of a vertex* v is equal to the number of incident edges (where loops are counted twice) and we denote it by $d\,v = d_G\,v$. So, in a graph without loops, we have $d\,v = |E\,v|$.

We complete the basics with the handshaking lemma.

(8.3) **2.2 Handshaking Lemma ([Die16]).** *In a graph G, $\sum_{v \in V} d\,v = 2\|G\|$.* ◁

Statements like these will be referred to in the main text by their name. Moreover, this is a good opportunity to explain some conventions we employ. All environments with the exception of proofs end with the symbol ◁ to make it visually clear where they end. Additionally, if we just cite the result, like the one above, the text will be slanted. We also note the references on the margin. These appear throughout this thesis and have the following meaning: if a theorem, for example, comes with references in brackets, then these are used in its proof and references in parentheses make use of the theorem.

Subgraphs and graph operations. The graph H is a *subgraph* of G if $VH \subseteq VG$ and $EH \subseteq EG$. In this situation we call G a *supergraph* of H.

For $\varnothing \neq X \subseteq VG$, $G[X]$ is the *subgraph induced by X*. We write $G - X$ for $G[VG \setminus X]$. Furthermore, for a set X disjoint from VG, we denote the graph with vertex set $VG \cup X$ and edge set EG by $G + X$. For $F \subseteq EG$, $G[F]$ is the *subgraph induced by F*, that is, the subgraph with vertex set VG and edge set F. We write $G - F$ for $G[EG \setminus F]$. Since we forbid subgraphs induced by the empty vertex set, $G[\varnothing]$ is well-defined. If F is a set of potential edges, then we write $G + F$ for the graph $(VG, EG \cup F)$.

The *union* $G \cup H$ and *intersection* $G \cap H$ of G and H are obtained by taking the union and intersection of their vertex and edge sets, respectively.

Let uv be an edge of G. *Subdividing* the edge uv results in the graph G' which is obtained from G by removing uv and adding a new vertex $w \notin VG$ together with the edges uw and wv. We call w a *subdivision vertex*. The inverse operation is the following: given a vertex $w \in VG$ with $d\,w = 2$ and neighbours u and v, *suppressing* w yields the graph $G - w + uv$.

Paths and cycles. A *path* P in G is a sequence $v_0e_1v_1 \dots e_kv_k$ alternating between vertices $v_0, \dots, v_k \in VG$ and edges $e_1, \dots, e_k \in EG$ such that v_{i-1} and v_i are the ends of the edge e_i for $i \in \{1, \dots, k\}$ and all the vertices are different. Usually, we drop the edges in the sequence: they are implicitly given unless we are in the multigraph case, though we also omit them when the choice of edge is not unique, but irrelevant. The *length* of P is k. Additionally, we regard P as a subgraph of G and, as such, we may write $VP = \{v_0, \dots, v_k\}$ and $EP = \{e_1, \dots, e_k\}$. Note that the sequence definition of a path specifies a 'direction' while the subgraph variant has none in undirected graphs. So the two paths $v_0e_1v_1 \dots e_kv_k$ and $v_ke_kv_{k-1} \dots e_1v_0$ correspond to the same subgraph.

We say that P is a *v_0v_k-path* and call the vertices v_0 and v_k the *start* and *end* of P, respectively. The remaining vertices $v_1, \dots, v_{k-1}$ are *inner* vertices of P. For $S, T \subseteq VG$, an ST-path in G is an st-path in G for some $s \in S$ and $t \in T$. The *distance from S to T* is the length of a shortest ST-path in G and denoted by $d(S,T) = d_G(S,T)$. In accordance with our singleton-set policy, the distance from u to v in G is $d(u,v)$.

For the path P and $i \leq j$, we write v_iPv_j for the path $v_ie_{i+1}v_{i+1} \dots e_jv_j$ and call it a *subpath of P*. In case that $i = 0$ or $j = k$ we write Pv_j for v_0Pv_j or v_iP for v_iPv_k, respectively. If $i < j$ we write $\mathring{v}_iPv_j$ for $v_{i+1}Pv_j$ and $v_iP\mathring{v}_j$ for v_iPv_{j-1}. When $i < j - 1$ we also denote $v_{i+1}Pv_{j-1}$ by $\mathring{v}_iP\mathring{v}_j$ and write $\mathring{P}$ for $\mathring{v}_0P\mathring{v}_k$, if allowed.

Let $Q \coloneqq w_0f_1w_1 \dots f_lw_l$ be another path in G. If $v_k = w_0$, we write PQ for the *concatenation* $v_0e_1v_1 \dots e_kv_kf_1w_1 \dots f_lw_l$ of P and Q. While this need not be a path in general since vertices could repeat, we apply this operation only when the result is again a path. Similarly, if h is another index with $v_i = w_h$, then we write Pv_iQ for the concatenation of Pv_i and $v_iQ = w_hQ$. If G is undirected, then the *inverse path* P^{-1} of P is $v_ke_kv_{k-1} \dots e_1v_0$.

We also generalise this notation to trees: since there is a unique path between two vertices u and v in a tree T, we use uTv to denote this path.

A *cycle* $C \coloneqq v_0v_1 \dots v_kv_0$ is a path together with a further edge connecting its ends, that is, $C = P + v_kv_0$ where $P \coloneqq v_0v_1 \dots v_k$ is a path and $v_0v_k \notin P$. Again, we also regard cycles as subgraphs and obtain the notation $V\,C$ and $E\,C$. An edge between two non-adjacent vertices of a cycle C is a *chord* of C.

Some types of graphs. We use the following notation for specific isomorphism classes of graphs:

P_n	A path of order n.
C_n	A cycle of order n.
K_n	A complete graph of order n.
$K_{n,m}$	A complete bipartite graph with parts of size n and m.
E_n	An edgeless graph of order n.

We write $2K_2$ for the disjoint union of two copies of a K_2, so for the union of two edges.

Some graph properties. We now introduce some useful graph properties:

forest	An acyclic graph is a *forest* and its degree 1 vertices are its *leaves*.
star	A *star* is a $K_{1,n}$. The vertex adjacent to all others is the *centre*.
max. degree	The *maximum degree* $\Delta\,G$ *of* G is the maximum degree of any vertex.
min. degree	The *minimum degree* $\delta\,G$ is defined analogously.
regular	A graph G with $d\,v = k$ for all $v \in V\,G$ is k*-regular*, or *cubic* if $k = 3$.
diameter	The maximum distance between two vertices in G is its *diameter* $\operatorname{diam} G$.
girth	The minimum length of a cycle in G is the *girth* of G.
Hamiltonian	A cycle C or path P in G whose vertex set is $V\,G$ is *Hamiltonian.* If G has a Hamiltonian cycle or path, then it is *Hamiltonian* or *traceable.*
H-free	If G has no subgraph isomorphic to a graph H, then G is H*-free.*
connectivity	The graph G is k*-(vertex-)connected* if $\lvert G\rvert > k$ and $G - X$ is connected for all $X \subseteq V\,G$ with $\lvert X\rvert < k$. The graph G is l*-edge-connected* if $G - F$ is connected for all $F \subseteq E\,G$ with $\lvert F\rvert < l$.

Decompositions. A *decomposition* of a graph G consists of subgraphs of G whose edge sets partition $E\,G$. As this notion will see some major use in Part I, we make it a definition.

2.3 Definition. Let G be a graph and $G_1, \dots, G_k$ be subgraphs of G. Then $(G_1, \dots, G_k)$ is a *decomposition* of G if the edge sets of the G_i partition $E\,G$, that is, $\bigcup_{i=1}^{k} E\,G_i = E\,G$ and $E\,G_i \cap E\,G_j = \varnothing$ for $i \neq j$. $\triangleleft$

Note that we are generous with our use of the word *partition*, it allows some sets to be empty, which is often not the case.

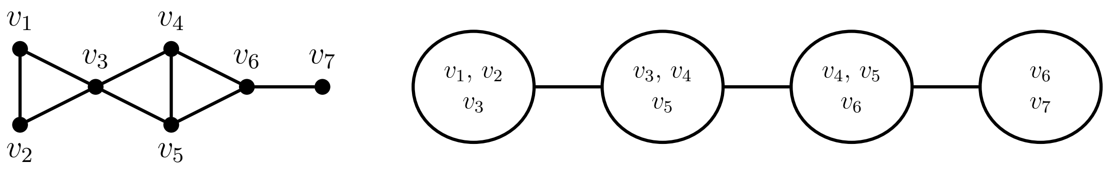


Figure 2.1.: An example of a graph of path-width 2 on the left and a corresponding path-decomposition on the right. This path-decomposition is not smooth, but can be made to be by inserting the bag $\{v_2, v_3, v_4\}$ between the first and the second bag on the path and adding the vertex v_5 to the last bag.

Tree- and path-width. We now get to a graph parameter that is central for this thesis: the path-width. Since it goes hand-in-hand with tree-width, we define both here.

Let T be a tree and $\mathcal{V} = (V_t)_{t\in T}$ be a family of vertex sets $V_t \subseteq VG$. The pair $(T, \mathcal{V})$ is a *tree-decomposition* of G if it satisfies the three properties below.

(i) Every vertex $v \in VG$ is contained in some set V_t.
(ii) For each edge $uv \in EG$ there exists a set V_t such that $\{u, v\} \subseteq V_t$.
(iii) The set $\{t\colon\ v \in V_t\}$ is connected for all $v \in VG$.

The sets V_t are called *bags* of the decomposition $(T, \mathcal{V})$ and the value $\max\{|V_t|\colon\ t \in T\}-1$ is its *width*. Furthermore, the *tree-width* $\operatorname{tw} G$ of a graph G is the minimal width of any of its tree-decompositions. Note that tree-decompositions are *not* decompositions in the sense of Definition 2.3.

If the tree above is required to be a path, the resulting decomposition is a *path-decomposition* and its *width* as well as the *path-width* of G are defined analogously and denoted by $\operatorname{pw} G$. An easy example of a graph of path-width 2 and a corresponding path-decomposition can be found in Figure 2.1.

A tree-decomposition (or path-decomposition) $(T, \mathcal{V})$ of width k is *smooth* if

(i) all bags have cardinality $k + 1$ and
(ii) $|V_{t_1} \cap V_{t_2}| = k$ for all $t_1t_2 \in T$.

2.4 Theorem ([Bod98]). *Let G be a graph. There exists a smooth tree-decomposition of G with width $\operatorname{tw} G$. The same holds true for path-decompositions.* ◁ (2.11) (5.20)

The decomposition in Figure 2.1 can be made smooth by adding a new vertex to the path, between the first and second one, whose associated bag is $\{v_2, v_3, v_4\}$. Additionally, the last bag needs an additional vertex, adding v_5 does the job, for example.

The following shorthand is very useful and thus deserves to be set apart.

2.5 Definition. Given a smooth path-decomposition $(P, \mathcal{V})$ of the graph G with a path $P = 1 \dots n$, then there exists a unique vertex in $V_i \setminus V_{i-1}$ for $i \in \{2, \dots, n\}$, which we call the *vertex entering* V_i. Similarly, we call the unique vertex in $V_i \setminus V_{i+1}$, for $i \in \{1, \dots, n-1\}$, the *vertex leaving* V_i. ◁

Again, we refer to Figure 2.1, where we saw how to make the path-decomposition smooth. The second and third bag in the figure remain unchanged in the smooth version, and the vertex leaving the second bag is v_3, while the vertex entering the third is v_6.

Connectivity. Like decompositions, the next definition is used frequently in Part I.

2.6 Definition. A cycle C in a graph G is *non-separating* if $G - E\,C$ is connected. $\triangleleft$

We also recall some basic connectivity results. A *cut-vertex* of G is a vertex v such that $G - v$ has more components than G. A *block* of G is an inclusion-wise maximal connected subgraph without a cut-vertex, making the K_1 and K_2 the only potential blocks that are not 2-connected. Every edge of G lies in a unique block and different blocks have at most one (cut-)vertex in common. The *block-cutpoint graph* of G has a vertex for every block and one for every cut-vertex of G, which are connected if the cut-vertex is contained in the block.

(3.6) **2.7 Theorem ([HT73]).** *The block-cutpoint graph is a tree and can be computed in linear time.* $\triangleleft$

If G is connected and its block-cutpoint graph consists of more than one vertex, it has at least two leaves, which correspond to blocks with exactly one cut-vertex. We call these *leaf blocks*.

The next result we need is Menger's theorem, which has many formulations, of which we present just one.

(4.25) **2.8 Menger's Theorem ([Men27, cf. Die16]).** *Let s, t be two vertices in a graph G*
(5.6) *with $st \notin G$. The minimum number of vertices required to separate s and t is equal to*
(5.7) *the maximum number of internally vertex-disjoint st-paths in G.* $\triangleleft$

Paths P, Q are *internally vertex-disjoint* if $\mathring{P}$ and $\mathring{Q}$ have no vertices in common. From this theorem, it follows that G is k-connected if and only if there are (at least) k internally vertex-disjoint paths between any pair of vertices.

We also wish to recall the max-flow min-cut theorem. Since we only need the result, we refrain from introducing flow notation here and just refer to [AMO93].

(5.17) **2.9 Max-Flow Min-Cut Theorem ([DF55, cf. AMO93]).** *Let D be a digraph together with a source s and a sink t. The value of a maximum st-flow in D is equal to the capacity of a minimum st-cut.* $\triangleleft$

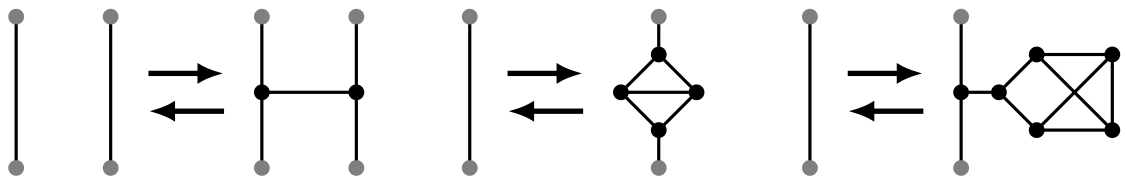


Figure 2.2.: The operations described in Theorem 2.12 to generate all cubic graphs starting from a K_4. In all cases, the grey vertices are those with neighbours outside of the drawn configuration. Operations (i) to (iii) are displayed from left to right and we note that the grey vertices need not be distinct in the subdivision operation (on the left).

Cubic graphs. Since cubic graphs play an important role in Parts I and II, we present some useful results concerning these. First, we take a look at their behaviour with respect to connectivity and path-width.

2.10 Observation. For cubic graphs, vertex- and edge-connectivity coincide. $\triangleleft$ [Die16] (8.12)

Proof. The vertex-connectivity is always a lower bound for the edge-connectivity, see [Die16, Proposition 1.4.2]. For the other inequality, suppose $G - X$ is disconnected for some $X \subseteq VG$ and let $U_1 \cup U_2$ be a partition of $VG \setminus X$ with $E(U_1, U_2) = \varnothing$. Every vertex $x \in X$ has at most one neighbour in one of these two sets, and adding it to the other yields a partition $V_1 \cup V_2$ of VG with at most $|X|$ edges in $E(V_1, V_2)$. $\square$

2.11 Observation. Cubic graphs have tree- and path-width at least 3. $\triangleleft$ [2.4] (8.12)

Proof. This follows directly from graphs of tree-width at most k having a vertex of degree at most k, which can be seen by considering the bag of a leaf in a smooth tree- or path-decomposition. $\square$

We now cite the characterisations of (3-connected) cubic graphs by Tutte and Wormald.

2.12 Theorem (Wormald's Characterisation [Wor79]). *Apart from the K_4, every connected cubic graph can be obtained from a smaller cubic graph by one of the following three operations, which are shown in Figure 2.2.* (8.13)

(i) Subdivide two edges and join the subdivision vertices by a new edge.
(ii) Replace an edge by a diamond and join the degree-2 vertices of the diamond with the former ends of the replaced edge.
(iii) Take the disjoint union with a K_4, subdivide an edge in each component of this union and join the subdivision vertices.

Moreover, every graph obtained from the K_4 by a repeated application of these operations is connected and cubic. $\triangleleft$

2.13 Theorem (Tutte's Characterisation [Tut66, cf. Tut19]). *Excluding the K_4, every 3-connected cubic graph can be obtained from a smaller cubic graph using Operation (i) and every graph obtained this way is 3-connected and cubic.* $\triangleleft$ (4.24) (8.16)

Automorphisms and actions. Let φ be a bijection with domain VG. For $U \subseteq VG$ and $F \subseteq EG$, we define $\varphi U := \{\varphi u \colon u \in U\}$ and $\varphi F := \{(\varphi u)(\varphi v) \colon uv \in F\}$. We write φG for the graph with vertex set $\varphi(VG)$ and edge set $\varphi(EG)$. The map φ is an *isomorphism* from G to H if $H = \varphi G$. The graph G is *isomorphic* to H, denoted by $G \cong H$, if an isomorphism from G to H exists. If $\varphi G = G$, then φ is an *automorphism* of G. The automorphisms of a graph G together with the composition of maps form a group and every subgroup Γ of this group of automorphisms acts on VG by mapping (φ, v) to φv. Let $v \in VG$. The *orbit* of v (under the action of Γ) is the set $v^\Gamma := \{\varphi v \colon \varphi \in \Gamma\}$ and the *stabiliser* of v is $\Gamma_v := \{\varphi \in \Gamma \colon \varphi v = v\}$. We also recall the orbit stabiliser theorem, in this particular setting.

2.14 Orbit-Stabiliser Theorem ([Rot95]). *Let G be a graph, Γ be a subgroup of the automorphism group of G, and $v \in VG$. Then*

$$|v^\Gamma| \cdot |\Gamma_v| = |\Gamma|. \qquad \lhd$$

2.2. Complexity theory

Complexity classes and undecidability. The most-used complexity classes in this thesis are P and NP, which are luckily also the most well-known. Aside from these two classes, we need the class $\Sigma_2\mathsf{P}$ in Chapter 10. All three of these classes are in the first three levels of the polynomial hierarchy. As a very rough overview, a language L is in

P if there exists a deterministic polynomial-time Turing machine M such that

$$a \in L \iff M(a) = 1.$$

NP if there exists a deterministic polynomial-time Turing machine M and a polynomial p such that

$$a \in L \iff \exists x \in \{0,1\}^{p(|a|)} : M(a, x) = 1.$$

$\Sigma_2\mathsf{P}$ if there exists a deterministic polynomial-time Turing machine M and a polynomial p such that

$$a \in L \iff \exists x \in \{0,1\}^{p(|a|)} : \forall y \in \{0,1\}^{p(|a|)} : M(a, x, y) = 1.$$

We also make use of the fact that $\Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$, meaning that the languages in $\Sigma_2\mathsf{P}$ are exactly those languages decidable by a polynomial-time Turing machine with access to an oracle for some NP-complete problem [AB09, Remark 5.16]. We refer to [GJ90] for more information on the classes P and NP. In particular, it provides a list of many NP-complete problems, one of which we define shortly. For $\Sigma_2\mathsf{P}$ we refer to [AB09, Pap94, Haa19].

Complete problems. To show that problems are complete for the classes NP and $\Sigma_2\mathsf{P}$, we present (polynomial-time) Karp reductions from the complete problems we present here. For these, we note that a Boolean formula φ is in 3-CNF if it is the conjunction of clauses and each clause is the disjunction of exactly three literals. Similarly, it is in 3-DNF if it is the disjunction of clauses which are the conjunction of three literals.

2.15 Problem (3-SAT). Given a formula φ in 3-CNF depending on variables x, does a truth assignment to the x-variables exist that satisfies φ? $\triangleleft$

2.16 Problem (Σ_2SAT). Given a formula φ in 3-DNF depending on variables x and y, does a truth assignment to the x-variables exist that satisfies φ for all possible truth assignments to the y-variables? $\triangleleft$

The 3-SAT problem is NP-complete and Σ_2SAT is $\Sigma_2\mathsf{P}$-complete, see [GJ90, Problem LO2] and [Pap94, Theorem 17.10] or [Haa19, Section 2.2.1], respectively.

Undecidability. Finally, we also need the notion of undecidability. A language L is undecidable if there exists no Turing machine M that terminates on all inputs a and outputs 1 if and only if $a \in L$. To show undecidability later, we use the Post correspondence problem, which is undecidable [Pos46].

2.17 Problem (Post correspondence problem). Let $x_1, \ldots, x_N$ and $y_1, \ldots, y_N$ be strings over the alphabet $\{a, b\}$. Does a finite sequence $(i_1, \ldots, i_k)$ of indices exist such that $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$? $\triangleleft$

2.3. Local certification

We formally define local certification here. It is mainly needed in Chapter 9, but also in a part of Chapter 7. For a motivation, an intuitive description of the concept, and information on the corresponding literature, we refer to Chapter 9.

Provers and verifiers. Let $\mathcal{G}$ be a class of (directed or undirected) graphs, for example, $\mathcal{G}$ could be the class of all connected undirected graphs. Further, let $\mathcal{F} \subseteq \mathcal{G}$ be the graphs in $\mathcal{G}$ that satisfy a certain property (bipartiteness, for example). In this context we specify that graphs $G \in \mathcal{G}$ have an *identity* for every vertex $v \in VG$ which can be encoded in $\mathcal{O}(\log(|G|))$ bits. All identities in a graph are distinct, making them an injective map from VG to $\{0,1\}^{c\log(|G|)}$ for some constant c. Furthermore, vertices also have *labels*, which can contain further information (but may be empty). For example, these can be used to indicate colours of vertices or to select a certain subset of incident edges.

A *proof* for a graph G is a function $\mathbf{P}\colon VG \to \{0,1\}^*$ that assigns a binary *certificate* (a bit string) to each vertex of G. The *size* $|\mathbf{P}|$ of a proof is the maximum number of bits in any of its certificates and the set of all proofs for a graph G is denoted by $\mathbb{P}\,G$. For the empty certificate and the empty label we use the usual notation for the empty word: ε.

A *verifier* for a class $\mathcal{G}$ is a function $\mathcal{V}$ defined for all triples $(G, \mathbf{P}, v)$ with $G \in \mathcal{G}$, $\mathbf{P} \in \mathbb{P}\,G$, and $v \in VG$. Its output is a single bit, that is,

$$\mathcal{V}\colon \bigcup_{G \in \mathcal{G}} (\{G\} \times \mathbb{P}\,G \times VG) \to \{0, 1\}\,.$$

A verifier $\mathcal{V}$ *accepts* (a proof $\mathbf{P}$) at a vertex v if $\mathcal{V}(G, \mathbf{P}, v) = 1$ and rejects if $\mathcal{V}(G, \mathbf{P}, v) = 0$. If $\mathcal{V}$ accepts at all $v \in VG$, then it accepts (a proof $\mathbf{P}$ for) a graph G and it *rejects* the proof otherwise. The verifier is *sound* for a subset $\mathcal{F}$ of $\mathcal{G}$ if it rejects any graph not in $\mathcal{F}$, regardless of the proof provided, that is, for all $G \in \mathcal{G} \setminus \mathcal{F}$ and all proofs $\mathbf{P} \in \mathbb{P}\,G$, there exists a vertex $v \in VG$ such that $\mathcal{V}(G, \mathbf{P}, v) = 0$.

A *prover* is a function $\mathcal{P}$ that maps every $G \in \mathcal{F}$ to a proof $\mathcal{P}\,G \in \mathbb{P}\,G$. The *size* $s\,\mathcal{P}$ of a prover is the maximum size of any proof it assigns to a graph in $\mathcal{F}$, often expressed as a function of $|G|$. We note that, technically, the proofs assigned by a prover have access to $2^{s\,\mathcal{P}+1} - 1$ many different certificates, but we will assume they only use the $2^{s\,\mathcal{P}}$ many of length exactly $s\,\mathcal{P}$ as this is convenient and commonplace in the literature. A prover $\mathcal{P}$ is *complete* for a verifier $\mathcal{V}$ if $\mathcal{V}$ accepts $\mathcal{P}\,G$ for all $G \in \mathcal{F}$. Note that the label of a vertex v is part of the graph G whereas its certificate is provided by the prover.

A pair $\pi = (\mathcal{P}, \mathcal{V})$, consisting of a prover and a verifier, is a *local certification* for $\mathcal{F} \subseteq \mathcal{G}$ if $\mathcal{P}$ is complete for $\mathcal{V}$ and $\mathcal{V}$ is sound for $\mathcal{F}$. The *size* $s\,\pi$ of π is the size of its prover $\mathcal{P}$.

An example. To illustrate these concepts, we take a look at an easy example. We can verify whether a graph is bipartite as follows [GS16]: the prover, given a bipartite graph, specifies a bipartition using the certificates 0 and 1. The verifier then only needs to check, at each vertex, that the certificate of this vertex differs from the certificates of its neighbours. If this is the case everywhere, then the graph is bipartite, so the verifier never accepts a non-bipartite graph. Furthermore, the bipartition specified by the prover makes the verifier accept. Hence, this prover-verifier pair is a local certification for bipartiteness using a single bit.

Restricting the verifier. In the example, the verifier only looked at its direct neighbours. However, so far, the definition does not include locality and this is where the various concepts that are used in the literature differ. We now describe restrictions on the verifier that make its computation local and lead to the definition of proof labelling schemes and locally checkable proofs.

Let G be a graph and $v \in VG$. The set of vertices in G with distance at most r (with respect to number of edges) to v is denoted by $B^r_G\,v$ or just $B^r\,v$ and called the *ball of* *radius r at v.* For directed graphs, we always use the underlying undirected graph to determine balls. A verifier is *r-local* if it satisfies that

$$\mathcal{V}(G, \mathbf{P}, v) = \mathcal{V}(G_v^r, \mathbf{P}_v^r, v) \text{ for all } G, \mathbf{P}, v$$

where $G_v^r = G[B^r\, v]$ and $\mathbf{P}_v^r = \mathbf{P}[B^r\, v]$ is the restriction of $\mathbf{P}$ to $B^r\, v$. We say a verifier is *neighbourhood-local* if

$$\mathcal{V}(G, \mathbf{P}, v) = \mathcal{V}(G_v^N, \mathbf{P}_v^1, v) \text{ for all } G, \mathbf{P}, v$$

where $G_v^N = (B^1\, v, E\, v)$. If the verifier does not depend on the labels or identities of any vertex other than v when faced with $(G, \mathbf{P}, v)$, then we call the verifier *label-* or *identity-restricted*, respectively. We say it is *restricted*, if it is both label- and identity-restricted and satisfies that

$$\mathcal{V}(G, \mathbf{P}, v) = \mathcal{V}(G_v^-, \mathbf{P}_v^-, v) \text{ for all } G, \mathbf{P}, v$$

where G_v^- is the star with v at its centre and an edge vx_e for each edge $e = vu$ in G. (For digraphs vx_a or $x_a v$ is used for arc a as appropriate.) The proof $\mathbf{P}_v^-$ maps v to $\mathbf{P}\, v$ and x_e to the certificate $\mathbf{P}\, u$ if $e = vu$ or $e = uv$. The difference between this construction and the graph G_v^N is that it also hides parallels (and anti-parallels) since it 'splits' such vertices.

With this notation we can now define proof labelling schemes and locally checkable proofs.

2.18 Definition. A local certification for $\mathcal{F} \subseteq \mathcal{G}$ whose verifier is restricted is a *proof labelling scheme.* $\triangleleft$

2.19 Definition. A local certification for $\mathcal{F} \subseteq \mathcal{G}$ whose verifier is r-local for some constant $r \geq 1$ is a *locally checkable proof.* $\triangleleft$

A few remarks on these definitions are in order. Our notation is mostly based on the paper by Göös and Suomela [GS16], who study locally checkable proofs. Sadly, the notation is inconsistent with that of Korman, Kutten, and Peleg [KKP10] who use the term 'label' for what we have called a certificate. Thus, they named their concept 'proof labelling schemes', which is a misleading name here. However, in order to adhere to the literature, we keep this name and emphasise here that these schemes do not actually set the labels, but the certificates, despite their name.

Moreover, we note that, in a proof labelling scheme, the verifier at a vertex v may only use v's identity and label, as well as the certificates assigned to v and its neighbours. As such, the local certification we presented in the example above meets the requirements of a proof labelling scheme. In contrast, the verifier in a locally checkable proof has access to much more information. Even if we restrict ourselves to a 1-local verifier, it can still use the identities and labels of the neighbours as well as the graph structure of the neighbourhood, meaning it knows which of its neighbours are the same and whether or not they are adjacent.

We introduced the term *neighbourhood-local*, which comes between the two. In contrast to proof labelling schemes, the verifier here can detect parallels and anti-parallels, but, unlike locally checkable proofs, it is still limited to direct neighbours and does not know which of these are adjacent. This is needed in Chapter 9.

A local class of graphs. For Chapter 7 we also introduce the term highly local.

2.20 Definition. Let $\mathcal{G}$ be the class of all undirected graphs. A class $\mathcal{F} \subseteq \mathcal{G}$ is *highly local* if there exists a local certification $\pi = (\mathcal{P}, \mathcal{V})$ for $\mathcal{F} \subseteq \mathcal{G}$ with the following properties:

- $\mathbf{P}$ has constant size s,
- $\mathcal{V}$ is r-local and identity-restricted, and
- $\mathcal{V}$ is *invariant under isomorphism*, that is, for every $G \in \mathcal{G}$, $v \in VG$, and every bijection φ with domain VG we have that

$$\mathcal{V}(G, \mathbf{P}, v) = \mathcal{V}(\varphi\, G, \varphi\, \mathbf{P}, \varphi\, v)$$

  where $\varphi\, \mathbf{P} \colon \varphi\, G \to \{0,1\}^*$ is the map satisfying $\varphi\, \mathbf{P}(\varphi\, v) = \mathbf{P}\, v$. ◁

Examples for highly local graph classes are the bipartite graphs (as we saw in the example) or, more generally, l-colourable graphs (using $r = 1$, $s = \lceil \log l \rceil$). Moreover, l-regular graphs (with $r = 1$, $s = 0$) and graphs without an (induced) subgraph H (using $r = \left\lceil \log \frac{\operatorname{diam} H}{2} \right\rceil$, $s = 0$) are further examples.

Part I.

The 3-Decomposition Conjecture

In this part, we present new results concerning the 3-decomposition conjecture, which states that any connected cubic graph can be decomposed into a spanning tree, cycles, and a matching. In Chapter 3 we survey known and prove some preliminary results. The latter include an equivalent conjecture, as well as an efficient algorithm for computing a weaker decomposition in which the matching part is replaced by paths of length at most 2. We generalise the result that all Hamiltonian cubic graphs have a 3-decomposition to the class of star-like cubic graphs in Chapter 4. We then determine several reducible configurations (subgraphs that can be shrunk without hurting the existence of 3-decompositions) in Chapter 5. Using these, we derive properties of minimum counterexamples, in particular, we determine that graphs of path-width at most 4 satisfy the conjecture.

Chapter 3.

Known and Preliminary Results

The 3-decomposition conjecture, by Hoffmann-Ostenhof, states that every connected cubic graph can be decomposed into a spanning tree, a 2-regular subgraph, and a matching. In this chapter, we consider known equivalent formulations of this conjecture and add a new one to the list. We also review for which classes of graphs the conjecture is known to hold and broadly sort these into two groups (a 'local' and 'global' one), based on the proof techniques employed. Next, we deviate a bit from the conjecture and study the complexity of finding non-separating cycles, in particular, we show that in a cubic graph, one such cycle can be found in linear time. We use this result to efficiently compute a relaxation of a 3-decomposition, in which the matching part is replaced by disjoint paths of length at most 2.

Removing the edges of a spanning tree from a cubic graph results in a graph of maximum degree 2, which is necessarily a union of vertex-disjoint paths and cycles. The 3-decomposition conjecture, postulated by Hoffmann-Ostenhof, see [Hof11, Cam11], asserts that the spanning tree can be chosen so that all of the arising paths are of length 0 or 1.

3-Decomposition Conjecture. Every finite connected cubic graph has a decomposition consisting of a spanning tree, a 2-regular subgraph, and a matching. $\triangleleft$

Recall Definition 2.3: a decomposition of a graph G is a tuple of edge-disjoint subgraphs whose union is G. Any such decomposition (T, C, M), consisting of a spanning tree T, a 2-regular graph C, and a matching M, is called a *3-decomposition.* Note that formally the last component is a subgraph whose edge set is a matching, but, to adhere to the literature, we ignore this distinction. Moreover, the matching may be empty (and a simple counting argument shows that the cycle component is not).

Decompositions into trees and graphs of small maximum degree are of general interest and find applications, for example, in determining upper bounds for the game chromatic number [He+02]. Several such decompositions are known, for example, Balogh et al. [Bal+05] showed that it is possible to decompose a planar graph into three forests such that one of them has degree at most 8 and Gonçalves [Gon09] presented decompositions into two forests and a graph of maximum degree at most 4. By requiring the girth to be at least 8, Wang and Zhang [WZ11] found a decomposition into a forest and a matching.

We also note that many conjectures in graph theory consider or can be reduced to (bridgeless) cubic graphs. Thus, obtaining structural information about these is of interest. A prominent example is the cycle double cover conjecture [Jae85], three others are the Berge-Fulkerson [Ful71], the shortest cycle cover [AT85], and the Fan-Raspaud conjectures [FR94].

The 3-decomposition conjecture is the topic of this entire part, but in this chapter, we start by taking a look at three equivalent reformulations in Section 3.1, the last of which is new and would provide a constructive way of obtaining all cubic graphs. Next, we review the literature to see which classes of graphs are known to have 3-decompositions in Section 3.2. We also use this opportunity to discuss the proof techniques used and to sort these into two groups, which can be roughly summarised as ‘using local arguments’ and ‘using global arguments’.

Lastly, we investigate a relaxation of a 3-decomposition in Section 3.4, in which the third component is allowed to contain paths of length 1 and 2. Compared to a matching, which is a collection of paths of length 1, the last value is new. These decompositions are known to exist, and we provide a linear time algorithm to compute them. This requires a more detailed look at the complexity of finding non-separating cycles in a graph, which we study in Section 3.3. It turns out that this problem is NP-hard, even on subcubic graphs, but it is possible to find one in linear time in a graph with minimum degree at least 3.

The content of Sections 3.1 and 3.2 is joint work with Irene Heinrich and is included in [BH21] on arXiv. The last two sections are unpublished.

3.1. Equivalent conjectures

We start this section with two known equivalent reformulations of the 3-decomposition conjecture. These are the *strong 3-decomposition conjecture* and the *2-decomposition conjecture*, both of which can be found in [Hof15].

Strong 3-Decomposition Conjecture. Let G be a connected cubic graph and C' be a non-separating 2-regular subgraph of G. Then there exists a 3-decomposition (T, C, M) of G such that $C' \subseteq C$. ◁

2-Decomposition Conjecture. Let G be a connected graph in which all vertices have degree 2 or 3 and every cycle is separating (recall Definition 2.6). Then G has a decomposition into a spanning tree and a matching. ◁

We note that replacing the degree condition by the adjective subcubic, which allows vertices of degree 1, does not strengthen the conjecture. The edge incident to any vertex of degree 1 is automatically part of the tree. Thus, by (iteratively) removing such vertices we obtain a subgraph that has minimum degree 2 and a decomposition there can be extended to the entire graph since all deleted edges must be added to the tree.

Hoffmann-Ostenhof, Kaiser, and Ozeki [HKO18] prove that the 3- and 2-decomposition conjectures are equivalent. That the latter implies the former can be seen as follows: remove the edges of a maximal non-separating 2-regular graph C (in the sense of Definition 2.6) to obtain a graph as required for the 2-decomposition conjecture. It then provides the spanning tree and the matching.

For the missing implication, we attach a small graph to each degree 2 vertex of a subcubic graph G such that the new edge is a bridge (see Operation (iii) of Theorem 2.12). A 3-decomposition of this new graph has no cycles in G since these are separating (in the larger graph as well). Thus, the restricted decomposition is as desired.

This also gives us the result for the strong 3-decomposition conjecture. One implication is simply obtained by letting C' be the null graph. For the other, we are given a non-separating 2-regular graph C'. By taking a maximal non-separating 2-regular supergraph C of C' and using the previous equivalence, $G - E\,C$ has a decomposition into a spanning tree and a matching. This yields a 3-decomposition that contains C' in its 2-regular subgraph.

We complete this section by presenting a new equivalent conjecture: the HIST-extension conjecture. A HIST is a *homeomorphically irreducible spanning tree*, that is, a spanning tree without vertices of degree 2. For an overview of the study of HISTs, see [CS13]. Furthermore [HNO18] provides a necessary condition for their existence specifically in the setting of cubic graphs.

Let us begin by explaining the conjecture and formally stating it. Essentially, the HIST-extension conjecture claims that every cubic graph G can be constructed from a smaller cubic graph G' that has a HIST using just two operations. More precisely, the graph G can be obtained by starting with a graph G' that has a HIST T', colouring exactly the edges of T' in green, and then iteratively applying Tutte- and diamond-extensions. These operations are shown in Figure 3.1. If we do not require the graphs to be simple, then it already suffices to only use Tutte-extensions, which are now also allowed to choose the same edge twice. In both cases, the extensions have to respect the edge colours.

We can now state the conjecture, where we call an extension (and reduction) *simple* if it results in a simple graph.

HIST-Extension Conjecture. Let G be a connected cubic graph. There exists a cubic graph G' admitting a HIST $T' \subseteq G'$ such that, if exactly the edges of T' are coloured green, then G can be obtained from G' by a finite sequence of simple Tutte- and diamond-extensions (see Figure 3.1). ◁

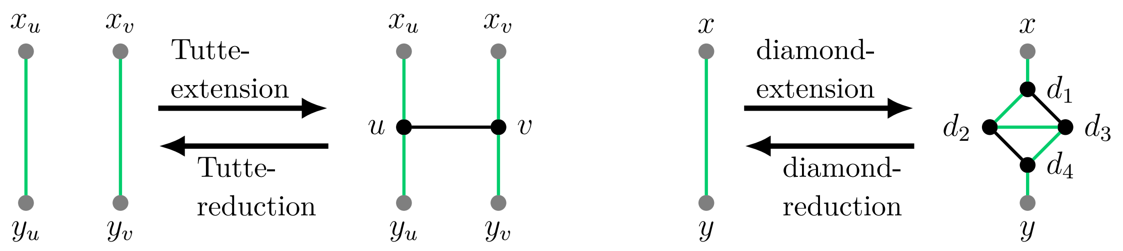


Figure 3.1.: Coloured extensions and reductions. Grey vertices are those with neighbours outside of the configuration. A *Tutte-extension* takes two green edges, subdivides them, and connects the resulting subdivision vertices by an uncoloured (black) edge. Note that the two chosen edges may be incident and, if we do not restrict to simple graphs, they need not even be different. This last case creates parallels. On the other hand, a *diamond-extension* inserts a diamond onto a green edge xy, that is, it removes xy, inserts new vertices d_1, d_2, d_3, and d_4, as well as edges xd_1, d_1d_2, d_2d_3, d_3d_4, d_4y, d_1d_3, and d_2d_4, where all but the last two are green.

The version for multigraphs drops the word 'simple' and the diamond reduction.

HIST-Extension Conjecture (Multigraph Version). Let G be a connected cubic multigraph. There exists a cubic multigraph G' admitting a HIST $T' \subseteq G'$ such that, if exactly the edges of T' are coloured green, then G can be obtained from G' by a finite sequence of Tutte-extensions (see Figure 3.1). ◁

We wish to emphasise the strong similarities between the HIST-extension conjecture and Wormald's characterisation of cubic graphs. The first two operations in Wormald's theorem are the same as ours, with the exception that they need not conform to a colouring. Thus, the difference between the theorems is that the HIST-extension conjecture restricts the operations that may be performed and instead allows a substantially larger (infinite) ground set, namely all cubic graphs containing a HIST, instead of just the K_4.

Let us now prove the equivalence. First we note that if a cubic graph G has a HIST T, then it satisfies the 3-decomposition conjecture, as all vertices in $G - E\,T$ have degree 0 or 2. Therefore, using a HIST, we obtain a decomposition into a spanning tree and a 2-regular subgraph. The idea of the proof is to show that our allowed extensions result in new (green) trees that are part of a 3-decomposition (Lemma 3.1) and we then show how to reduce a 3-decomposition in a graph without a HIST to one in a smaller graph, which may or may not have one (Theorem 3.2). This is done by using a Tutte- or diamond-reduction and, inductively, this smaller graph is obtained from one with a HIST. Thus, the original graph is as well.

We begin by showing that 3-decompositions are preserved by the operations described.

(3.3) **3.1 Lemma.** Let G' be a cubic multigraph with a spanning tree T'. Colour exactly the edges of T' green. If G and T are obtained from G' and T' by either a Tutte- or a diamond-extension (see Figure 3.1), then T is a tree. Moreover $G - E\,T$ decomposes into cycles and a matching if and only if $G' - E\,T'$ does. ◁

Proof. First observe that the respective extensions and reductions preserve the property that the green edges form a spanning tree (both only insert subdivision vertices on green edges and connect them by black edges). Since $G' - E\,T'$ is a graph of maximum degree 2 it decomposes into a 2-regular graph C' and a disjoint union of paths P'. If G is obtained from G' by a Tutte-extension, then G decomposes into T, the 2-regular graph C', and the disjoint union of P' with an additional K_2. Otherwise, G is a diamond-extension of G' and decomposes into T, C', and the disjoint union of P' with a $2K_2$. In both cases P' is a matching if and only if $G - E\,T - E\,C'$ is a matching. □

The lemma above gives us one direction of the equivalence, namely that any cubic graph constructed this way has a 3-decomposition. For the converse, we show that we can perform a reduction somewhere in a 3-decomposition, unless its tree is a HIST.

3.2 Theorem. Let G be a cubic multigraph with a 3-decomposition (T, C, M) in which exactly the edges of T are coloured green. If T is a HIST, then $M = \varnothing$. (3.3)

Otherwise, G with its 3-decomposition (T, C, M) can be obtained by a Tutte-extension from a smaller cubic multigraph G' with 3-decomposition (T', C, M') where exactly the edges of T' are green. In particular, every 3-decomposition can be reduced to a 3-decomposition with a HIST by a finite sequence of Tutte-reductions.

Analogously, if G is simple, it can be obtained by a simple Tutte- or diamond-extension from a smaller simple cubic graph G' with 3-decomposition (T', C, M') and thus, every 3-decomposition can be reduced to a 3-decomposition with a HIST by a finite sequence of simple Tutte- and diamond-reductions. ◁

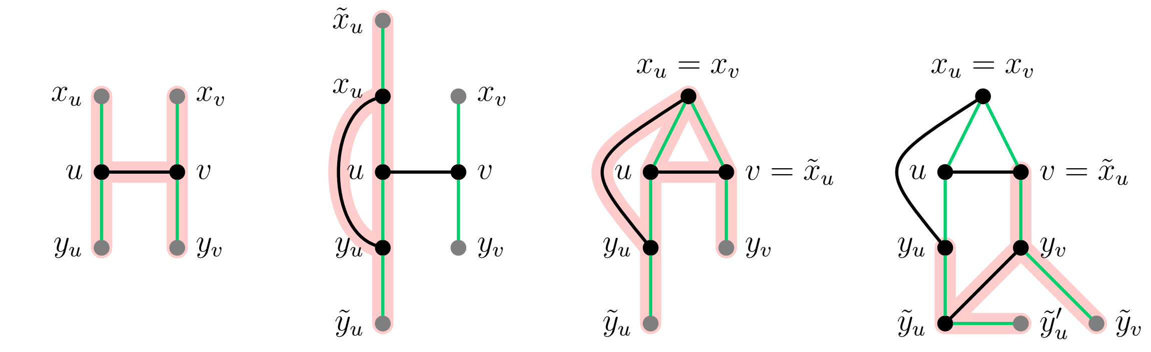
Proof. In this proof, a graph G is always given with a 3-decomposition (T, C, M). The edges of T are coloured green and all other edges are black. If T is homeomorphically irreducible, then $G - E\,T$ only has vertices of degree 0 and 2. In particular, $G - E\,T$ is a vertex-disjoint union of isolated vertices and cycles. Hence, $M = \varnothing$.

From now on, we may assume that T contains a vertex v with $d_T\, v = 2$. Since (T, C, M) is a 3-decomposition of G, we obtain that v is incident to an M-edge uv in G and, in particular, $u \neq v$. Moreover, u has degree 2 in T as well, and we may apply a Tutte-reduction (potentially yielding a single edge), completing the first part of the theorem.

We may now assume that G is simple and obtain that both u and v differ from their neighbours. We denote these neighbours by x_u, y_u, x_v, and y_v respectively as depicted in Figure 3.2a. Since uv is an M-edge, the edges ux_u, uy_u, vx_v, and vy_v are T-edges.

We can apply a Tutte-reduction here unless one of the edges x_uy_u or x_vy_v is present in G since these would create parallels. Hence, if neither of these edges is in G, then G and (T, C, M) can be obtained via a Tutte-extension from $G - \{u, v\} + \{x_uy_u, x_vy_v\}$ with the 3-decomposition $(T - \{x_uu, uy_u, x_vv, vy_v\} + \{x_uy_u, x_vy_v\}\,, C, M \setminus \{uv\})$.

By symmetry, we may assume that $x_uy_u \in E\,G$. This implies that x_uy_u is an M-edge: a T-edge would cause the cycle ux_uy_uu in T whereas a C-edge would disconnect the



(a) A degree 2 vertex. (b) $\tilde{x}_u$, $\tilde{y}_u \neq v$. (c) $\tilde{x}_u = v$. (d) $\tilde{y}_u y_v \in G$.

Figure 3.2.: Key steps in the proof of Theorem 3.2. The subgraphs shown here represent an expanding local view of the entire graph G where the green edges are exactly those in the tree part of the 3-decomposition of G. Note that the vertices x_u, x_v, y_u, y_v need not be distinct, as is seen on the right.

T-edges ux_u and uy_u from the rest of the tree. We may conclude that $x_u\tilde{x}_u$ and $y_u\tilde{y}_u$ are T-edges, where $\tilde{x}_u$ (respectively $\tilde{y}_u$) denotes the unique neighbour of x_u (respectively y_u) outside the triangle ux_uy_uu. In particular, we obtain $\tilde{x}_u \neq \tilde{y}_u$ since T is cycle-free. This is illustrated in Figure 3.2b.

The only obstacle to applying a Tutte-reduction using the M-edge x_uy_u is the case where $\tilde{x}_u = v$ or $\tilde{y}_u = v$ since this creates a parallel to the M-edge uv. Thus, if $v \notin \{\tilde{x}_u, \tilde{y}_u\}$, then G can be obtained by a Tutte-extension of $G' := G - \{x_u, y_u\} + \{u\tilde{x}_u, u\tilde{y}_u\}$ with the 3-decomposition $(T - E(\tilde{x}_u x_u u y_u \tilde{y}_u) + \{\tilde{x}_u u, u\tilde{y}_u\}, C, M \setminus \{x_u y_u\})$.

Otherwise $\tilde{x}_u = v$ or $\tilde{y}_u = v$. Assume that $\tilde{x}_u = v$ (the other case works similarly by interchanging the roles of x and y). Moreover, we assume that $x_u = x_v$ (otherwise rename x_v and y_v). This is illustrated in Figure 3.2c. Observe that $y_v \neq \tilde{y}_u$ since this would cause a cycle in T. If y_v and $\tilde{y}_u$ are not adjacent, then G can be obtained by a diamond-extension of the graph $G - \{y_u, x_u, u, v\} + \{\tilde{y}_u y_v\}$ together with the 3-decomposition $(T - E(\tilde{y}_u y_u u x_u v y_v) + \{\tilde{y}_u y_v\}, C, M \setminus \{y_u x_u, uv\})$. If, otherwise, $y_v\tilde{y}_u \in E\,G$, then this edge is an M-edge (a T-edge would cause a cycle in T and a C-edge would disconnect T). Let $\tilde{y}_v$ be the unique vertex in $N_G\, y_v \setminus \{\tilde{x}_u, \tilde{y}_u\}$ and let $\tilde{y}'_u$ be the unique neighbour of $\tilde{y}_u$ which is not in $\{y_u, y_v\}$. This situation is illustrated in Figure 3.2d. Since y_u and v already have three neighbours, we obtain that $y_u\tilde{y}'_u$ and $v\tilde{y}_v$ are not edges of G. In particular, G can be obtained by a Tutte-extension of $G - \{y_v, \tilde{y}_u\} + \{v\tilde{y}_v, y_u\tilde{y}'_u\}$ with the 3-decomposition $(T - \{\tilde{y}'_u\tilde{y}_u, \tilde{y}_u y_u, v y_v, y_v \tilde{y}_v\} + \{v\tilde{y}_v, y_u\tilde{y}'_u\}, C, M \setminus \{y_v\tilde{y}_u\})$. □

We derive the equivalence of the two conjectures from Lemma 3.1 and Theorem 3.2.

[3.1] [3.2] **3.3 Corollary.** The 3-decomposition conjecture and the HIST-extension conjecture are equivalent. Additionally, the 3-decomposition conjecture holds for all multigraphs if and only if the multigraph version of the HIST-extension conjecture is true. ◁

Proof. Throughout this proof, whenever a graph is given with a spanning tree we implicitly assume that the edges of the tree are green and all other edges are black.

Let G be a connected cubic multigraph. First assume that G satisfies the 3-decomposition conjecture and let (T, C, M) be a 3-decomposition of G. If T is homeomorphically irreducible, then G trivially satisfies the HIST-extension conjecture (with an empty sequence of extensions). Otherwise, by Theorem 3.2, there exists a multigraph G' with a HIST T' such that $(T', C, \varnothing)$ is a 3-decomposition of G' and G can be obtained from G' (with the green-black colouring induced by T') by a finite sequence of Tutte-extensions or simple Tutte- and diamond-extensions, that is, G satisfies the HIST-extension conjecture, in the multigraph or normal version.

Now we assume that G satisfies the HIST-extension conjecture. In particular, there exists a graph G' with a homeomorphically irreducible spanning tree T' such that G can be obtained by a finite sequence of Tutte- (or simple Tutte- and diamond-extensions) from G'. Since all vertices of T' are of degree 1 or 3, the graph $G - ET'$ decomposes into a 2-regular graph C and isolated vertices. Thus, $(T', C, \varnothing)$ is a 3-decomposition of G'. By Lemma 3.1, G satisfies the 3-decomposition conjecture. □

Note that, while seemingly weaker, the HIST-extension conjecture has some advantages over the multigraph variant. First, it already holds if the 3-decomposition conjecture is true for graphs, making it potentially easier to prove (or, in the perhaps unlikely case that the conjecture only holds for simple and not for multigraphs, making it provable at all). It also imposes a further restriction on the Tutte-extension, forbidding those that subdivide the same edge twice (since these do not lead to simple graphs). This is useful for the case that one wants to only deal with graphs, where the operations now guarantee that all of these are obtained without needing to go over intermediate graphs that are not simple.

3.2. Studied classes

Whilst the 3-decomposition conjecture itself is still open, it has been proved for quite a few classes of graphs. In this section, we give an overview of the results that have been obtained and we classify the methods used to obtain them into two types.

Manipulation of a global structure. One common technique that is used to find 3-decompositions is restrict to a certain class of graphs that has some global structure, such as a Hamiltonian cycle, and then use this structure as a starting point to find a 3-decomposition of the graph.

The Hamiltonian case was proved by Akbari, Jensen, and Siggers [AJS15] in 2015. Abdolhosseini et al. [Abd+16] and Li and Liu [LL20] showed that traceability is a sufficient requirement already. Ozeki and Ye [OY16] presented first results for planar graphs, namely that 3-connected cubic graphs satisfy the conjecture if they are planar or on the projective plane. Xie, Zhou, and Zhou [XZZ20] proved the conjecture for graphs with a 2-regular subgraph consisting of three cycles. We take a look at a similar structure in Chapter 4, where we want a 2-regular subgraph whose cycles are arranged in a star, roughly speaking. Last year, Botler et al. [Bot+21] showed that the 2-decomposition conjecture holds if the graphs in question are restricted to ones in which the degree 3 vertices induce a certain collection of cacti. (A cactus is a graph in which every edge is part of at most one cycle.)

Reductions. Another technique finds (small) local structures that behave well with respect to the conjecture. This is often done by assuming that the conjecture does not hold for the class of graphs in question, and taking a minimum counterexample. One then determines that certain structures must appear and that replacing these by smaller ones makes it possible to extend decompositions of the resulting smaller graph to the larger one. By checking that the replacement preserves containment in the class, a contradiction is derived because the smaller graph would also need to be a counterexample.

Let us provide some classes where these local arguments are used: as an extension of [OY16], Bachstein [Bac15] deals with 3-connected cubic graphs on the Torus and Klein Bottle and Hoffmann-Ostenhof, Kaiser, and Ozeki [HKO18] showed that all connected plane cubic graphs satisfy the conjecture in 2018. It was also proved to hold for claw-free (sub)cubic graphs by Aboomahigir, Ahanjideh, and Akbari [AAA18] and Hong, Liu, and Yu [HLY20]. Heinrich [Hei19] verified it for 3-connected cubic graphs of (at most) tree-width 3, and we extend this technique to get the analogous result for path-width at most 4 in Chapter 5.

3.3. Finding non-separating cycles

In this section, we investigate the complexity of finding non-separating cycles (see Definition 2.6) in graphs. We obtain that it is, in general, NP-hard to determine the existence of a non-separating cycle in a graph, even when restricting to subcubic graphs. However, if we require the graph to have a minimum degree of 3, we can then compute a non-separating cycle in linear time. These results will be put to use in the next section, where we consider a relaxation of the 3-decomposition conjecture and show that these weaker decompositions exist and can be computed efficiently.

We begin by showing that finding a non-separating cycle in a subcubic graph is hard.

(3.5) **3.4 Theorem.** It is NP-complete to determine whether a given graph G has a non-separating cycle. ◁

Proof. The problem is in NP since, given a cycle C, it is easy to determine whether $G - E\,C$ is connected. To obtain NP-hardness, we describe a reduction from 3-SAT

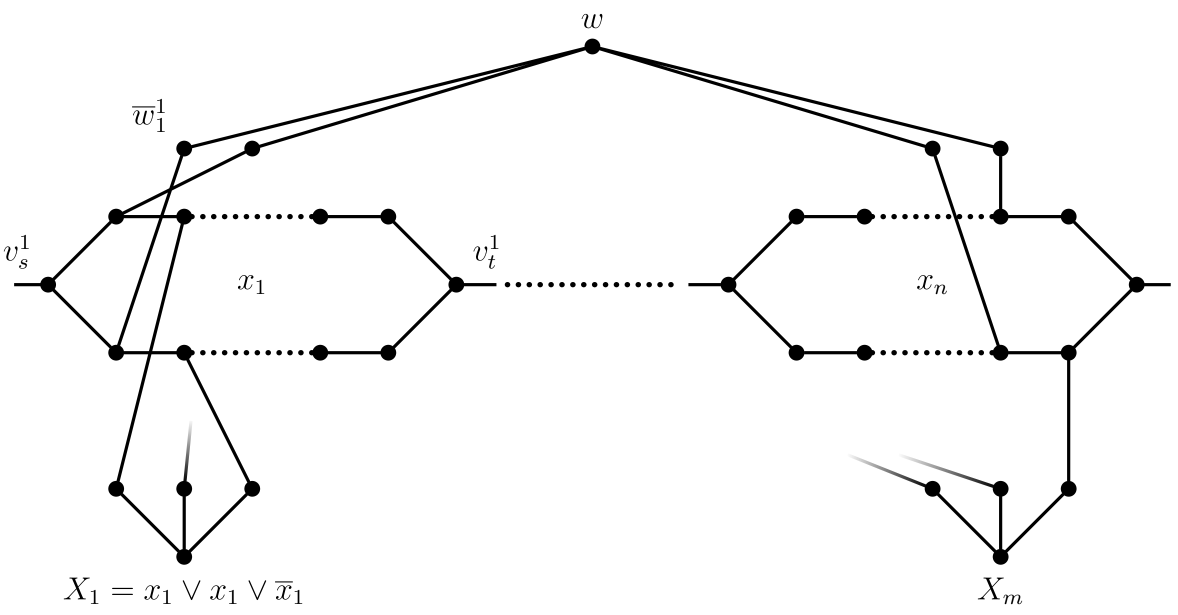


Figure 3.3.: The graph constructed for the reduction in Theorem 3.4. The clause gadgets (see Figure 3.4b) can be seen at the bottom whilst the variable gadgets (Figure 3.4a) are in the middle. Some exemplary connecting edges are drawn.

(Problem 2.15) similar to that used by Even, Itai, and Shamir [EIS76] to prove that the integer multi-commodity flow problem is NP-hard. To this end, let φ be a formula with variables $x_1, \dots, x_n$ and clauses $X_1, \dots, X_m$. Without loss of generality we assume that the first n clauses are of the form $x_i \vee x_i \vee \overline{x}_i$. We construct a graph G which is visualised in Figure 3.3.

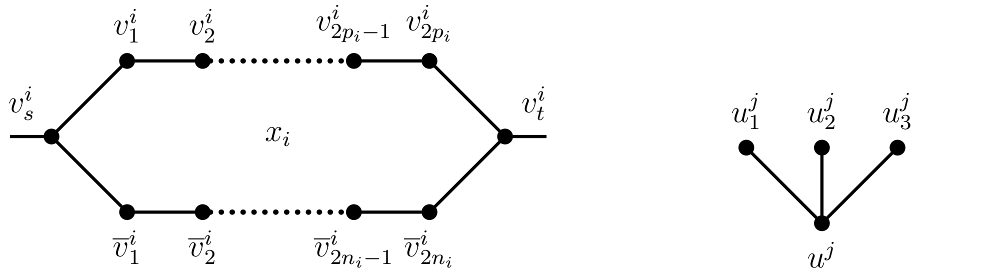
For the variable x_i, let p_i be the number of clauses containing it in a non-negated fashion and n_i the number containing $\overline{x}_i$. Then G has vertices v_s^i, v_t^i, $v_1^i, \dots, v_{2p_i}^i$, $\overline{v}_1^i, \dots, \overline{v}_{2n_i}^i$. These vertices are connected to form two paths

$$v_s^i v_1^i v_2^i \dots v_{2p_i-1}^i v_{2p_i}^i v_t^i \text{ and } v_s^i \overline{v}_1^i \overline{v}_2^i \dots \overline{v}_{2n_i-1}^i \overline{v}_{2n_i}^i v_t^i.$$

We call these parts *variable gadgets* and they are connected in series, that is, v_t^i is connected to v_s^{i+1} by an edge, for $i \in \{1, \dots, n-1\}$. This is illustrated in Figure 3.4a.

Next, we get further vertices u^j, u_1^j, u_2^j, u_3^j for every clause X_j. These are connected by edges $u^j u_l^j$ for $l \in \{1, 2, 3\}$, as seen in Figure 3.4b. We call these *clause gadgets* and they interconnect with the variable gadgets as follows. If the lth literal of X_j is x_i and this is the kth occurrence of x_i in φ, then the edge $u_l^j v_{2k}^i$ is part of the graph. Similarly, if the lth literal is $\overline{x}_i$ and this is its kth occurrence, then the edge $u_l^j \overline{v}_{2k}^i$ is added.

To complete the construction, we add a vertex w with a path of length 2 to all v_k^i where k is odd and call the intermediate vertices w_k^i. We do the same for the $\overline{v}_k^i$ with intermediate vertex $\overline{w}_k^i$. We also add an edge connecting v_s^1 to v_t^n.



(a) The ith variable gadget. (b) The jth clause gadget.

Figure 3.4.: The gadgets used in Theorem 3.4. For their interconnections in the complete graph, see Figure 3.3.

Claim. The constructed graph G has a non-separating cycle if and only if the formula φ is satisfiable. $\triangleleft$

Proof. We begin with the forward implication, in which we have a non-separating cycle C. We notice that C does not contain a vertex of degree 2 as any such cycle is separating. In particular, C only consists of edges that are in the variable gadgets or connect them. We see that C cannot be the unique cycle in x_i's gadget as this separates the gadget of X_i from the remaining graph. By symmetry, we can assume that C contains an edge in one of the paths of x_1's gadget, and, consequently, it contains an entire path. It cannot use any of the edges on the remaining one, as otherwise it would already be the cycle in x_1's gadget. The remaining cycle is, thus, a path from v_t^1 to v_s^1 using no edges in the first variable gadget. As a result, it contains all the edges connecting the variable gadgets and uses one of the two available paths in each. We now get a truth assignment T by setting $Tx_i \coloneqq$ `True` when the bottom path is used and $Tx_i \coloneqq$ `False` when the top one is. This assignment satisfies φ as, for any j, X_j's clause gadget is not separated from the graph, yielding a path from one u^j to w. But this path must be of the form $u^j u_l^j xy \ldots w$ where x and y are two neighbouring vertices in the variable gadget corresponding to the lth literal of X_j. Therefore, the cycle avoids the path corresponding to this literal, so X_j is satisfied by T.

For the converse, let φ by satisfiable with satisfying assignment T. We obtain a cycle C as in the forward implication by taking the top path through a variable gadget when the corresponding variable is set to `False` and the bottom one otherwise, together with all the edges between the variable gadgets. This cycle in non-separating. To see this we check that all vertices are connected to w. This is trivial for all vertices with an odd index in a variable gadget, as they have a direct path. The initial vertices v_s^i are covered this way as well. On the other hand, all the even vertices, and with them all the final vertices v_t^i, are connected to some vertex u^j, so we only need to ensure that these are connected to w as they also cover their neighbours. But as T was satisfying, the clause X_j has a satisfied literal, say the lth which is x_i, the case $\overline{x}_i$ is analogous. If this is its kth occurrence, then $u^j u_l^j v_{2k}^i v_{2k-1}^i w_{2k-1}^i w$ connects u^j to w as required and C is non-separating. $\diamond$

This completes the reduction. $\square$

3.5 Corollary. Deciding whether a graph has a non-separating cycle is already NP-complete in the case of subcubic graphs. ◁ [3.4]

Proof. Notice that the only vertex in the construction of Theorem 3.4 with a degree exceeding 3 is the vertex w. We modify the construction slightly by replacing it by a path of length $3m$ where each vertex receives one of the edges originally incident to w. The backwards implication is unaffected by this, as we remove none of the connecting edges and all these vertices are in the same component after removing the cycle anyway. For the forward implication we only need to check that the non-separating cycle still has the same form. But since all edges incident to the path cannot be part of a non-separating cycle, none of the edges on it can be either. So this adjustment does not harm the validity of the construction. □

With this part completed, we can finish the section by proving the second result we promised.

3.6 Theorem. Let G be a connected multigraph with $\delta G \geq 3$, then G has a non-separating cycle C and such a cycle can be computed in linear time. ◁ [2.7] [3.11] (3.17)

That such a cycle exists was shown by Thomassen and Toft [TT81, Corollary 2]. Note that the cited corollary is stated in a slightly different setting, where the vertices of the cycle are deleted instead of the edges. But the cycles are required to be induced, so it carries over to our case, thanks to the imposed degree condition: the only case where removing vertices instead of edges saves on components are when the cycle has degree 2 vertices.

The proof provided chooses an induced cycle maximising the order of some component. It is constructive and yields a quadratic algorithm, which we now sketch. The algorithm starts with some cycle C and fixes a component K^* of $H := G - C$. If the cycle is separating, it determines a new cycle C' by starting at a vertex in C that is in a component different from K^*. From there, it performs a breadth-first search in H. Under reasonable assumptions, like $\delta G \geq 3$, this either finds a path that ends at a vertex in C or a cycle completely contained in this other component. In either case, we can find a cycle C' such that $G - C'$ has a component $K' \supsetneq K^*$ in linear time. Since K^* can only grow at most n times before the cycle is non-separating, this runs in quadratic time.

The rest of this section is devoted to cutting down this runtime by removing unnecessary operations. Intuitively, the idea is to perform a single breadth-first search instead of restarting the computations for each subsequent cycle. Since this requires a rather precise formulation of the algorithm, we first present a mildly coarser variant for graphs, with the primary focus of proving correctness for it. We then flesh out the algorithmic details to show that all operations used can be implemented in a way that does not require more than linear time. Finally, we argue that we can handle the more general case of multigraphs using this algorithm as well.

Algorithm 3.1: An algorithm for computing a non-separating cycle.

Input: A connected graph G and a vertex $v^* \in G$ such that all vertices in $VG \setminus \{v^*\}$ have degree at least 3.

Output: A non-separating cycle C in G.

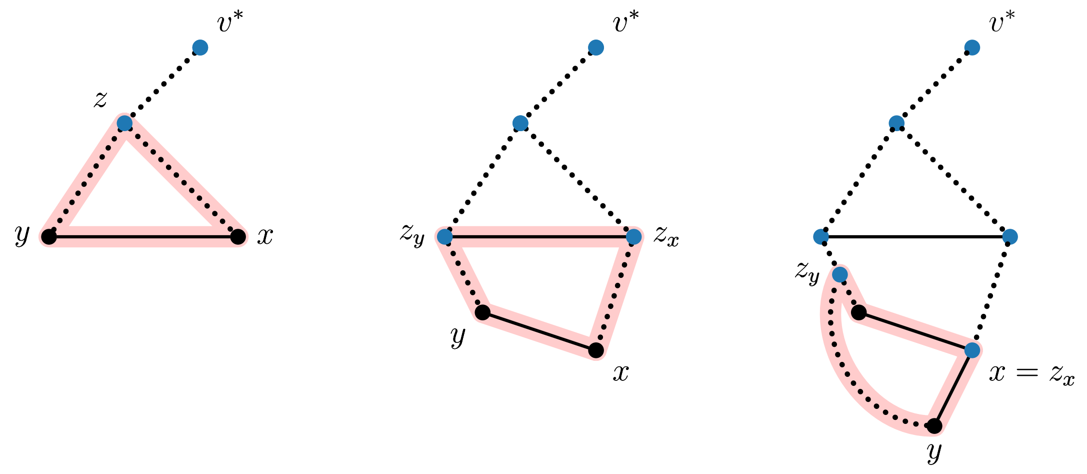
```
def NonSeparatingCycle($G$):
    for $v \in V$ do
        vis[$v$] $\leftarrow$ False, $\pi[v] \leftarrow$ None                # initialise BFS-variables
    $C \leftarrow \varnothing$, vis[$v^*$] $\leftarrow$ True, $Q \leftarrow [v^*]$
    while $Q \neq \varnothing$ do
        $x \leftarrow Q$.dequeue()                                  # explore the next vertex
        if $x \in K^*$ then continue                       # ignore vertices already in $K^*$
        for $y \in N_x$ do                        # not updated by calls to UpdateCycle
            if vis[$y$] = False then
                vis[$y$] $\leftarrow$ True, $\pi[y] \leftarrow x$, $Q$.append($y$)        # update BFS-variables
            else
                UpdateCycle()                    # update the cycle using cross-edges
                if $x \in K^*$ then break                       # stop on entering $K^*$
    return $C$

def UpdateCycle():
    Let $z_x$, $z_y$ be the first vertices of $P^\pi_x$, $P^\pi_y$ that are in $C$, or $v^*$ if none of them are
    if $P^\pi_x z_x \cap P^\pi_y z_y \neq \varnothing$ then
        Let $z$ be the first vertex these paths have in common
        Update $C \coloneqq xP^\pi_x z (P^\pi_y)^{-1} yx$
    else
        Update $C \coloneqq xP^\pi_x z_x C z_y (P^\pi_y)^{-1} yx$, where $z_x C z_y$ avoids a vertex of $K^*$
```

Let us start with the coarse variant which is shown in Algorithm 3.1. Note that in the algorithm and its discussion, we use the following notation:

- G is a connected graph, $v^* \in G$, and all vertices in $VG \setminus \{v^*\}$ have degree at least 3,
- $H \coloneqq G - E\,C$, where C is a (potentially empty) cycle in G,
- K^* is the component containing v^* in H if $C \neq \varnothing$ or $K^* \coloneqq \varnothing$ otherwise,
- the array vis keeps track of which vertices have been visited and the array π tracks the predecessors,
- $N_v \coloneqq N_H\, v \setminus \{\pi[v]\}$, and
- $P^\pi_v \coloneqq v\pi[v]\pi^2[v] \dots v^*$ is the predecessor-path from v to v^*.

We see that the algorithm mainly consists of a single breadth-first search in G, starting at v^*. Once a cross-edge is found, it creates the first cycle C. Afterwards, the breadth-first search disregards all vertices that are in K^* already. It thus explores the remaining components and uses cross-edges to create new cycles that enlarge the component K^*.

These cycles are obtained by following the predecessor-paths back until they either meet in a common vertex or reach C. In the first case we obtain a new cycle that is completely



(a) An update as in Line 19. (b) An update as in Line 21. (c) A second update at x.

Figure 3.5.: Cycles obtained by a call to UPDATECYCLE. Blue vertices indicate containment in K^*, dotted edges represent predecessor paths, and the cycles are highlighted.

contained in a component of H different from K^*. The second case yields two paths to vertices in C which can be augmented by one of two paths between them in C. Since some vertex in C is in K^*, we can choose a path that avoids this vertex and K^* grows.

We explicitly remark that, when exploring a vertex x, we ignore the edges at x that are in C, by definition of N_x. If UPDATECYCLE is called, this set may change, but the for loop continues to iterate over the original version.

To prove correctness, we shall start by convincing ourselves that the UPDATECYCLE function actually finds cycles.

3.7 Lemma. After each call to UPDATECYCLE, C is a cycle. ◁ (3.9)

(3.10)

Proof. We denote the data before the update by C', H', and $(K')^*$ and write C for the cycle after. First, we note that, if UPDATECYCLE was called for vertices x and y, then vis$[x]$ = True = vis$[y]$. We claim that $xy \notin C'$: this holds for the first call to UPDATECYCLE at x by the condition of the for loop. Moreover, a call to UPDATECYCLE satisfies that $(E\,C \setminus E\,C') \cap E\,x \subseteq \{xy, x\pi[x]\}$. Thus, for all edges considered later, the property still holds.

Hence, P_x^π and P_y^π are defined and we obtain z_x and z_y in Line 16. Let $P_1 := P_x^\pi z_x$ and $P_2 := P_y^\pi z_y$ and assume $P_1 \cap P_2 \neq \varnothing$, meaning the update in Line 19 is performed. This is exemplified is Figure 3.5a. Then $P := P_1 z P_2^{-1}$ is a path and $C = xPyx$ is a cycle unless $xy \in P$. But $xy \in P$ implies that $P = xy$ and thus $\pi[x] = y$ or $\pi[y] = x$. The first does not occur by Line 8 and the second one implies that the BFS-variables for y were updated at x, in which case UPDATECYCLE is not called. Therefore, C is a cycle.

We may now assume that $P_1 \cap P_2 = \varnothing$ as illustrated in Figure 3.5b. In particular, $C' \neq \varnothing$ since, otherwise, $z_x = v^* = z_y$ and the paths P_1 and P_2 are not disjoint. Also,

x, $y \notin (K')^*$, which holds for x by Lines 7 and 13 and for y since they are in the same component of H. Consequently, the paths P^π_x and P^π_y intersect C' and z_x, z_y are both in C'. This yields disjoint paths P_1, P_2 that live in H as well as two paths in C' between z_x and z_y. Since C' contains at least one vertex of $(K')^*$, we can, indeed, choose a path $z_xC'z_y$ such that it avoids an element of $(K')^*$ and $P := P_1z_xC'z_yP_2^{-1}$ is a path. The cycle update in Line 21 sets C to $xPyx$, which is a cycle unless $xy \in P$. But again this can only occur if $P = xy$, which implies that $z_xC'z_y$ is empty since $xy \notin C'$. Hence, we have $z_x = z_y$, contradicting the assumption $P_1 \cap P_2 = \varnothing$. □

Next we show several useful invariants of the for loop.

(3.9) (3.11) **3.8 Lemma.** After every iteration of the for loop in Line 8 the following properties hold:

(a) $(K')^* \subseteq K^*$,
(b) $E\,C' \setminus E\,C \subseteq E\,K^*$,
(c) $v \in C \Rightarrow v \in K^*$ or $v\pi[v] \in C$, and
(d) $v \notin K^* \Rightarrow N_v \neq \varnothing$,

where C' be the cycle before this iteration of the loop, $H' := G - E\,C'$, and $(K')^*$ be the component of H' containing v^*. If UpdateCycle is called at a vertex x in this iteration, we also obtain that

(e) $x \in C' \Rightarrow x \in K^*$. In particular, UpdateCycle is called at most twice at any vertex. ◁

Proof. We prove Properties (a) to (c) together. All three of them are true initially (after the 0th iteration), where $C' = C = \varnothing$, and they continue to hold after an iteration in which UpdateCycle is not called since $C' = C$ in this case. So we may assume that UpdateCycle is called on vertices x and y in this iteration.

If C is set to $xP^\pi_x z(P^\pi_y)^{-1}yx$ in Line 19, then

$$E\,C = \{xy\} \cup \left\{v\pi[v] \colon\ v \in P^\pi_x \mathring{z} \cup P^\pi_y \mathring{z}\right\}.$$

Since $x \notin (K')^*$, we have $E\,C \cap E((K')^*) = \varnothing$ and $(K')^* \subseteq K^*$, proving Property (a). Additionally, unless C' is empty, it contains a vertex of $(K')^*$ and $E\,C' \cap E\,C = \varnothing$. Hence, $C' \subseteq K^*$, showing Property (b). Lastly, for all $v \in C - z$, we have that $v\pi[v] \in C$ and $z \in K^*$ since P^π_z is a zv^*-path in H, proving Property (c).

If C is set to $xP^\pi_x z_xC'z_y(P^\pi_y)^{-1}yx$ in Line 21, then

$$E\,C = \{xy\} \cup \left\{v\pi[v] \colon\ v \in P^\pi_x \mathring{z}_x \cup P^\pi_y \mathring{z}_y\right\} \cup E(z_xC'z_y).$$

Here $C' \neq \varnothing$ and $(E\,C \setminus E\,C') \cap E((K')^*) = \varnothing$, showing $(K')^* \subseteq K^*$ and Property (a). Moreover, $E\,C' \setminus E\,C = E(C' - \mathring{z}_xC'\mathring{z}_y)$ which is completely contained in K^* since

$z_x C' z_y$ avoids an element of $(K')^*$, proving Property (b). Finally, for all vertices v in $C - V(z_x C' z_y)$, we have $v\pi[v] \in C$. For the vertices $v \in \mathring{z}_x C' \mathring{z}_y$ the edge $v\pi[v]$ is in C if and only if it is in C'. Since we may assume that Property (c) holds for C' and $(K')^*$, we either get $v\pi[v] \in C$ or $v \in (K')^* \subseteq K^*$. The vertices z_x and z_y are in K^* by Property (b) since they are ends of edges in $E\,C' \setminus E\,C$. Thus, Property (c) holds as well.

We use this to quickly prove Property (e): note that $x \in C'$ implies that $z_x = x$ as illustrated in Figure 3.5c. This gives us that $x = P^\pi_x z_x$ and $P^\pi_y z_y$ are disjoint and the cycle is updated as in the paragraph above. Therefore we know that $x = z_x \in K^*$. This also shows that UpdateCycle can be called at most twice at x: after the first call, x is either in K^* or on the cycle, and the latter places it in K^* after the next call, ending the iteration at x.

Property (d) follows from Property (c) and the requirement on the minimum degree: let $v \notin K^*$. If K^* is empty, then $H = G$ and $|N_v| = |N_G\, v| \geq 1$ since G is connected. Otherwise, $v \neq v^*$ and $N_G\, v$ contains at least three vertices. If $v \notin C'$, then N_v contains at least two vertices and, otherwise, two of the edges incident with v are removed in the transition to H, so $N_H\, v$ contains at least one vertex. Moreover, since $v \notin K^*$, the edge $v\pi[v]$ is in C by Property (c) and $N_H\, v = N_H\, v \setminus \{\pi[v]\} = N_v$. □

Now that we have gathered these properties, let us prove correctness of the algorithm. We know that C is a cycle after each call to UpdateCycle by Lemma 3.7, which ensures that the result returned by the algorithm is, indeed, a cycle. This is true since UpdateCycle must be called at least once because cross-edges exist due to the requirement on the minimum degree. To see that this cycle is non-separating, we prove the following invariant of the while loop.

3.9 Theorem. After every iteration of the while loop in Line 5 every component $K \neq K^*$ of H contains a vertex in Q. ◁ [3.7] [3.8] (3.10)

Proof. To see that the property holds, we note that it is true initially, where there is just one component and $v^* \in Q$. We can now assume it was true before the current iteration and need to show that it remains true after. To this end, let C, H, K^*, and Q be the corresponding variables after this iteration.

We prove three claims.

Claim 1. If x has an unvisited neighbour y at the start of the iteration, then x and y are in the same component of H and x is in K^* or $y \in Q$. ◁

Proof. We have already seen, in the proof of Lemma 3.7, that a call to UpdateCycle at x and y' adds the edge xy' to the cycle. Since y is not visited, UpdateCycle is not called at y and $xy \notin C$. In particular, x and y are in the same component of H. Additionally, $y \neq \pi[x]$, so $y \in N_x$. Thus, y is considered in the for loop in Line 8 and added to Q unless x is in or enters the component containing v^*. In either of the latter cases, the iteration terminates with x in K^* by Property (a). ◇

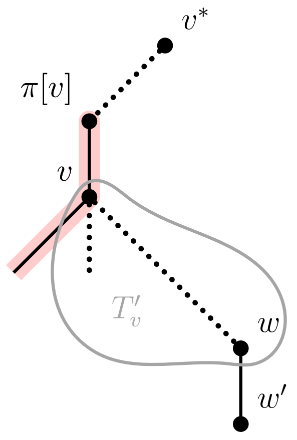


Figure 3.6.: The BFS-tree in the proof of Claim 3. Dotted edges represent predecessor paths in T', and a part of the cycle C' is highlighted. The edge ww' is not part of T'.

Claim 2. If x has no unvisited neighbour at the start of the iteration, then $x \in K^*$. ◁

Proof. If x has no unvisited neighbours, then $x \neq v^*$ and UpdateCycle is called on any vertex considered by the for loop. This occurs for all vertices in the set N specified in Line 8 unless x enters the same component as v^*, putting it in K^*. But the set N has at least one element by Property (d) (of Lemma 3.8) and it has at least two if x is not on the cycle at the start of the iteration. In the latter case, a call to UpdateCycle either places x in K^* or ensures that it lies on the cycle, unifying the two cases. A call to UpdateCycle now places $x \in K^*$ by Property (e). ◇

Claim 3. Let C' be the cycle after an iteration of the for loop, $H' := G - E\,C'$, and let $(K')^*$ the component of H' containing v^*. Then any component $K' \neq (K')^*$ of H' with $x \notin K'$ contains a vertex in Q. ◁

Proof. Let K' be a component of H', $K' \neq (K')^*$, and $x \notin K'$, then K' contains a vertex $v \in C'$. Because $K' \neq (K')^*$, Property (c) implies that $v\pi[v] \in C'$. Let T' be the BFS-tree of the current iteration, that is, T' contains all arcs $\pi[v']v'$ for vertices v' with $\pi[v'] \neq$ `None`. This can be regarded as a tree rooted at v^*. (We refer to [Cor+22] for more information on BFS-trees.) Let T'_v be the subtree of T' rooted at v which contains all vertices reachable from v in $T' - E\,C'$ and is illustrated in Figure 3.6. We show that all leaves of T'_v, which are all in K' as well, are in Q, proving the claim.

Let w be such a leaf and suppose it is not in Q. Since $w \neq x$ by choice of K', w was removed from the queue in a prior iteration of the while loop. Let C^w be the cycle from the start of this prior iteration, $H^w := G - E\,C^w$, and let $(K^w)^*$ be the component of H^w containing v^*. Moreover, let $N^w := N_{H^w}\, w \setminus \{\pi[w]\}$ and $N' := N_{H'}\, w \setminus \{\pi[w]\}$.

We know that $w \notin (K')^*$ since v is not, which implies that $N' \neq \varnothing$ by Property (d). We take a vertex $w' \in N'$ and claim that $w' \in N^w$. Because $ww' \notin C'$, $ww' \notin C^w$ since it would be removed in a call to UpdateCycle otherwise, which implies that $w \in (K')^*$

by Properties (a) and (b), yielding a contradiction. By $w' \in N'$ we know $w' \neq \pi[w]$ and, as $ww' \in H^w$, we get $w' \in N^w$.

Furthermore, when w is considered, it is not in $(K^w)^*$ since $w \notin (K')^*$. Thus, w' is considered in the iteration where w is removed from the queue. Since w is a leaf of T'_v and ww' is in H' but not in T'_v, we get that $\pi[w'] \neq w$. In particular, UpdateCycle is called on w and w'. Consequently, ww' enters the cycle and, since it is not in C', Properties (a) and (b) imply again that $w \in K^*$, yielding a contradiction once more. ⋄

To see that these claims imply the theorem, note that any component $K \neq K^*$ of H contains x or a vertex of Q by induction on the iterations of the for loop using Claim 3. For the component K containing x we have $K = K^*$ if x has no unvisited neighbour by Claim 2 and, otherwise, Claim 1 yields $K = K^*$ or K contains a vertex of Q. □

3.10 Corollary. Algorithm 3.1 is correct. ◁ [3.7] [3.9] (3.11)

Proof. Correctness follows from Lemma 3.7 and Theorem 3.9: the first guarantees that the algorithm returns a cycle while the second ensures that H is connected on termination, since Q is empty. □

We now get to the promised implementation details. First, note that most of Algorithm 3.1 already runs in linear time: the initialisation, the vertex exploration, and the BFS-variable updates are all unproblematic. What we need to efficiently implement are the calls to UpdateCycle as well as the checking of whether or not a vertex x is in K^*, see Lines 7 and 13.

We can easily combine these problems as follows: we create an array K, which has an entry for every vertex, is initialised to `False`, and maintains the invariant $\mathsf{K}[v] =$ `True` if and only if $v \in K^*$. Therefore, we can replace both conditions by $\mathsf{K}[v] =$ `True`, which can be checked in constant time. Since the entries in K only change after a cycle update, we can move the maintenance of this array to UpdateCycle as well. Hence, it suffices to show that we can implement UpdateCycle in such a way that all calls to it combined take linear time.

For the implementation of UpdateCycle we need to discuss how we represent the cycle. It also comes with an array C, which has an entry specifying containment in C for every vertex and is initialised with `False` since no cycle is available initially. Additionally, we store the cycle as a circular doubly linked list, which is what we return. (This is just a doubly linked list in which the first and last element point to one another.) The 'first' element in this list, to which we store a pointer, is ensured to be in K^* and we denote it by z^*. Finally, we track the distances to v^* obtained by the BFS in NonSeparatingCycle, using an array d.

If UpdateCycle is called on x and y, we know that $|\mathsf{d}[x] - \mathsf{d}[y]| \leq 1$ by the properties of BFS. We set $x' \leftarrow x$ and $y' \leftarrow y$ and then go over to predecessors by setting $x' \leftarrow \pi[x']$

and $y' \leftarrow \pi[y']$. We do this at most once for either x' or y' to ensure they have the same distance and then we continue updating both simultaneously, such that we always consider vertices with the same distance. This continues until either

- $x' = y'$ or
- $x' \in C$ or $y' \in C$.

In the second case, we continue transitioning at the remaining vertex, until it, too, ends in C. This yields the paths $P_x^\pi z_x$ and $P_y^\pi z_y$ used in Line 21 whereas the first case finds the paths $P_x^\pi z$ and $P_y^\pi z$ used in Line 19.

Next, we remove those vertices from the cycle that leave it (this step is skipped when C is empty). If we update as in Line 19, then we set $\mathsf{C}[v] \leftarrow$ `False` for all vertices different from z on the current cycle and delete the linked list. For the update in Line 21, we start at z^* and follow the cycle in both directions until z_x and z_y are found. Vertices along the way are removed from C and their entries in the linked list are deleted. This leaves us with a doubly linked list representing the path $z_x C z_y$ that avoids the vertex $z^* \in K^*$. In both cases we can now create a circular doubly linked list representing the new cycle and update C accordingly. As the first vertex we choose z or z_x, depending on the case, both of which are in K^* by Properties (b) and (c) (of Lemma 3.8).

We still need to update K, which will involve a second BFS. When the first cycle is found, we start a BFS at v^* in H. Afterwards, the only additional edges that can be explored are at vertices that were previously in $K^* \cap C$ since these are the only ones with newly accessible edges. But when we explore the cycle to update the array C, we can easily add all vertices v with $\mathsf{K}[v] =$ `True` to the queue again and continue exploring from there. Since a vertex is only added again if it leaves C (to K^*), we get that any vertex is readded at most once by Property (a). Hence, this update runs in linear time since each neighbourhood is explored at most twice.

This just leaves the cycle updates, whose running time we need to determine. The removal of vertices from C and the deletion of their list entries is unproblematic, since it only happens once per vertex as we just argued. However, at any vertex, we only transition to its predecessor in at most one call to UpdateCycle since such an update places it in the cycle and stops subsequent paths at it. Thus, we only have a linear number of transitions to predecessors amongst all cycle updates. But all other operations (adding the edge xy, connecting to the path $z_x C z_y$) only require constant time and there are at most a linear number of calls to UpdateCycle. Hence, the runtime is linear and we obtain the theorem below.

[3.8] [3.10] (3.6) **3.11 Theorem.** Let G be a connected graph and $v^* \in VG$ such that all vertices in $VG \setminus \{v^*\}$ have degree at least 3. Then G contains a non-separating cycle and such a cycle can be found in linear time. ◁

For the proof of Theorem 3.6 we still need to deal with multigraphs G that have loops or parallels. To this end, we first note that, without loss of generality, G has no loops, since any loop is a non-separating cycle. Similarly, we can assume there is no pair of vertices with three or more edges between them. To deal with the remaining parallels, notice that two edges between vertices u and v yield a non-separating cycle unless u and v are cut-vertices. Consequently, we can begin by computing the blocks of G, which can be done in linear time by Theorem 2.7. If the graph is 2-connected, we are done. Otherwise, we take a leaf block and obtain a 2-connected subgraph of G in which all vertices other than the cut vertex v^* have degree at least 3. If it contains parallels, we have found a non-separating cycle and otherwise it is simple and we can apply Theorem 3.11.

3.4. A relaxation of the conjecture

In this section, we study a relaxation of the conjecture where we allow paths of length up to 2 instead of restricting to a matching. We call these decompositions *long 3-decompositions* and they are guaranteed to exist.

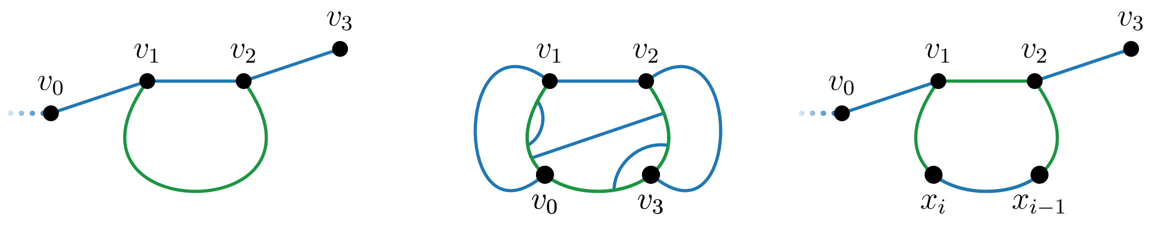
3.12 Theorem. Every connected cubic graph has a long 3-decomposition. ◁ [LC14] [LM19]

This is not a new result, it was already shown by Li and Cui [LC14] and, in a more general form, by Lyngsie and Merker [LM19]. (If we were to classify these papers according to the proof techniques presented in Section 3.2, the first would fall into the manipulation and the second into the reduction category.) Since long 3-decompositions exist, we are also interested in how they can be computed.

We first argue that the proof presented in [LM19] does not directly provide us with an algorithm. Next, we present another proof for the existence of long 3-decompositions based on a potential function argument and use it to obtain an algorithm. Finally, we note that the proof presented in [LC14] is incomplete, present a small example where it goes wrong, and show how to avoid this case. Moreover, this proof is based on depth-first search and, together with Theorem 3.6, can be turned into a linear time algorithm.

As announced, we begin by looking at the proof in [LM19], in which it is actually shown that it is possible to decompose any connected, not necessarily cubic, graph into a spanning tree, an even graph, and a star forest. The last part is a forest in which each component is a star. If we look at cubic graphs, which have a maximum degree of 2 after removing a spanning tree, the second component is a disjoint union of cycles and the last part can only contain paths of length at most 2, giving us a long 3-decomposition.

The proof reduces to the case that all cycles in the graph G are separating (since any non-separating cycle can be put into the even graph), and then shows properties that hold for any minimal counterexample. These are all shown by finding a way to reduce the problem to a smaller graph. Once sufficiently many properties have been obtained, a contradiction is derived. As such, the proof seems constructive, since it provides operations that can be performed in any step to reach a smaller graph. The problem is the initial assumption that all cycles of the graph are separating, which we would need



(a) The paths P and Q. (b) The situation with no x_i. (c) After the transition to T'.

Figure 3.7.: The reduction used in the potential function proof for Theorem 3.12. Here, green edges are part of the tree and the remaining ones are blue. Note that, on the left, the endpoint v_0 need not be disjoint from the cycle $Q + v_1v_2$, and neither does v_3 if it is also an endpoint.

to guarantee algorithmically. However, by Corollary 3.5 this is a hard problem, even in the subcubic case (which we reach after the first cycle removal).

Next, we present a proof that uses a potential to obtain this result.

Proof (of Theorem 3.12 using a potential function argument). Let G be a connected cubic graph and let H be a spanning subgraph of G. Then $G - E\,H$ is a disjoint union of paths and cycles. Let $\mathcal{P}\,H$ be the set of paths of length at least 2 in $G - E\,H$ and define

$$\varphi\,H := \sum_{P \in \mathcal{P}\,H} \|P\|^2.$$

Let T be a spanning tree in G that minimises $\varphi\,T$ amongst all spanning trees. We claim that, for this T, $G - E\,T$ consists only of paths of length at most 2.

Suppose not, and let $P := v_0 \dots v_k$ be a longest path in $G - E\,T$ of length $k \geq 3$. Let $Q := v_2Tv_1 = x_1x_2 \dots x_t$ be the unique path in T from v_2 to v_1. This situation is visualised in Figure 3.7a.

We prove that Q contains a vertex x_i that is not incident to any non-tree (blue) edge. Suppose this were not the case. The two vertices v_1 and v_2 have degree 1 in T and 2 in $G - T$, while the remaining vertices of Q have degree at least 2 in T. But now, all these are incident to a non-tree edge as well, meaning they have degree exactly 2 in T. As a result, Q is a component of T and, as T is a spanning tree, $Q = T$ and $V\,G = V\,Q$. Hence, P has length exactly 3 and both endpoints are on the cycle. The form of G in this case is illustrated in Figure 3.7b and we note that $E\,G \setminus (E\,Q \cup \{v_1v_2\})$ is a matching. We now show how to obtain a 3-decomposition in this case, which contradicts the minimality of $\varphi\,T > 0$ as the tree of a 3-decomposition has potential 0.

To this end, we choose a non-tree edge $e = x_ix_j$, with $i < j$ such that $\ell\,e := j - i$ is minimal. Note that $e \neq v_2v_1 = x_1x_t$, as it attains the maximal value $\ell(x_1x_t) = t - 1$, so at least v_2v_3 is preferred. Therefore, we may consider the cycle $C := x_iQx_j + e$ and the path $P^* := x_ix_{i-1} \dots v_2v_1x_{t-1} \dots x_j$ in $G - E\,C$ that connects all vertices except those in the set $\{x_{i+1}, \dots, x_{j-1}\}$. But any vertex not connected to P^* yet has an incident edge that is not in C. Its other endpoint is on P^*, since it cannot also be on C by the minimality of $\ell\, e$. By adding all these edges to P^*, we obtain a spanning tree T^* that yields a 3-decomposition as desired. To see that this is indeed the case, note that $G - E\, T^*$ consists of the cycle C and edges in the matching $E\, G \setminus (E\, Q \cup \{v_1 v_2\})$.

Consequently, we now know we have a vertex $x_i \in Q$ whose incident edges are all in T. We define T' as the tree $T + v_1 v_2 - x_{i-1} x_i$, which is visualised in Figure 3.7c. To complete the proof, we show that $\varphi\, T' < \varphi\, T$, giving us our desired contradiction. To this end, we first consider $\varphi\, G'$ where $G' \coloneqq T + v_1 v_2$. By adding the edge $v_1 v_2$ to T, the path P is split into the path $v_2 \dots v_k$ and the edge $v_0 v_1$. This means that we lose a path of length k and obtain one of length $k-2$ and one of length 1. Hence,

$$\varphi\, G' \leq \varphi\, T - k^2 + (k-2)^2 = \varphi\, T - 4k + 4,$$

where the first inequality is only strict when $k = 3$, in which case the $(k-2)^2$ would not be added. Due to the removal of $x_{i-1} x_i$ from T, we obtain an additional edge in the set of paths. By construction, the degree of x_i in $G - T'$ is one, making it the end of a path. It can be a matching edge, which does not increase the value of φ at all, or it lengthens a path in $G - G'$ of length k' by one edge. If $k' = 1$, we obtain that

$$\varphi\, T' = \varphi\, G' + 4 \leq \varphi\, T - 4k + 8 < \varphi\, T$$

and for $k' \geq 2$ we get

$$\begin{aligned} \varphi\, T' &= \varphi\, G' - (k')^2 + (k'+1)^2 \\ &\leq \varphi\, T - 4k + 4 + 2k' + 1 \\ &\leq \varphi\, T - 2k + 5 \\ &< \varphi\, T. \end{aligned}$$

In all three cases we have obtained $\varphi\, T' < \varphi\, T$ and the proof is complete. □

Let us note a couple of things. First, we observe the existence of x_i can be obtained faster using that the path Q together with the edge $v_1 v_2$ forms a Hamiltonian cycle and the 3-decomposition conjecture is known to hold for these graphs. This would have provided us with the tree whose potential is 0. What we did instead is, in essence, a repetition of the proof we could have used. The purpose of this exercise was to see how the decomposition is obtained since this lets us turn the proof into an algorithm.

3.13 Corollary. For a connected cubic graph we can compute a decomposition into a spanning tree, a 2-regular subgraph, and vertex-disjoint paths of length at most 2 in $\mathcal{O}(n^3)$ time. ◁

Proof. Notice that the potential proof above is constructive in the sense that it gives an explicit transformation that reduces the value of $\varphi\, T$ in the case that $G - E\, T$ has paths of length at least 3. It can thus be used to construct a decomposition of a graph into

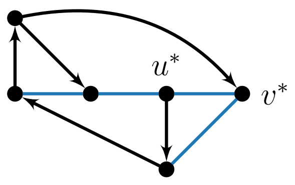


Figure 3.8.: An example of a depth-first search tree that leaves a path of length 4. The black arcs are part of the DFS-tree, which has u^* as its root while the remaining blue edges form a path of length 4.

a spanning tree and paths of length at most 2. As G is cubic, it has $\frac{3n}{2}$ edges, leaving only $\frac{n}{2}+1$ for $G - E\,T$. Therefore,

$$\varphi T = \sum_{P \in \mathcal{P}\,T} \|P\|^2 \leq \left(\sum_{P \in \mathcal{P}\,T} \|P\|\right)^2 \leq \left(\frac{n}{2}+1\right)^2 \in \mathcal{O}\left(n^2\right).$$

So, by starting with some spanning tree T, it takes at most $\mathcal{O}(n^2)$ transformation steps to find a spanning tree T' such that $G - E\,T'$ has no path of length 3 or more or we run into the special case, where we can compute a 3-decomposition.

Computing a spanning tree using depth-first-search only takes $\mathcal{O}(n)$ time, so it suffices to show that we can implement the transformation steps or find the 3-decomposition in linear time. We start with the former: finding the longest path P in $G - E\,T$ and the corresponding path Q in T from the proof can be done in $\mathcal{O}(n)$ time. Finding the vertex x_i and swapping the edge is also unproblematic.

This just leaves the computation of the 3-decomposition, which occurs when no vertex x_i is found. Here, checking the distance between the endpoints of all edges in $G - E\,T$, removing the cycle, and updating the tree are all operations that can be completed in linear time. This yields an $\mathcal{O}(n^3)$ time algorithm. □

To round out this section, we look at [LC14]. The authors obtain the desired long 3-decomposition as a direct consequence of the following result.

3.14 Theorem ([LC14]). *Let G be a connected subcubic graph. Then G contains a spanning tree T such that every component of $G - E\,T$ is a path of length at most 3. Moreover, the number of paths of length 3 is at most one.* ◁

Their proof claims that the spanning tree obtained by depth-first search does the trick but the example in Figure 3.8 illustrates that this is not the case in general.

However, this example already illustrates the worst case that can occur: there is at most one path whose length exceeds 2 and this path has length at most 4. This result is implied by the following lemma.

(3.16) **3.15 Lemma.** Let G be a connected subcubic multigraph and T be a spanning tree in G computed by depth-first search starting at $u^* \in G$. If u and v are distinct vertices of degree 2 in $G - E\,T$ and $uv \in E\,G$, then without loss of generality $u = u^*$. ◁

Proof. Let T, u^*, u, and v be as in the statement and assume without loss of generality that u is visited before v by the depth-first search. We regard T, like in Figure 3.8, as a directed graph with an edge xy directed from x to y if y is visited from x by the depth-first search. Since u has an unvisited neighbour, namely v, it has out-degree at least 1. Moreover, u has degree 2 in $G - ET$, giving it degree 1 in T. Therefore, u has in-degree 0 and $u = u^*$. □

This limits the longest path length to 4 since any edge between degree 2 vertices must have u^* as an end, meaning there are at most two such edges in all of G and they lie in the same component of $G - ET$. As a result, if we replace the 3 in Theorem 3.14 by a 4, the claim is now proved (even without needing the restriction to simple graphs, only loops need to be prohibited). Furthermore, if G contains a vertex of degree at most 2, then starting the DFS at this vertex guarantees that no path of length 3 exists: in this case, we can choose such a vertex as u^* and get that u^* has degree at most 1 in $G - ET$ and by Lemma 3.15 no adjacent degree 2 vertices exist.

We get the complete result of Theorem 3.14 by modifying the depth-first search slightly, to exclude the case that a length 4 path exists. In fact, we can actually show the result for all graphs without loops (which necessarily leave cycles in $G - ET$). To achieve this, we first choose a vertex $u^* \in G$, giving preference to one that has degree at most 2 or that is adjacent to parallels if the graph is not simple. Then we run depth-first search starting at u^* and have it prioritise neighbours of u^* in case it has a choice of successors to visit next. We call this algorithm *modified DFS*.

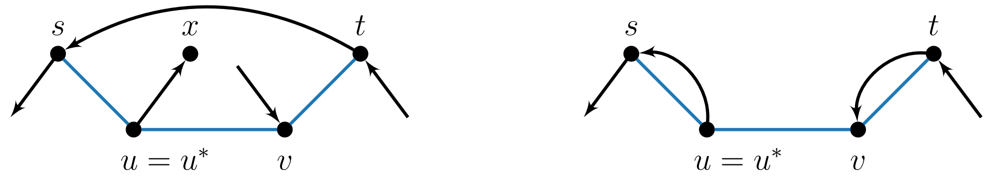
3.16 Theorem. Let G be a connected subcubic multigraph without loops and T be a spanning tree in G computed by modified DFS. Then every component of $G - ET$ is a path of length at most 3. Moreover, the number of paths of length 3 is at most one. ◁ [3.15]

Proof. Let u^* be the vertex chosen by modified DFS. If u^* has degree at most 2, we are done by the previous arguments. Thus, we may assume that u^*, and therefore all vertices of G, have degree 3.

Furthermore, let u and v be two adjacent vertices of degree 2 in $H := G - ET$ with $P := suvt \subseteq H$. By Lemma 3.15 we may assume that $u = u^*$. We now prove that P is a component of H. This implies the theorem as it shows that adjacent degree 2 vertices live in a path of length at most 3 that contains u^*, making this path unique.

We refer to Figure 3.9 for an illustration of the vertices considered in the following. Let $Nu := \{s, v, x\}$. When v is visited by the modified DFS, vt is not chosen, so t must have been visited prior to v. On the other hand, when t is visited, tv is not chosen either, so some edge ty must have been used, for an unvisited vertex y. Since v is preferred over any non-neighbour of u and x has been visited, we can conclude that $y \in Nu \setminus \{x\}$.

We start with the case that $y = s$, which is illustrated in Figure 3.9a. Note that this is mandatory if G is simple. By the fact that $tv \notin T$, we also conclude that v is visited



(a) The case that $ts \in G$. (b) The situation in multigraphs.

Figure 3.9.: The behaviour of modified DFS. As before, black arcs are part of the DFS-tree, which is rooted at u, while blue ones are not.

before we backtrack at s, in particular, modified DFS finds an unvisited successor at s. This makes P a component, since both s and t have degree 1.

This leaves the case $y = v$ shown in Figure 3.9b. Here we also have that the degree of t in H is 1, but we still need to consider s. However, parallels exist in G in this case, and by choice of $u^* = u$, this vertex must be incident to a pair of parallels. Since a parallel to uv is not an option for degree reasons, we get a parallel to the edge us, which is also the first edge traversed by the modified DFS. But, once more, we must visit v before backtracking at s, so modified DFS finds an unvisited successor and $d_H\, s = 1$. □

We can obtain the existence of long 3-decompositions for all cubic graphs (not just the simple ones) by the same argument as in [LC14]. Let G be a cubic graph. By Theorem 3.6 we can find a non-separating cycle C in G in linear time. The graph $G - E\,C$ has a vertex of degree 1, so choosing it as u^* and running DFS (modified or otherwise), yields a long 3-decomposition. This shows the theorem below since, for a long 3-decomposition, the now allowed loops pose no threat.

[3.6] **3.17 Theorem.** Every connected cubic multigraph has a long 3-decomposition and such a decomposition can be computed in $\mathcal{O}(n)$ time. ◁

Chapter 4.

Star-Like Graphs

In this chapter, we consider a natural extension of Hamiltonian graphs: removing a Hamiltonian cycle from a cubic graph leaves a perfect matching. Conversely, removing a perfect matching M from a cubic graph G leaves a disjoint union of cycles. Contracting these cycles yields a new graph G_M. The graph G is star-like if G_M is a star for some perfect matching M, making Hamiltonian graphs star-like. We extend the technique used to prove that Hamiltonian graphs satisfy the 3-decomposition conjecture to show that 3-connected star-like graphs satisfy it as well.

Here we look at graphs that are a natural extension of Hamiltonian cubic graphs in the context of the 3-decomposition conjecture. Notice that a cubic graph G with a Hamiltonian cycle C has a perfect matching, namely the edges of $G - EC$. In general, for a cubic graph G with a perfect matching M, $G - M$ is the disjoint union of cycles, leading us to the following definition.

4.1 Definition. Let G be a connected cubic graph with a perfect matching M. Then $G - M$ consists of disjoint cycles and contracting these in G to single vertices yields a new graph G_M, the *contraction graph*, that has a vertex for every cycle in $G - M$ and an edge between two vertices if the corresponding cycles are connected by an edge of M. If G has a perfect matching M such that G_M is a star, then G is *star-like*. $\triangleleft$

We wish to make a few remarks on this definition, which is illustrated in Figure 4.1 by an example. First note that all Hamiltonian cubic graphs are star-like. Furthermore, by Petersen's theorem [Pet91], all bridgeless cubic graphs have a perfect matching. Also, using this definition, the main theorem in [XZZ20] now reads as: the 3-decomposition conjecture holds for any connected cubic graph with a perfect matching such that its contraction graph has order 3. We observe that a graph has a contraction graph of order 1 if and only if it is Hamiltonian. Moreover, graphs with a contraction graph of order 2 are traceable, so, in fact, we can say that the 3-decomposition conjecture holds for all graphs with a contraction graph of order at most 3.

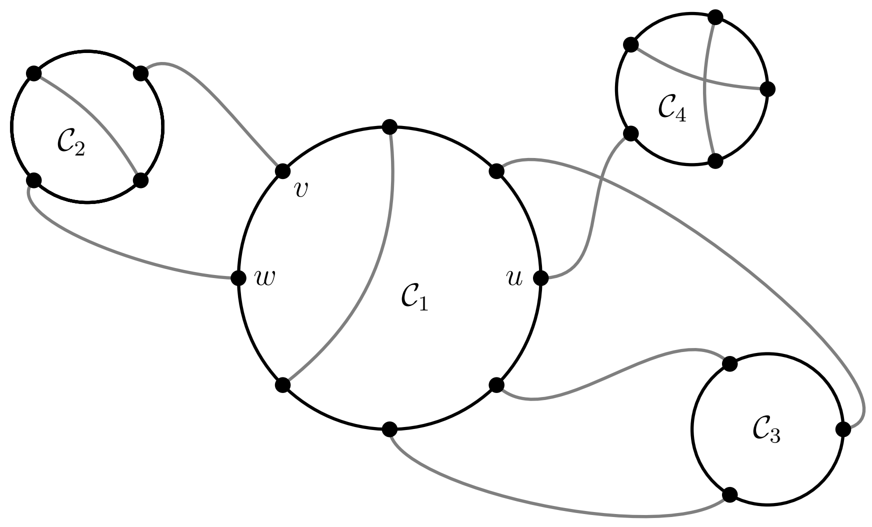


Figure 4.1.: An example of a star-like graph. The edges of M are grey, while the edges of $G - M$ are black.

Our goal here is to show that the conjecture holds for all star-like graphs, which restricts how the cycles of G_M may be arranged, but does not limit their number. More precisely, we shall prove the following theorem:

[4.8] [4.12] [4.13] [4.14] [4.15] [4.18] **4.2 Theorem.** Every 3-connected star-like graph has a 3-decomposition. ◁

The idea of this proof is to construct a tree on the vertices of the centre cycle and to iteratively extend it to the tips of the star. Once extended to all cycles it yields a 3-decomposition. To make this precise, we introduce two types of decompositions in Section 4.1. One describes this tree and the other formalises the properties that we need to extend it to further cycles. To achieve this extension, we show that certain decompositions always exist in Section 4.2 and that these are actually sufficient in Section 4.3. As such it falls into the first proof type in our classification from Section 3.2, since we manipulate a global structure.

We note that the result in [XZZ20] is similar to ours, both in their claim as in the approach, though the final result differs. However, our approach has two main advantages: due to the definition of the decompositions we provide, it is easier to obtain reusable components (whereas the proof in [XZZ20] consists of a series of very long case distinctions in which a lot of details have been omitted). In contrast to this, our final proof is very short and a direct application of the components we proved before.

Additionally, these decompositions have straightforward extensions that allow them to deal with more general contraction graphs and our results carry over to this more general form. This leaves hope that the decompositions we introduce here could be a helpful tool for proving the conjecture for larger classes of graphs, tree-like being a natural candidate. We go into a bit more detail on this in Section 4.4.

Finally, in the Section 4.5, we exhibit an infinite family of graphs that are 3-connected and star-like, and therefore covered by Theorem 4.2, but for which the conjecture had not yet been proved.

The content presented in this chapter is joint work with Sven O. Krumke and is published in [BK22].

4.1. Decompositions and their extension

We start by fixing some notation.

4.3 Convention. From now on, until the end of this chapter, let

- G be a star-like graph
- with perfect matching M
- and cycles $\mathcal{C}_1, \dots, \mathcal{C}_l$ in $G - M$, where $\mathcal{C}_1$ is the centre cycle. ◁

Also, the following definitions are used excessively.

4.4 Definition. For $\varnothing \neq I \subseteq \{1, \dots, l\}$, we denote $\bigcup_{i \in I} V\mathcal{C}_i$ by V_I and $G[V_I]$ by G_I, writing G_i for $G_{\{i\}}$. We write ∂G_I for the set of degree 2 vertices in G_I and call these vertices the *boundary of* G_I. ◁

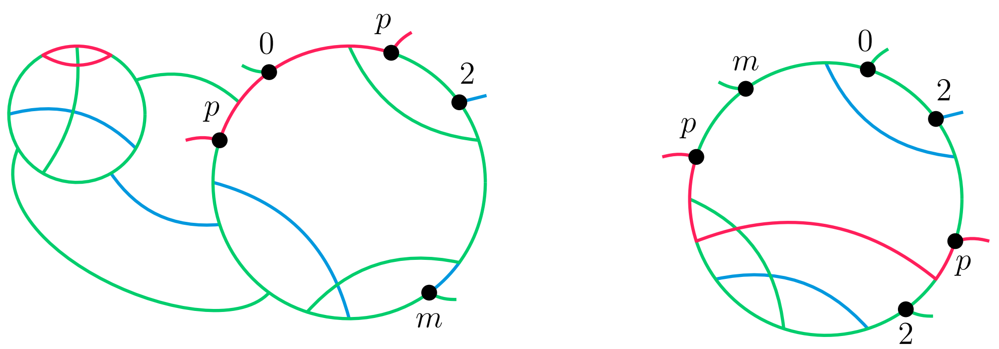
We refer back to Figure 4.1 to illustrate this definition as well. The boundary of G_3 are all three vertices of $\mathcal{C}_3$. For $I = \{1, 3\}$, the graph G_I contains all vertices but the four in $\mathcal{C}_2$ and the five in $\mathcal{C}_4$. Its boundary ∂G_I consists of the vertices in the set $\{u, v, w\}$.

Once more, we recall Definition 2.3 and note that we allow a matching to be part of a decomposition despite not formally being a subgraph. As promised, we begin with the two types of decompositions we need, starting with the one describing the tree we wish to extend. Intuitively, it describes a tree T in G_I, for $1 \in I \subseteq \{1, \dots, l\}$, that could be part of a 3-decomposition of the entire graph. The definition does this by ensuring a few necessary conditions: it requires that all vertices of degree 3 in G_I are part of T. Additionally, edges not in T should either be matching edges or in the set of cycles and paths one gets by restricting a collection of cycles to a subgraph. These paths need to be extended to cycles in a later step, so they must end at the boundary ∂G_I.

4.5 Definition. Let $1 \in I \subseteq \{1, \dots, l\}$ and $\mathcal{D}_I = (T_I, C_I, M_I)$ be a decomposition of G_I such that

- T_I is a tree containing all degree 3 vertices of G_I,
- C_I is a disjoint union of (positive length) paths and cycles in G_I, and
- M_I is a matching.

If all path components (components that are paths) of C_I end at vertices in the boundary ∂G_I, then $\mathcal{D}_I$ is an *I-decomposition.* ◁



(a) An I-decomposition with two cycles. (b) An (A_0, A_p, A_m, A_2)-decomposition.

Figure 4.2.: The two types of decompositions introduced in Definitions 4.5 and 4.7. Green edges should be part of the tree in the final decomposition, red ones are on cycles, and the blue ones form a matching. This is also the colour scheme for figures concerning 3-decompositions in general.

Note that for $I = \{1, \ldots, l\}$ this is just a 3-decomposition since C_I is no longer allowed to contain path components. We also remark that this does not describe all possible restrictions of a 3-decomposition of G to the graph G_I, we would have to allow forests for that to be true, but, for the upcoming proof, trees suffice.

An example of an I-decomposition is shown in Figure 4.2a. There, the edges in T_I are coloured in green, those in C_I are red, and the ones in M_I are blue. The edges on the boundary are not actually part of the decomposition, but they exist and their colours describe which component they should eventually end up in.

A bit of additional notation is useful at this point.

4.6 Definition. We write $A_i\,\mathcal{D}_I$, $i \in \{0, 1, 2\}$, for the set of vertices in ∂G_I that have degree i in T_I, where $A_0\,\mathcal{D}_I$ denotes those that are not in T_I at all. Moreover, we split the set $A_1\,\mathcal{D}_I$ into $A_p\,\mathcal{D}_I$ and $A_m\,\mathcal{D}_I$, where the former contains those degree 1 vertices of T_I that are ends of path components of C_I, whereas the latter contains the ends of matching edges. $\lhd$

As the vertices in $A_1\,\mathcal{D}_I$ have degree 2 in G_I and degree 1 in T_I, they are either in $A_p\,\mathcal{D}_I$ or $A_m\,\mathcal{D}_I$. As a result, we have that ∂G_I is the disjoint union of the four sets $A_x\,\mathcal{D}_I$ for $x \in \{0, p, m, 2\}$. The drawn vertices in Figure 4.2a represent the boundary and those in set $A_x\,\mathcal{D}_I$ are labelled by x.

We can now move on to the second type of decomposition we need. The next definition might seem cryptic at first glance, which is why we shall invest some time into giving an intuition for why this is what we want our decomposition to do. Our goal is to formalise the extension of an I-decomposition $\mathcal{D}_I$ to another cycle $\mathcal{C}_i$, with $i \notin I$, by describing a spanning forest F_i of G_i that satisfies conditions analogous to those of the tree T_I above. Once again, the remaining edges of G_i should either be part of a matching M_i or on cycles or paths in C_i.

But we cannot take just any decomposition of G_i, it also needs to 'fit together' with the I-decomposition that we have. We notice that we do not actually need the details of $\mathcal{D}_I$, it suffices to know the behaviour of the vertices on the boundary of G_I with an edge to $\mathcal{C}_i$. Let u be a vertex in $\partial\, G_I$ with unique neighbour v in $\mathcal{C}_i$, more precisely in $\partial\, G_i$. We now classify the vertices v based on the behaviour of u, placing them into the four sets $A_x \coloneqq N(A_x \mathcal{D}_I) \cap V\mathcal{C}_i$ for $x \in \{0, p, m, 2\}$. Let us see what we want v to satisfy in each of these cases.

If $u \in A_2\, \mathcal{D}_I$ and thus $v \in A_2$, then uv can either be in the tree or matching part, depending on whether we need it to connect to $\mathcal{C}_i$ or not. So vertices of this type actually have some leeway in their behaviour, making them rather useful when finding decompositions. The remaining types will not be as kind.

When u is in $A_p\, \mathcal{D}_I$, we need to extend the path ending at this vertex to a cycle, meaning it must continue in $\mathcal{C}_i$. Hence, this calls for a path at v to another vertex in $\partial\, G_i$ of this type. Thus, we require that the vertices in A_p are exactly the ends of path components of C_i. This is a necessary condition as paths must end at the boundary and all other types of vertices have conflicting behaviour.

If u is in $A_m\, \mathcal{D}_I$, then the edge uv must be part of the tree and we have to ensure that it does not create cycles. This is achieved by requiring that any component of F_i contains at most one vertex that is either in A_m or both a leaf of F_i and in A_2. Such vertices need a tree edge to T_I, which we have already seen for those in A_m and it holds for the leaves as well: the missing edge in G_i at such a vertex must be in M_i as the ends of paths are in A_p.

Finally, u can be in $A_0\, \mathcal{D}_I$, where it needs the edge uv to be part of the tree and v must be connected to T_I in $\mathcal{C}_i$. We ensure that v ends up in a component of F_i that can be connected to T_I. For this we require every component of F_i to contain an element of $A_2 \cup A_m$, which are vertices that can or need to connect to T_I. With these ideas at hand, let us give the definition.

4.7 Definition. Let (A_0, A_p, A_m, A_2) be a partition of $\partial\, G_i$ for some $i \in \{1, \dots, l\}$. Also let $\mathcal{D}_i = (F_i, C_i, M_i)$ be a decomposition of G_i such that

- F_i is a spanning forest in G_i,
- C_i is a disjoint union of (positive length) paths and cycles in G_i, and
- M_i is a matching.

The decomposition $\mathcal{D}_i$ is an (A_0, A_p, A_m, A_2)-*decomposition of* $\mathcal{C}_i$ if it satisfies the following two conditions.

(i) Every component of F_i contains a vertex in $A_2 \cup A_m$ and it contains at most one vertex v such that $v \in A_m$ or v is both in A_2 and a leaf of F_i.

(ii) The set of ends of path components of C_i is exactly A_p. $\triangleleft$

Figure 4.2b on Page 48 visualises such a decomposition, where vertices in A_x are labelled by x for $x \in \{0, p, m, 2\}$ and the colour scheme is analogous to before: the edges in F_i are coloured in green, those in C_i are red, and the ones in M_i are blue.

We are now in a position to prove that the conditions we incorporated into our definitions suffice to let us extend an I-decomposition. More precisely, we show that an I-decomposition can be extended to a new cycle $\mathcal{C}_i$ if we have an (A_0, A_p, A_m, A_2)-decomposition there. Here the sets A_x, for $x \in \{0, p, m, 2\}$, are assigned exactly those vertices we gave them when we described the intuition behind Definition 4.7. Afterwards, we only need to take a look at what kinds of decompositions we can find in the cycles $\mathcal{C}_i$ and how we can piece them together to obtain one of G.

(4.2) **4.8 Lemma.** Let $1 \in I \subseteq \{1, \dots, l\}$, $i \notin I$, $J := I \cup \{i\}$, and $\mathcal{D}_I := (T_I, C_I, M_I)$ be an I-decomposition of G. If there exists an (A_0, A_p, A_m, A_2)-decomposition $\mathcal{D}_i := (F_i, C_i, M_i)$ of $\mathcal{C}_i$ where

$$A_x := N(A_x\, \mathcal{D}_I) \cap V\mathcal{C}_i \text{ for } x \in \{0, p, m, 2\}\,,$$

then G has a J-decomposition (T_J, C_J, M_J) with

$$T_I \cup F_i \subseteq T_J,\ C_I \cup C_i \subseteq C_J, \text{ and } M_I \cup M_i \subseteq M_J. \qquad \lhd$$

Proof. Let $\mathcal{D}_I$ and $\mathcal{D}_i$ be decompositions as in the claim. In order to get a decomposition $\mathcal{D}_J$ as desired, we need to assign the edges in $E(VG_I, V\mathcal{C}_i)$ to the graphs $T_I \cup F_i$, $C_I \cup C_i$, and the set $M_I \cup M_i$. Note that, by definition of the sets A_x,

$$E(VG_I, V\mathcal{C}_i) = E(\partial\, G_I, V\mathcal{C}_i) = \bigcup_x E(A_x\, \mathcal{D}_I, A_x) \text{ where } x \in \{0, p, m, 2\}\,.$$

We add the set $E(A_p\, \mathcal{D}_I, A_p)$ to $C_I \cup C_i$ to get C_J. The sets $E(A_m\, \mathcal{D}_I, A_m)$ and $E(A_0\, \mathcal{D}_I, A_0)$ are both added to $T_I \cup F_i$. Additionally, for any component K of F_i that contains a vertex of A_2 but none of A_m, we pick a vertex in $A_2 \cap VK$ of least degree in K and add the edge incident to it with an end in $A_2\, \mathcal{D}_I$ to the tree part as well. Such vertices exist by Condition (i) of Definition 4.7 and the minimality just means that we choose a leaf in case one is present. This yields T_J. The remaining edges of $E(A_2\, \mathcal{D}_I, A_2)$ are added to $M_I \cup M_i$ to get M_J.

We claim that $\mathcal{D}_J := (T_J, C_J, M_J)$ is a desired J-decomposition of G. The set C_J is the union of two disjoint graphs C_I and C_i, both of which consist of paths and cycles, together with edges $E(A_p\, \mathcal{D}_I, A_p)$ connecting degree 1 vertices of these subgraphs. Hence it, too, is a disjoint union of paths and cycles as required. Furthermore, a degree 1 vertex in C_J must have degree 1 in C_I or C_i. In the first case, it is an element of $\partial\, G_I$ by definition of an I-decomposition and it cannot be part of $N(V\mathcal{C}_i)$ without increasing its degree when we add the edges in $E(A_p\, \mathcal{D}_I, A_p)$. So it is in $\partial\, G_J$ as desired. The second case does not occur as vertices of degree 1 in C_i are in A_p and have degree 2 in C_J.

The set M_J is also a matching as it is the union of two matchings M_I and M_i in disjoint subgraphs and the additional edges are part of $E(A_2\, \mathcal{D}_I, A_2)$, meaning their ends in G_I

have degree 2 in $T_I \subseteq T_J$. Their ends in $\mathcal{C}_i$ also have degree 2 in $F_i \subseteq T_J$ as a lower degree makes them a leaf or an isolated vertex of F_i. In the first case the component containing that vertex cannot contain a vertex in A_m and the leaf is unique, meaning the edge is added to T_J by our construction. The second case is faced with a component having a unique edge to G_I, which is also added to T_J.

This just leaves T_J. Let $\mathcal{K}$ be the set of components of F_i and let F be the union of T_I with the components in $\mathcal{K}$. By adding the edges of $E(A_m\,\mathcal{D}, A_m)$ to F we have connected all components $K \in \mathcal{K}$ that contain a vertex of A_m to T_I by exactly one edge each. The result is a new forest F' consisting of a tree $T' \supseteq T_I$ and remaining components $\mathcal{K}' \subseteq \mathcal{K}$ that have no vertex in A_m. By adding our chosen elements of $E(A_2\,\mathcal{D}_I, A_2)$ we connect the components of $\mathcal{K}'$, which all contain an element of A_2, to T' by exactly one edge. This results in a tree T''. Finally, the edges in $E(A_0\,\mathcal{D}_I, A_0)$ connect vertices of G_I that are not in T'' to it by a single edge, creating the tree T_J.

Now we only need to check that T_J spans all vertices of degree 3 in G_J. To this end, consider a vertex of G_J that is not part of T_J. It cannot be in $\mathcal{C}_i$ as all the components of F_i are part of T_J and F_i was spanning. A vertex in G_I that is not part of T_J is also not in T_I, putting it in $A_0\,\mathcal{D}_I$. But such vertices still have degree 2 in G_J as they cannot be in $A_0\,\mathcal{D}_I \cap N(V\mathcal{C}_i)$ because the degree of such a vertex is 1 now. □

4.2. Finding decompositions in cycles

In this section, let $\mathcal{C}_i$ be some cycle in $G - M$, as defined in Convention 4.3, and A_0, A_p, A_m, and A_2 be a partition of $\partial\, G_i$ for which we want to find an (A_0, A_p, A_m, A_2)-decomposition. We need four different types of decompositions in order to handle all cases that occur when combining them.

Before we start, a bit more notation will come in handy, which we now introduce.

4.9 Definition. For a chord $e \in G_i$ of $\mathcal{C}_i$ we get two paths in $\mathcal{C}_i$ between its ends which, together with the chord, yield two cycles, say C' and C''. We call $C \in \{C', C''\}$ *minimal* or a *minimal cycle of* $\mathcal{C}_i$ if it is a chordless cycle in G. The unique edge in $M \cap E\,C$ of a minimal cycle C is denoted by e_C and we write P_C for the path $C - e_C$. ◁

We directly observe that, if we want to spare a vertex $x \in \partial\, G_i$, then we can find a minimal cycle in $\mathcal{C}_i - x$, as long as $\mathcal{C}_i$ has a chord.

4.10 Observation. If $\mathcal{C}_i$ has a chord, then there is a minimal cycle of $\mathcal{C}_i$ that avoids any specific vertex in $\partial\, G_i$. ◁ (4.12) (4.14)

Proof. Let e be a chord of $\mathcal{C}_i$ and let P_1 and P_2 be the two paths in $\mathcal{C}_i$ between its ends. For $j \in \{1, 2\}$, we choose $v_j w_j \in M$ such that $d_{P_j}(v_j, w_j)$ is minimal. Using these edges, we find two minimal cycles $v_j P_j w_j v_j$ that meet disjoint sets of vertices of $\partial\, G_i$. Note that the vertices v_j, w_j always exist as the ends of e are candidates. The cycles are minimal as a chord of $v_j P_j w_j v_j$ must be an edge of M whose ends have smaller distance. □

A useful construction that we apply regularly is the following. Let C be a minimal cycle of $\mathcal{C}_i$ that does not contain some vertex $x \in A_2 \cup A_m$ and where $V C \cap \partial G_i$ contains only vertices of $A_2 \cup A_m$. Additionally, we require that $A_p = \varnothing$ and $|A_m \setminus V C| \leq 1$. We assign the edges of $E G_i$ to our three components by setting

$$C_i := C,\ M' := M \cap (E G_i \setminus E C), \text{ and } F' := \mathcal{C}_i - E C.$$

In this assignment C_i is a cycle and thus contains no path components, M' is a matching, and F' consists of a path P together with a set of isolated vertices. As $x \in P$, this path is not disjoint from $A_2 \cup A_m$ and has no leaf in ∂G_i as its ends are incident to a chord, so it satisfies Condition (i) of Definition 4.7. Let v be an isolated vertex of F'. If v has degree 3 in G_i, then it is incident to an edge $vu \notin C$ whose other end is in P. We remove this edge from M' and add it to F', leaving F' acyclic by adding a new leaf to the tree P. As this leaf has degree 3 in G_i, the larger component continues to satisfy Condition (i). In the case where v has degree 2 in G_i, we assumed that $v \in A_2 \cup A_m$ and this component also satisfies Condition (i). The resulting spanning forest F_i and matching M_i therefore form an (A_0, A_p, A_m, A_2)-decomposition of $\mathcal{C}_i$ together with C_i. We call this the *decomposition given by C*. To summarise, we obtain the following result.

(4.12) (4.14) (4.18) **4.11 Observation.** If C is a minimal cycle of $\mathcal{C}_i$ satisfying that

$$V C \cap \partial G_i \subsetneq A_2 \cup A_m,\ |A_m \setminus V C| \leq 1 \text{ and } A_p = \varnothing,$$

then the decomposition given by C is an (A_0, A_p, A_m, A_2)-decomposition of $\mathcal{C}_i$. ◁

We now show that certain (A_0, A_p, A_m, A_2)-decompositions exist, starting with $A_0 = \varnothing$, $A_p = \varnothing$, $A_m = \{x\}$ for some $x \in \partial G_i$, and $A_2 = \partial G_i \setminus A_m$.

[4.10] [4.11] (4.2) (4.13) **4.12 Lemma.** There exists an $(\varnothing, \varnothing, \{x\}, A_2)$-decomposition of $\mathcal{C}_i$. ◁

Proof. If the cycle $\mathcal{C}_i$ is chordless, $V\mathcal{C}_i = \partial G_i$ and $E G_i = E \mathcal{C}_i$. Here, setting

$$F_i := G_i[\varnothing],\ C_i := \mathcal{C}_i \text{ and } M_i := \varnothing$$

does the trick. In the case where the cycle $\mathcal{C}_i$ has a chord, it contains a minimal cycle C that avoids x by Observation 4.10 and we can use the decomposition given by C by Observation 4.11. □

We also find decompositions this way when all elements of ∂G_i are in A_2, since this is just a weaker requirement.

[4.12] (4.2) **4.13 Corollary.** If $\partial G_i \neq \varnothing$, then $\mathcal{C}_i$ has an $(\varnothing, \varnothing, \varnothing, \partial G_i)$-decomposition. ◁

Proof. By Lemma 4.12 there exists an $(\varnothing, \varnothing, \{x\}, A_2 \setminus \{x\})$-decomposition of $\mathcal{C}_i$ for some $x \in A_2$. This is an $(\varnothing, \varnothing, \varnothing, A_2)$-decomposition by definition. □

Next, let $A_0 = \{x\}$ for some $x \in \partial G_i$, $A_p = \varnothing$, $A_m = \varnothing$, and $A_2 = \partial G_i \setminus A_0$.

4.14 Lemma. If $A_2 \neq \varnothing$, then there exists an $(\{x\}, \varnothing, \varnothing, A_2)$-decomposition of $\mathcal{C}_i$. $\triangleleft$ [4.10] [4.11] (4.2)

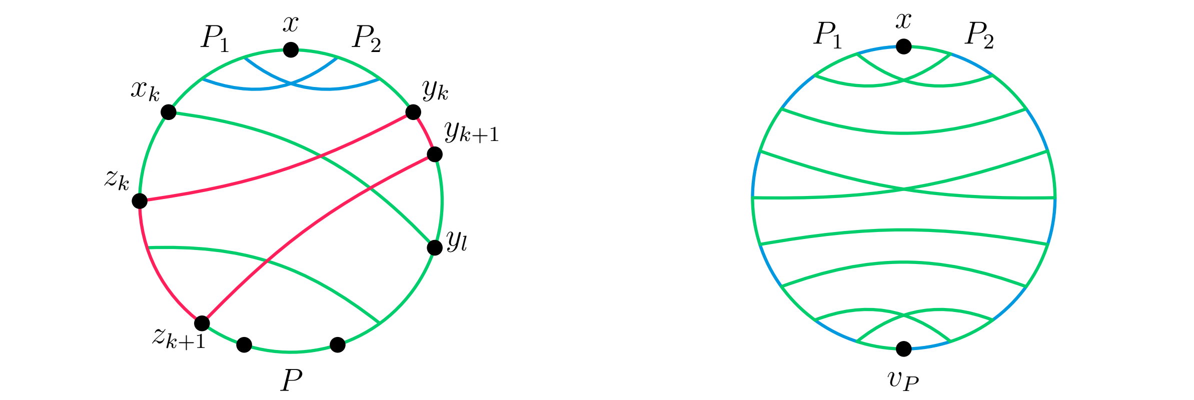
Proof. We begin by looking at the case where $\mathcal{C}_i$ is chordless. Let y be a neighbour of x in $\mathcal{C}_i$ and consider the spanning tree $F_i \coloneqq \mathcal{C}_i - xy$. This contains an element of A_2 and it has only one leaf in A_2, namely y. Thus, F_i satisfies Condition (i) of Definition 4.7. The last missing edge xy of $E\,G_i$ is assigned to M_i, making this a matching and leaving C_i with no edges and thus no path component. This gives us an $(\{x\}, \varnothing, \varnothing, A_2)$-decomposition.

Now we assume that $\mathcal{C}_i$ has a chord. The existence of a minimal cycle C that neither contains x nor all elements of A_2 is another good case since it satisfies both requirements necessary for us to obtain a decomposition given by C, see Observation 4.11.

In the final and most complicated case, we may assume that $\mathcal{C}_i$ has chords but none of them yield a cycle as described above. We have already seen that any chord naturally gives rise to two minimal cycles C, C' for which P_C, $P_{C'}$ have no inner vertex in common in Observation 4.10. Consequently one of them must contain x while the other contains all vertices of A_2. Hence, all vertices of A_2 must form a path P: a vertex of degree 3 between them is incident to a chord and this would yield a minimal cycle as above. Let P_1 and P_2 be the two paths in $\mathcal{C}_i - E\,P$ between x and the ends of P. Then any chord e of $\mathcal{C}_i$ must connect an inner vertex of P_1 to one of P_2: if both were on the same path, then one of the two minimal cycles that e yields would contain neither x nor any element of A_2, contradicting our assumption. Hence, P_1 and P_2 have the same length.

Let $x_1, \dots, x_r$ and $y_1, \dots, y_r$ be the inner vertices of P_1 and P_2 respectively, ordered by increasing distance to x. We call a chord $x_k y_l$ of $\mathcal{C}_i$ *short* if $|k - l| \leq 1$ and *long* otherwise. As an illustration, the two blue chords in Figure 4.3a are short, whereas the edge $x_k y_l$ is a long chord. It turns out that long chords are helpful and the presence of only short ones is structurally very restrictive. More precisely, we prove by induction on $1 \leq d \leq r$ that if all chords $x_k y_l$ in G_i with $k, l \leq d$ are short, then we have $x_k y_l$ is in G if and only if $x_l y_k$ is, for all $k, l \leq d$. The case that $d \leq 1$ is clear as the only candidate is $x_1 y_1$. So let it hold up to $d - 1$ for $d \geq 2$ and let $x_k y_l$ satisfy $k,\ l \leq d$. If $k,\ l < d$ or $k,\ l = d$ we are done, so we may assume, by symmetry, that $k = d$ and $l = d - 1$. But x_{d-1} is matched to a vertex in $\{y_{d-2}, y_{d-1}, y_d\}$ by M. Of these three, only y_d is an option as y_{d-1} is taken by x_d and $x_{d-1} y_{d-2}$ cannot be in M by induction since $x_{d-2} y_{d-1}$ is not.

With this preparation, we can now look at the case in which a long chord $x_k y_l$ exists. We choose one minimising $\min\{k, l\}$ and may, by symmetry, assume that $k < l$ and all chords with an end of index at most $k - 1$ are short. As a result, the vertices y_k, y_{k+1} are matched to vertices in $x_{k+1} P_1$: this holds as neither is matched to x_k whose neighbour in $G[M]$ is y_l with $l \geq k + 2$. All vertices in $P_1 x_{k-2}$ have neighbours in $P_2 y_{k-1}$, so none of these are possible either. This just leaves the vertex x_{k-1}. Since $x_k y_{k-1} \notin M$, y_{k-1} is matched to x_{k-1} or x_{k-2}, giving us $x_{k-1} y_{k-1} \in M$ or $x_{k-1} y_{k-2} \in M$, eliminating x_{k-1} as well.



(a) The cycle obtained from a long chord $x_k y_l$. (b) The case when all chords are short.

Figure 4.3.: The decompositions used in the proof Lemma 4.14. On the left, the presence of a long chord yields a decomposition from a cycle. On the right the non-existence of long chords leads to a Hamiltonian path through the cycle.

Now, let z_k, z_{k+1} be the neighbours of y_k, y_{k+1} in $G[M]$ and take a look at the cycle

$$C := y_k y_{k+1} z_{k+1} P_1 z_k y_k$$

shown in Figure 4.3a. We now apply a construction similar to the one for minimal cycles: assign the edges of $E\,G_i$ to the three components by setting

$$C_i := C,\ M' := M \cap E\,G_i \setminus E\,C, \text{ and } F' := \mathcal{C}_i - E\,C.$$

Then C_i is a cycle, M' is a matching, and F' consists of two paths together with isolated vertices. The ends of these paths are part of C, so they have degree 3 in G_i and we can connect the two paths using the long chord $x_k y_l$. This replaces the two paths by a tree T' and the isolated vertices have degree 3 in G_i with a neighbour in P_2. As their neighbours are not on C, they are part of T' and we can connect them by adding such edges to T'. Give F_i the edges of T' and put the remaining edges into M_i, then $M_i \subseteq M'$ is still a matching and F_i is a spanning tree. Since F_i contains P and thus (all) vertices of A_2, the conditions of an $(\{x\}, \varnothing, \varnothing, A_2)$-decomposition are satisfied.

This just leaves the case that all chords are short, which makes use of the very potent knowledge of the way these behave. Here, we give C_i no edges of $E\,G_i$, instead we divide them up amongst F_i and M_i. Note that the degree 2 vertices of G_i are those in the set $\{x\} \cup A_2$ where the vertices of A_2 are all on P. We suppress all vertices of degree 2 except x and one element of A_2. The resulting graph G' is left with just two vertices of degree 2, namely x and a vertex v_P that has replaced the entire path P. The paths P_1 and P_2 are now xv_P-paths, so we have $P_1 = xx_1 \dots x_r v_P$ and $P_2 = xy_1 \dots y_r v_P$. An illustration of this and the decomposition we now choose can be found in Figure 4.3b.

Since P_1 and P_2 have the same length, $P_1 \cup P_2$ is a Hamiltonian cycle of even length. Its edges thus decompose into two perfect matchings M_1 and M_2, and we claim that the graph $Q := G' - M_1$ is a Hamiltonian xv_P-path in G'. To see that this is indeed the case, notice that all vertices in $V G' \setminus \{x, v_P\}$ have degree 2 in Q, where the two excluded ones have degree 1. Hence, Q consists of an xv_P-path and possibly additional cycles. Suppose it contains a cycle C and choose a vertex of minimal index in C. By symmetry, we assume this vertex is $x_k \in P_1$. Then the edge x_kx_{k-1} is in M_1 as $x_{k-1} \notin C$ (where x_0, $y_0 \coloneqq x$ and x_{r+1}, $y_{r+1} \coloneqq v_P$). Thus, x_kx_{k+1} and y_ky_{k-1} are in M_2 and Q. The edge x_ky_{k-1} cannot be in G' either since y_{k-1} is not part of C. But now $x_ky_k \in Q$ or x_ky_{k+1}, $x_{k+1}y_k \in Q$. Both yield an x_ky_{k-1}-path in C, $x_ky_ky_{k-1}$ or $x_kx_{k+1}y_ky_{k-1}$ respectively, a contradiction.

Hence, Q is a Hamiltonian path and we can obtain the desired decomposition by replacing v_P by the path P again and putting all edges of $P \cup Q$ into F_i, which is a Hamiltonian path in G_i ending at x and at an end of P. The graph C_i receives no edges and $M_i \coloneqq M_1$. This is an $(\{x\}, \varnothing, \varnothing, A_2)$-decomposition of $\mathcal{C}_i$ since the only component of F_i contains an element of A_2 and it only has one leaf in A_2. □

Here we remark that this lemma was also obtained by Xie, Zhou, and Zhou [XZZ20, Lemma 2.3]. Their formulation essentially describes the two cases in our proof, as they either get a decomposition containing a cycle with two chords or a Hamiltonian path. We repeated the statement to make it fit into our notation. The reason we also presented our proof is that it is different and we believe that it reveals more structure of the graph. Our case distinction is based on the existence of long chords and we obtained that either the graph has one or all chords are short, in which case a Hamiltonian path is present. But we also know that all chords in this case are either of the form x_iy_i or they come in pairs x_iy_{i+1}, $x_{i+1}y_i$. Xie, Zhou, and Zhou prove this by distinguishing whether or not $\mathcal{C}_i$ has a non-separating two-chord cycle. This turns out to be exactly our distinction, as we obtain such cycles in the case that there is a long chord and they do not exist when all chords are short, but it obscures the structure of the chords.

Lastly, we let $A_0 = \varnothing$, $A_p = \{x, y\}$ for x, $y \in \partial G_i$, $A_m = \varnothing$, and $A_2 = \partial G_i \setminus A_p$. Due to the abundance of indices needed in the proof, we denote the considered cycle by $\mathcal{C}$ instead of $\mathcal{C}_i$ and write $G_\mathcal{C}$ for $G[V\mathcal{C}]$.

4.15 Lemma. If G is 3-connected, then $\mathcal{C}$ has an $(\varnothing, \{x, y\}, \varnothing, A_2)$-decomposition. ◁ (4.2)

Proof. Let P, P' be the two xy-paths in $\mathcal{C}$. Since $G - \{x, y\}$ is connected, A_2 is non-empty and we may assume that, without loss of generality, $VP' \cap A_2 \neq \varnothing$. Next, let

$$u_1v_1, u_2v_2, \dots, u_sv_s$$

be a maximal sequence of edges of M with ends in P satisfying, for all $i \in \{1, \dots, s\}$, that

(a) the path $P_i \coloneqq u_iPv_i$ is disjoint from all prior ones, that is, $P_i \subseteq P - \bigcup_{k<i} P_k$,
(b) P_i either contains an element of A_2 or there is an edge $e_i \in E(VP_i, X_i)$ where $X_i \coloneqq VP' \cup \bigcup_{k<i} VP_k$, and
(c) P has no vertices u', v' with $u'v' \in M$ and $P_i \subsetneq u'Pv' \subseteq P - \bigcup_{k<i} P_k$.

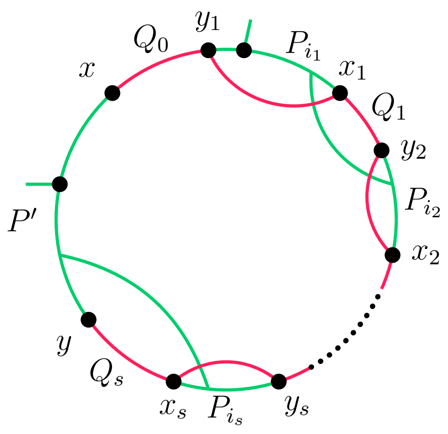


Figure 4.4.: The notation used in the proof of Lemma 4.15. The visible part of Q is red, that of F is green, and matching edges are omitted in favour of clarity.

We remark that these paths P_i end up as part of the tree and Property (b) ensures that they either contain an element of A_2 (and can form a component on their own) or can connect to a prior path or P', which will have such a vertex by induction. Also notice that Property (c) can be read as 'P_i is chosen maximally' in the sense that it forbids the existence of candidates u', v' for u_i, v_i that would yield a longer path.

Let $X_u \coloneqq \{u_i \colon i = 1, \dots, s\}$, $X_v = \{v_i \colon i = 1, \dots, s\}$. We assume that u_i occurs before v_i in P for all i. By Property (a), no P_i contains an element of $X_u \cup X_v$ as an inner vertex, meaning vertices of X_u and X_v alternate. Now let y_i (x_i) be the ith occurrence of a vertex of X_u (X_v) in P, for $i \in \{1, \dots, s\}$, which is just a labelling of the vertices in the order they appear on the path. We refer to Figure 4.4 to keep track of the notation.

We show that for $Q_i \coloneqq x_i P y_{i+1}$, where $i \in \{0, \dots, s\}$ and $x_0 \coloneqq x$, $y_{s+1} \coloneqq y$, there are no edges $u'v'$ in $E(V\mathring{Q}_i, V\mathring{Q}_j) \cap M$, for $i < j$: let m be the minimal index in $\{1, \dots, s\}$ such that P_m is part of $y_{i+1} P x_j$ where m is well-defined as $y_{i+1} P x_j$ contains at least the path $y_{i+1} P x_{i+1}$. Be aware that $y_{i+1} P x_{i+1}$ need not be P_m as we choose the path observed first by the construction, which might appear later in the ordering. Vertices $u' \in \mathring{Q}_i$, $v' \in \mathring{Q}_j$ satisfy

$$P_m \subsetneq u'Pv' \subseteq P - \bigcup_{k<m} P_k,$$

where the last subset relation uses the choice of m: before it, no paths in $x_i P y_{j+1} \supseteq u'Pv'$ were chosen. Hence, Property (c) implies that $u'v' \notin M$.

As exemplified in Figure 4.4, we now set

$$\begin{aligned} Q &\coloneqq \bigcup_{i=0}^{s} Q_i + \{x_i y_i \colon i \in \{1, \dots, s\}\} = Q_0 y_1 x_1 Q_1 y_2 \dots x_s Q_s, \\ F' &\coloneqq P' \cup \bigcup_{i=1}^{s} P_i + \{e_i \colon VP_i \cap A_2 = \varnothing\}, \text{ and} \\ F &\coloneqq (V\mathcal{C}, E\,F' \cup (E(S, VF') \cap M)) \text{ where } S \coloneqq \bigcup_{i=0}^{s} V\mathring{Q}_i. \end{aligned}$$

Notice that F' is well-defined: the edges e_i exist in the specified case by Property (b) and we have just fixed one of them arbitrarily. We now show that

$$F_{\mathcal{C}} \coloneqq F,\ C_{\mathcal{C}} \coloneqq Q, \text{ and } M_{\mathcal{C}} \coloneqq M \setminus (E\,F \cup E\,Q)$$

is an $(\varnothing, \{x, y\}, \varnothing, A_2)$-decomposition of C. The edges of $G_{\mathcal{C}}$ are partitioned completely as $F \cup Q$ contains all edges of $\mathcal{C}$. Additionally, $M_{\mathcal{C}}$ is a matching and $C_{\mathcal{C}}$ consists of a path from x to y. We thus need to show that F is a spanning forest in $G_{\mathcal{C}}$ whose components each contain an element of A_2, at most one of which is a leaf.

We first show that every vertex of $\mathring{Q}_i$, for $i \in \{0, \dots, s\}$, is either in A_2 or adjacent to a vertex of F' in $G[M]$. To see this, let W be the set of vertices in $\mathring{Q}_i$ that satisfy one of these two conditions and assume $W \neq V\mathring{Q}_i$. Then take two elements w, z of $W \cup \{x_i, y_{i+1}\}$ of minimal distance in Q_i such that wPz contains an element of $V\mathring{Q}_i \setminus W$. As $G - \{w, z\}$ is connected, there exists an edge e incident to a vertex u of $\mathring{w}P\mathring{z}$ with other end in $G - wPz$. We show that this edge could be used to extend the sequence $u_1v_1, u_2v_2, \dots, u_sv_s$, contradicting maximality.

Since $A_2 \cup \{x, y\}$ is disjoint from $\mathring{w}P\mathring{z}$, the other end v of e is also in $\mathcal{C}$. By assumption, it is a matching edge that does not end at a vertex in F'. But we already know that e cannot end in another $\mathring{Q}_j$, so it must have both ends in $\mathring{Q}_i$. Hence it satisfies Properties (a) and (b), where the latter holds as either w or z is part of the resulting path and this vertex is in W. Consequently, e either satisfies Property (c) or there exist vertices u', v' as stated there. Choosing them to have maximal distance (in $\mathring{Q}_i$) yields an edge $u'v'$ that satisfies all three properties. In either case we get a contradiction to the maximality of the sequence $u_1v_1, u_2v_2, \dots, u_sv_s$.

To see that F is a forest, note that F' is the disjoint union of paths with solitary edges connecting those with $VP_i \cap A_2 = \varnothing$ to ones prior in the ordering. This ensures that F' is acyclic and its leaves are precisely the ends of the paths P_i and P', none of which are in A_2. The transition to F now only adds edges of M between a vertex of F' and one not in F', which becomes a leaf of the component and is not in A_2. So F is a spanning forest of $G_{\mathcal{C}}$ without any leaves in A_2. Now take a component K of F'. If it contains one of the paths P_i then it has a vertex of A_2. This just follows from Property (b), for P_1 directly and for the others inductively. Any component of F that does not contain such a path must be an isolated vertex in $S = \bigcup_{i=0}^{s} V\mathring{Q}_i$ without an edge of M to a vertex in F'. But as vertices of S that do not have such an edge are in A_2 by the previous two paragraphs, we get that these components have a vertex of A_2, too. □

This lemma also requires us to take a look at [XZZ20] again. Our construction is similar to the one found in Lemma 2.1 there, though our assumption and obtained decomposition differ. If we formulate their lemma in our notation, we obtain the following.

4.16 Lemma. *Let x, $y \in \partial G_i$ and $A_2 = \partial G_i \setminus \{x, y\} \neq \varnothing$. The cycle $\mathcal{C}_i$ has an $(\varnothing, \{x, y\}, \varnothing, A_2)$-decomposition or there exists an $(A_0', \varnothing, \varnothing, A_2')$-decomposition for any choice of A_0', A_2' with $A_0' \cup A_2' = \partial G_i$, $A_0' \cap A_2' = \varnothing$, and $A_2' \neq \varnothing$.* ◁

We do, however, need the first decomposition to exist in our proof and cannot use this to eliminate the 3-connectivity requirement from Lemma 4.15.

4.3. Proof of the main theorem

To shorten the proof of Theorem 4.2, we define *desirable* I-decompositions and show that they exist. These are basically just decompositions of the centre cycle where the required behaviour of the remaining cycles corresponds to one of our previous lemmas.

4.17 Definition. Let $I := \{1\}$. We call an I-decomposition $\mathcal{D}_I$ of G *desirable* if every cycle $\mathcal{C}_j$ with $j > 1$ satisfies that

- $|A_0\,\mathcal{D}_I \cap N(V\mathcal{C}_j)| \leq 1$,
- $|A_m\,\mathcal{D}_I \cap N(V\mathcal{C}_j)| \leq 1$,
- $|A_p\,\mathcal{D}_I \cap N(V\mathcal{C}_j)| \in \{0, 2\}$, and
- at most one of the sets $A_x\,\mathcal{D}_I \cap N(V\mathcal{C}_j)$ for $x \in \{0, m, p\}$ is non-empty. $\triangleleft$

We begin by showing that desirable decompositions exist.

[4.11] (4.2) **4.18 Lemma.** Let $I = \{1\}$, then G has a desirable I-decomposition. $\triangleleft$

Proof. We may assume that $l > 1$ as otherwise G is Hamiltonian and has a 3-decomposition. We begin by taking a look at the case where $\mathcal{C}_1$ has a chord. This lets us basically repeat the construction for decompositions given by cycles (see Observation 4.11). We choose some minimal cycle C of G_1 and consider the number of elements in $N(V\mathcal{C}_j)$ that are on C for cycles $\mathcal{C}_j$ with $j > 1$. If these sets contain at most one element for all cycles, we set $C_I := C$ and let T be the path in $\mathcal{C}_1 - E\,C$. The path T is a tree spanning all degree 3 vertices of G_1 except those on C. As C is minimal, any such vertex is connected to T by an edge of M and we add this to T to obtain T_I. The remaining edges, which are a subset of M, form M_I and this is an I-decomposition $\mathcal{D}_I$. For this decomposition the sets $A_m\,\mathcal{D}_I$ and $A_p\,\mathcal{D}_I$ are both empty. Furthermore $A_0\,\mathcal{D}_I \cap N(V\mathcal{C}_j)$ contains at most one element of C and all vertices of $\partial\, G_1$ that are not in C are in $A_2\,\mathcal{D}_I$, so the conditions of a desirable decomposition are satisfied. This case is illustrated in Figure 4.5a.

Next, we assume that C contains multiple vertices of some $N(V\mathcal{C}_k)$. Choosing two vertices in $P_C \cap N\mathcal{C}_k$ for some k so as to minimise the distance between them, let P by the path between them in P_C. This path contains at most one vertex from sets $N(V\mathcal{C}_j)$ for $j \neq k$, $j > 1$. We apply a similar construction, where we set $C_I := P$ and let T be the path $\mathcal{C}_1 - V\mathring{P}$. Again we connect vertices of degree 3 in G_I that are not part of T yet and obtain T_I and an I-decomposition. This, too, is desirable as $A_m\,\mathcal{D}_I$ is still empty, $A_p\,\mathcal{D}_I$ contains exactly the two ends of P, which are in $N(V\mathcal{C}_k)$, and $A_0\,\mathcal{D}_I \cap N(V\mathcal{C}_j)$ is empty for $j = k$ and has at most one element otherwise. Figure 4.5b shows the decomposition constructed in this case.

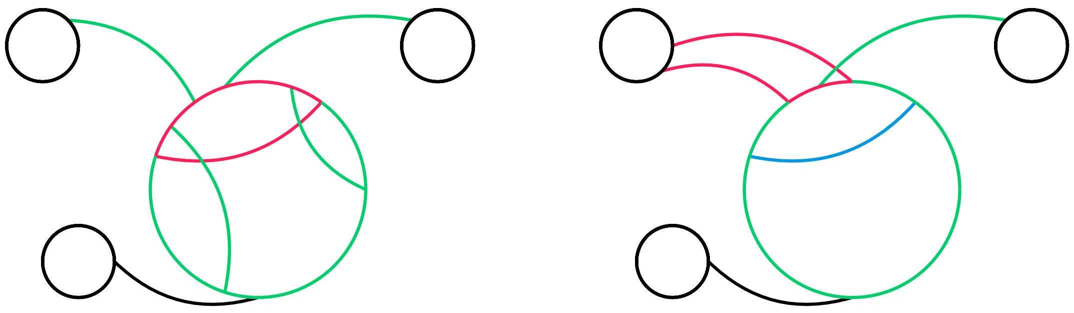

(a) T_I in case no cycle has multiple edges to C. (b) And T_I in case the cycle $\mathcal{C}_k$ does.

Figure 4.5.: The decompositions used in Lemma 4.18 if the centre has a chord.

Finally, in the case that $\mathcal{C}_1$ is chordless, we take two adjacent vertices u, v on $\mathcal{C}_1$. If they have neighbours in different cycles, we set $T_I \coloneqq \mathcal{C}_1 - uv$ and $M_I \coloneqq \{uv\}$, leaving C_I empty. This gives us a spanning tree and a matching that form a desirable I-decomposition $\mathcal{D}_I$ since all vertices are in $A_2\,\mathcal{D}_I$ except for u and v, which are part of $A_m\,\mathcal{D}_I$ and in different sets $N(V\mathcal{C}_j)$. Should u and v be in the same set $N(V\mathcal{C}_k)$, then we add uv to C_I instead and obtain another desirable I-decomposition, this time with $A_p\,\mathcal{D}_I = \{u, v\} \subseteq N(V\mathcal{C}_k)$. □

Now we can finally finish up the proof of Theorem 4.2.

Proof (of Theorem 4.2). Let $I \coloneqq \{1\}$. By Lemma 4.18 there exists a desirable I-decomposition $\mathcal{D}_{\{1\}}$ of G. We now iteratively extend this decomposition to more cycles by checking the conditions of Lemma 4.8 and verifying that we can satisfy them with the help of Corollary 4.13 and Lemmas 4.12, 4.14, and 4.15.

As long as $I \neq \{1, \dots, l\}$ take an element $i \notin I$ and set $J \coloneqq I \cup \{i\}$. Then we can apply Lemma 4.8 which gives us a J-decomposition if we can exhibit an (A_0, A_p, A_m, A_2)-decomposition where $A_x \coloneqq N(A_x\,\mathcal{D}_I) \cap V\mathcal{C}_i$, $x \in \{0, p, m, 2\}$. As G is star-like, we know that $\partial G_I \cap N\mathcal{C}_i = \partial G_1 \cap N\mathcal{C}_i$. Using that $\mathcal{D}_{\{1\}}$ is desirable we can conclude that all vertices in $\partial G_I \cap N\mathcal{C}_i$ are in $A_2\,\mathcal{D}_I$, with the possible exception of either one element in $A_0\,\mathcal{D}_I$, one in $A_m\,\mathcal{D}_I$, or two in $A_p\,\mathcal{D}_I$. Consequently, we have that all vertices in ∂G_i are in A_2 aside from a single one in either A_0 or A_m, or two in A_p. Note that this is true initially and remains true in later steps as Lemma 4.8 ensures that edges in the centre are never reassigned. The set ∂G_i contains at least three elements as it separates $\mathcal{C}_i$ from $\mathcal{C}_1$ in G and G is 3-connected. Hence, $A_2 \neq \varnothing$. We have thus fulfilled the premise of Corollary 4.13 and Lemmas 4.12, 4.14, and 4.15, giving us an (A_0, A_p, A_m, A_2)-decomposition and completing the proof. □

4.4. Extending the decompositions

In this section, we briefly sketch how our definitions can be extended in order to encompass a larger class of graphs. We still assume the existence of a perfect matching but we wish to allow more general contraction graphs than just stars. Definition 4.5 needs no immediate adjustment, though we can use $I \neq \varnothing$ instead of distinguishing the first cycle now, since we need not keep track of the centre of the star.

However, Definition 4.7 cannot be used in the more general case where multiple cycles can have edges to $\mathcal{C}_i$. If we wish to extend an I-decomposition to this cycle, we would need all neighbouring cycles to be contained in the decomposition already, which will not always be the case. Thus, an introduction of a further set R becomes necessary, in which all degree 2 vertices are collected that have neighbours in cycles that have yet to be included in I. Condition (i) is expanded to allow F to contain isolated vertices in R and Condition (ii) allows path components to end in $A_p \cup R$.

Note that this does not restrict the behaviour of vertices in R: they can be isolated in F_i or part of a component, in which case they are allowed to be incident to an edge in M_i or C_i. Thus, the results presented here remain true. All lemmas in Section 4.2 continue to hold, which can be seen by simply suppressing all vertices in R, taking the decomposition acquired there, and subsequently subdividing again. If an edge in F_i or C_i is subdivided, both resulting edges are associated with the same set. Subdividing an edge of M_i lets us assign one of the two edges to M_i and the other to F_i. This ensures that M_i remains a matching and affects only one component K of F_i which obtains a new leaf in R. The validity of Condition (i) is unaffected by this change and the decomposition remains valid. Furthermore, Lemma 4.8 is also true in this setting, so these extended versions can be combined.

Finally, let us take a look at what parts of our proof would extend to tree-like graphs: graphs with a contraction graph that is a tree. Recall that for the proof of Theorem 4.2 we defined desirable I-decompositions $\mathcal{D}_I$ in Definition 4.17 as ones in which the boundary vertices neighbouring a common cycle exhibited a very restrictive behaviour. In the generalisation to tree-like graphs, we want the extension to a new cycle $\mathcal{C}_i$ to again have the same restricted behaviour for all the new neighbouring cycles. Thus, we call an (A_0, A_p, A_m, A_2)-decomposition *desirable* if its vertices in R, which are the boundary vertices with neighbours in newly connected cycles, satisfy an analogous condition.

With this adjustment, Lemma 4.8 can be extended to allow the extension of a desirable I-decomposition by a desirable (A_0, A_p, A_m, A_2)-decomposition to obtain a desirable J-decomposition. Most of the decompositions we discussed in Section 4.2 are desirable or can be made to be (sometimes requiring additional restrictions that are guaranteed by 3-connectivity and the restriction to tree-like graphs). The only exception to this it the decomposition from Lemma 4.15. This type of decomposition is problematic however, since it cannot be ensured to exist without resulting in new behaviours at the boundary. We illustrate this with the example in Figure 4.6, in which a part of a 3-connected tree-like graph is shown.

We focus on the cycle in the centre, in which, based on the edge colours, we want to find an $(\varnothing, \{x, y\}, \varnothing, A_2)$-decomposition, where A_2 consists only of the vertex z. In such a decomposition $\mathcal{D}_i \coloneqq (F_i, C_i, M_i)$, C_i uses an edge at x and y. Thus, the edge uv separates

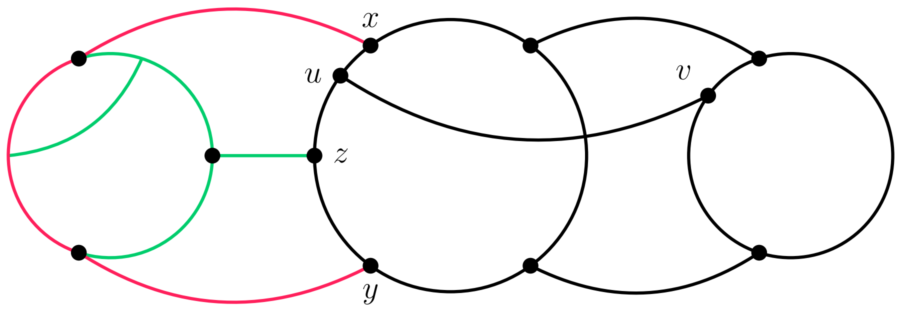


Figure 4.6.: The problematic decomposition for the tree-like case. When vertices in further cycles are present, it becomes impossible to ensure the existence of a path without allowing any new behaviours to occur at the boundary.

the tree we currently want to extend from the right cycle entirely. This means that uv must end up in the tree eventually, so we cannot include it in the cycle. Proceeding as we would in Lemma 4.15 and using the xy-path that avoids u and z causes us to isolate two vertices of R. This presents us with a new behaviour, namely the need to find a (A_0, A_p, A_m, A_2)-decomposition where A_0 contains two vertices. But not taking this path is not an option. At y, we cannot take the edge to z without cutting off the tree immediately, so the only alternative is to take the edge to the right cycle. But then we need to do this with the path at x as well since we need to close the cycle on the right. However, this isolates an edge of the centre cycle, which is not allowed either.

There are multiple ways to proceed at this point. Looking for further decompositions that allow us to cover more behaviours would be the primary goal, but with the rather strict requirements we currently impose, this is a difficult task. The natural alternative is to make the decompositions more powerful, for example by dropping the requirement that an I-decomposition is essentially a tree, up to isolated vertices. Once we allow larger components, we reach a state where any 3-decomposition can be represented. Additionally, we can be more lenient with the requirements on (A_0, A_p, A_m, A_2)-decompositions then, making them easier to obtain.

Of course this comes at a cost. Proving a statement analogous to Lemma 4.8 will be much harder. The core problem would be to keep track of which components are safe to combine and which are not, when not everything is directly attached to the main tree. To manage this, it appears necessary to either introduce restrictions after all, or to do a lot of tedious bookkeeping, resulting in a complicated generalisation of (A_0, A_p, A_m, A_2)-decompositions.

4.5. New graphs for which the 3-decomposition conjecture holds

We construct 3-connected star-like graphs, for which Theorem 4.2 shows there is a 3-decomposition. As the conjecture is already proved for graphs that are traceable, planar, claw-free, 3-connected and of tree-width at most 3, embeddable in the Torus or Klein bottle, or have a matching with a contraction graph of order at most 3, our goal is to find examples that have none of these properties. We present a construction closely based on [FS07], which we have modified in order to obtain graphs that are actually star-like.

4.19 Definition. A graph H is hypohamiltonian if it is not Hamiltonian, but $H - v$ is for all $v \in V$. ◁

(4.24) **4.20 Observation ([FS07, Lemma 3.1 (b)]).** For a hypohamiltonian graph H and $z \in H$, $H - z$ has no Hamiltonian path between two neighbours of z, as this could be extended to a Hamiltonian cycle of H. ◁

For the construction, let H_i, $i \in I := \{1, 2, 3\}$, be graphs that are 3-connected, cubic, non-planar, and hypohamiltonian. In order to see that such H_i exist and to obtain infinitely many examples, we exhibit an infinite family of candidates for the H_i. First, note that we can drop the 3-connectivity requirement.

4.21 Observation. Any hypohamiltonian graph is 3-connected. ◁

Proof. Let G be a hypohamiltonian graph and $v \in VG$. Since $G - v$ is Hamiltonian, it has at least three vertices, G has order at least 4, and $G - \{u, v\}$ is connected for any other vertex $u \in VG$. □

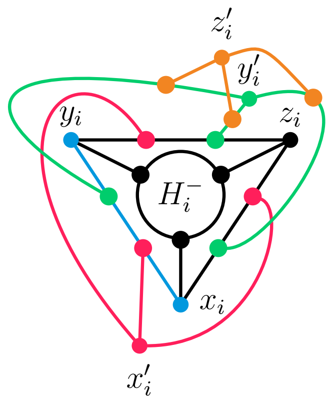
So we only need to find a family of cubic graphs that are hypohamiltonian and non-planar. The Petersen graph and the family of flower snarks satisfies both these properties.

4.22 Lemma ([HS93]). *The Petersen graph is cubic and hypohamiltonian. It also has a K_5 as a minor and is thus non-planar.* ◁

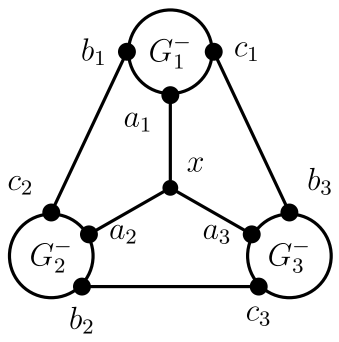
[CE83] [Isa75] **4.23 Lemma.** There exists an infinite family of cubic, non-planar, and hypohamiltonian graphs: the flower snarks. ◁

Proof. We refer to [CE83] for the definition of these cubic graphs since their description is lengthy and we do not need the details. The same paper shows that they are hypohamiltonian and [Isa75] proves non-planarity. □

Now on to the construction. For each $i \in I$ we pick some vertex $z_i \in H_i$ and set H_i^- to $H_i - z_i$. Our next goal is to expand H_i to the slightly larger 3-connected cubic graph G_i shown in Figure 4.7a. To ensure 3-connectivity we use Tutte's characterisation of 3-connected cubic graphs, by which we may iteratively subdivide two distinct edges and connect the resulting degree two vertices. We apply this first to two of the edges incident to z_i and call the subdivision vertices x_i and y_i. This step is drawn in blue in the figure. Next note that we can also subdivide three edges and connect the subdivision vertices to a single new vertex since this is just a sequence of two subdivision steps. We apply this



(a) The extension G_i of the graph H_i.



(b) The graph $G \coloneqq K_4[G_1, G_2, G_3]$.

Figure 4.7.: The star-like graph G that has none of the properties for which the conjecture is known. On the left, the construction preceding Lemma 4.24 is illustrated step by step, where the colour order is black, blue, red, green, and orange. On the right, the so constructed graphs G_i^- are combined to make the graph G.

to the edges of the triangle $x_i y_i z_i x_i$ and call the new vertex x_i', which is drawn in red. Next, we subdivide three of the newly split edges of the triangle that form a matching and connect them to the new vertex y_i', shown in green. Finally, we subdivide all edges incident to y_i' and connect them to z_i', which are the orange vertices and edges. Let the resulting graph be called G_i, denote $G_i - z_i'$ by G_i^-, and set $N_{G_i} z_i' \coloneqq \{a_i, b_i, c_i\}$.

Before we go on, we show some properties that the graphs G_i and G_i^- possess.

4.24 Lemma. The following properties hold: [2.13] [4.20] (4.25) (4.26)

(a) The graph G_i is 3-connected and cubic.
(b) The graph G_i^- has H_i as a minor.
(c) The graph G_i^- has no Hamiltonian path with both ends in $N_{G_i} z_i'$.
(d) For any $u \in G_i^-$, three $u\,N_{G_i} z_i'$-paths exist that are disjoint apart from u.
(e) For any pair of distinct vertices u, $v \in N_{G_i} z_i'$, the graph $G_i^- - H_i^-$ contains a Hamiltonian uv-path $P_i(u, v)$.
(f) The graph H_i^- is Hamiltonian. $\triangleleft$

Proof. Property (a) holds by the assumption on H_i and the construction, while the minor for Property (b) is obtained by taking the subgraph consisting only of the black edges in Figure 4.7a and suppressing the subdivision vertices. For Property (c) we notice that a Hamiltonian path in G_i^- that does not end in H_i^- must use exactly two of the three edges between $\{x_i, y_i, z_i\}$ and $N_{H_i} z_i$ (as they form a cut). Thus it would induce a Hamiltonian path in H_i^- with both ends in $N_{H_i} z_i$, which does not exist by Observation 4.20. The fourth part follows from the 3-connectivity of G, the paths for Property (e) are shown in Figure 4.8, and the last part holds because H_i is hypohamiltonian. $\square$

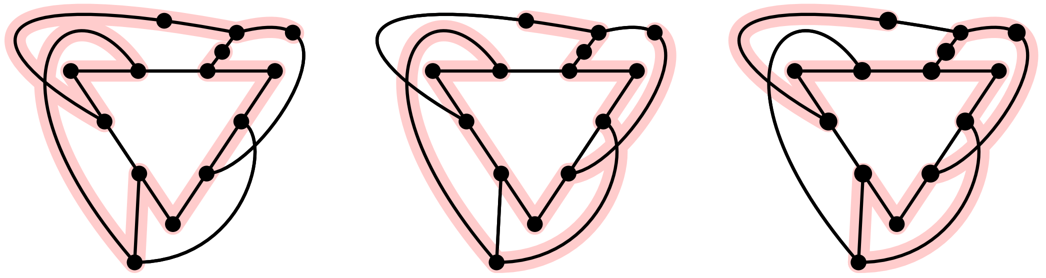


Figure 4.8.: The three Hamiltonian paths between pairs of neighbours of z_i in $G_i^- - H_i^-$. Note that these paths are symmetric, even if the drawing obscures it.

With this out of the way we now construct our desired graph $G \coloneqq K_4[G_1, G_2, G_3]$ as follows: let $G_4 \coloneqq (\{x\}, \varnothing)$ be a further vertex. Take $G_1^- \cup G_2^- \cup G_3^- \cup G_4$ and add the following six edges: xa_1, xa_2, xa_3, b_1c_2, b_2c_3, and b_3c_1. This graph is visualised in Figure 4.7b. We now prove that this does the trick.

[2.8] [4.24] **4.25 Theorem.** The graph G is cubic, non-traceable, star-like, and 3-connected. ◁

Proof. The graph is cubic by construction. To see that G is not traceable, we simply need to realise that a Hamiltonian path P would yield a graph G_i^- in which no end of the path resides. This means that P restricted to G_i^- would necessarily be a Hamiltonian path with ends in $N_{G_i}\, z_i'$, which does not exist by Lemma 4.24 (c).

In order to prove that G is star-like, we need to specify the cycles. We take a Hamiltonian cycle in each H_i^- for $i \in I$, which exist by Lemma 4.24 (f). In addition, we use the paths given by Lemma 4.24 (e) to obtain a Hamiltonian cycle C in $G - \bigcup_{i \in I} H_i^-$, namely

$$C \coloneqq xa_1P_1(a_1, b_1)b_1c_2P_2(c_2, b_2)b_2c_3P_3(c_3, a_3)a_3x.$$

This yields a decomposition where the contraction graph is a star in which the centre corresponds to C and the three tips to the cycles in the H_i^-.

Finally, we are left with the proof that G is 3-connected. To this end, we use Menger's theorem to find three internally vertex-disjoint paths between every pair of distinct vertices u, $v \in G$. Assume $u \in G_1^-$, without loss of generality. If $v \in G_1^-$, then $G/(VG \setminus VG_1^-) \cong G_1$ contains three uv-paths by the 3-connectivity of G_1. If one such path uses the super node, we replace it by a path through the subgraph we contracted. So assume that $v \notin G_1^-$. If $v \in G_2^-$ (the case that $v \in G_3^-$ is analogous), we use Lemma 4.24 (d) to reduce the problem to finding three disjoint $N_{G_1}\, z_1'\, N_{G_2}\, z_2'$-paths. But these exist, just take

$$P_1 \coloneqq a_1xa_2, \quad P_2 \coloneqq b_1c_2, \quad P_3 \coloneqq c_1b_3P_3(b_3, c_3)c_3b_2.$$

Similarly we deal with the case that $v = x$, where it suffices to find three $x\, Nz_1'$-paths that are disjoint aside from x. This time we use

$$P_1 \coloneqq xa_1, \quad P_2 \coloneqq xP_2a_2(a_2, c_2)c_2b_1, \quad P_3 \coloneqq xa_3P_3(a_3, b_3)b_3c_1.$$

This completes the proof. □

To see that the star-like graph G constructed this way fulfils none of the requirements necessary to apply one of the previously existing results, we now only need to verify that its tree-width is greater than 3, it has no contraction graph of order at most 3, and it cannot be embedded in the Torus or the Klein bottle. But as soon as one of the non-planar graphs contains a K_5-minor, the tree-width is at least 4. So, by assuming that H_1 is the Petersen graph, for example, we can ensure that the graphs constructed here have a sufficiently large tree-width.

To see that they do not admit a matching with a contraction graph of order 3, we show that G has no 2-regular subgraph consisting of 3 cycles. Let $\mathcal{C}$ be any 2-regular subgraph of G, then $\mathcal{C}$ contains a cycle completely contained in G_i^- for all $i \in \{1, 2, 3\}$. This is true as any cycle in $\mathcal{C}$ is either completely contained in G_i^-, disjoint from it, or consists of a path in it. But there can be at most one cycle that restricts to a path because $E(VG_i^-)$ contains only three edges. Since this path is not Hamiltonian by Lemma 4.24 (c), the graph G_i^- contains at least one cycle of $\mathcal{C}$. Thus $\mathcal{C}$ is made up of at least four cycles, one in each G_i^- and one containing x.

To complete the proof that our examples fall in none of the classes covered previously, we now only need to check that G cannot be embedded in the Torus or the Klein bottle. The *(non-orientable) genus* of a graph H is the minimal n such that H can be embedded in a surface of (non-orientable) genus n. In particular, since the Torus has genus 1 and the Klein bottle has non-orientable genus 2, it suffices to show that G has genus and non-orientable genus at least 3, which we now prove to finish this section.

4.26 Theorem. The graph G has genus and non-orientable genus at least 3. $\triangleleft$ [RS90] [Bat+62] [SB77] [4.24]

Proof. We denote the genus of a graph H by $\gamma\, H$ and its non-orientable genus by $\tilde{\gamma}\, H$. We first note that it suffices to find a minor H of G with $\gamma\, H \geq 3$ to get $\gamma\, G \geq 3$ since having genus at most 2 is characterised by forbidden minors [RS90]. The same holds for the non-orientable genus.

To obtain this minor, we use that the H_i are non-planar, meaning they each have a K_5- or $K_{3,3}$-minor H_i' for $i \in \{1, 2, 3\}$. By Lemma 4.24 (b), the graph G_i^- contains an H_i'-minor. As the G_i^- are connected, we may assume that all their vertices are contained in some super node of H_i'. By taking these minors and removing the edges b_1c_2, b_2c_3, and b_3c_1 from G, we obtain the minor $H_1' \cup H_2' \cup H_3' \cup G_4 + E'$ where E' contains the three edges from x to the super nodes containing the vertices a_1, a_2, and a_3. By contracting the edges in E' as well, we obtain a connected graph H with the three blocks H_1', H_2', and H_3'.

The genus of the H_i' is 1 and so is the non-orientable genus, as they are non-planar but can be embedded in the Torus or the projective plane, respectively. Consequently, we get $\gamma\, H = \sum_{i=1}^{3} \gamma\, H_i' = 3$, because the genus of a graph is the sum of the genuses of its blocks by [Bat+62]. We also have that H is not orientably simple by [SB77, Theorem 1], which states that a connected graph is orientably simple if and only if all of its blocks are, as K_5 and $K_{3,3}$ are not. Thus, the non-orientable genus can be computed as specified in Corollary 3 of the same paper, giving us

$$\begin{aligned}\mu\, H_i' &= \max\{2 - 2\,\gamma\, H_i', 2 - \tilde{\gamma}\, H_i'\} = \max\{0, 1\} = 1, \text{ and}\\ \tilde{\gamma}\, H &= 2 \cdot 3 - \sum_{i=1}^{3} \mu\, H_i' = 6 - 3 = 3.\end{aligned}$$

As a result, G has a minor of genus and non-orientable genus at least 3, proving the claim. $\square$

Chapter 5.

Reducible Configurations and Minimum Counterexamples

A graph that does not occur as a subgraph in (3-connected) minimum counterexamples to the 3-decomposition conjecture is a reducible configuration. We prove that the following graphs are reducible configurations: the triangle, the $K_{2,3}$, the Petersen graph with one vertex removed, the claw-square, the twin-house, and the domino.
As an application, we show that all 3-connected graphs of path-width at most 4 satisfy the 3-decomposition conjecture and that a 3-connected minimum counterexample to the conjecture is triangle-free, all cycles of length at most 6 are induced, and every edge is in the centre of an induced P_6.

In this chapter, we determine small graphs that do not occur as subgraphs in minimum counterexamples to the 3-decomposition conjecture (minimum with respect to number of vertices) and use these to prove properties that such counterexamples must possess.

A subcubic graph S is a *reducible configuration* if no minimum 3-connected counterexample to the 3-decomposition conjecture contains S as a subgraph. We show that the following six configurations, shown in Figure 5.1, are reducible: the triangle, the $K_{2,3}$, the Pet$^-$, the *claw-square*, the *twin-house*, and the *domino*.

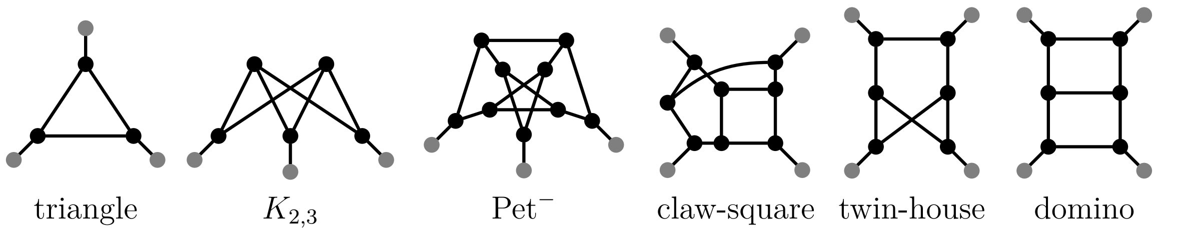


Figure 5.1.: Reducible configurations for the 3-decomposition conjecture. The names refer to the graphs induced by the black vertices and the grey vertices need not be distinct.

Roughly speaking, the reducibility proofs proceed as follows: given a cubic graph G containing the reducible configuration R, replace R by some smaller graph S. This yields a new graph H containing S and we show that every local behaviour that a 3-decomposition might exhibit on S can be extended to R. Consequently, these proofs are constructive: we can obtain a 3-decomposition of a graph G containing R from a 3-decomposition of the graph H. As such, these proofs fall into the second category we described in Section 3.2.

We formalise the concepts of extensions and reductions, which are fundamental for these proofs, in Section 5.1, where we also show initial properties concerning these. Furthermore, we give proofs of reducibility of the six configurations shown in Figure 5.1 in Section 5.2 (and partly in Appendix A).

In Section 5.3, we use the new reducible configurations to extend the list of graph classes which satisfy the 3-decomposition conjecture to include the 3-connected cubic graphs of path-width at most 4. Due to the constructive nature of the reducibility proofs, this yields a method of obtaining decompositions for these graphs, too.

As a further use of our reducible configurations, we gain new insights into the structure of (potential) 3-connected minimum counterexamples G to the 3-decomposition conjecture. Not containing any of the graphs in Figure 5.1 as a subgraph lets us show that G has girth at least 4, every cycle of length 4, 5, or 6 in G is induced, and every edge of G is the centre edge of an induced P_6.

The results covered in this chapter are joint work with Irene Heinrich and they, like Sections 3.1 and 3.2 in Chapter 3, are part of [BH21].

5.1. Extensions and reductions

Defining extensions and reductions. In the following, we formalise the replacement of an induced subgraph R of a cubic graph G by some other subcubic graph S. Intuitively, this means that we add the graph S to $G - VR$ and connect the vertices of S 'appropriately'. In order to define what 'appropriately' means, we augment R to a graph X by adding (new) leaves to all vertices of R of degree less than 3 such that all vertices of R are of degree 3 in X. We do the same for S and use these additional vertices to specify the edges between S and $G - VR$.

This leads us to the definition of a *template graph*.

5.1 Definition. A *template graph* is a graph X whose vertex set is partitioned into a set of *inner vertices* IX and a set of *outer vertices* $OX = \{v_1, \dots, v_k\}$. All inner vertices have degree 3 and all outer vertices have degree 1. We call $X - OX = X[IX]$ the *core of* X and denote it by cX. For a subcubic graph R, the unique template graph X (up to labelling the outer vertices) with R as its core is called the *template graph of* R. $\triangleleft$

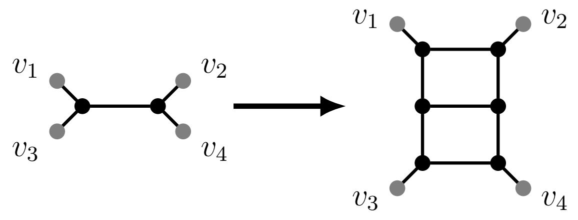


Figure 5.2.: Template graphs and transformations. This example illustrates the extension where $c\,X$ is the K_2 and $c\,Y$ is the domino. The outer vertices of the template graphs are grey.

The template graph of the domino can be seen on the right of Figure 5.2, where the outer vertices are grey.

We can now define the replacements we need: let R, S be subcubic graphs and let X, Y be their template graphs, for which we require that $OX = OY$. Furthermore, let G be a cubic graph containing an induced subgraph R' isomorphic to R, that is, $\varphi\,R = R'$ for some isomorphism φ. Since G is cubic, any vertex $\varphi\,v \in R'$ has exactly as many incident edges to vertices in $G - VR'$ as v has neighbours in OX. Thus, we can identify each neighbour w of v that is an outer vertex with a neighbour $\psi\,w$ of $\varphi\,v$ that is not in R'.

The graph

$$\begin{aligned} G_\varphi[X \to Y] :=& (G - VR') \cup S \qquad + \{v\,\psi\,w\colon\ vw \in Y,\, w \in OY\} \\ =& (G - \varphi(I\,X)) \cup c\,Y + \{v\,\psi\,w\colon\ vw \in Y,\, w \in OY\} \end{aligned}$$

is the (X, Y, φ)-*transformation* of G. Note that the resulting graph does not depend on the choice of ψ. When specifying transformations, we only care about the template graphs since we do not yet have the graphs they are embedded in.

5.2 Definition. Let X and Y be template graphs with $OX = OY$. An (X, Y)-*transformation* of G is a graph $G[X \to Y] := G_\varphi[X \to Y]$ for some isomorphism $\varphi\colon\ c\,X \to R'$ and induced subgraph R' of G. Moreover, the pair (X, Y) is called an (X, Y)-*transformation*. Transformations are called *extensions* if $|VX| < |VY|$ and *reductions* if $|VX| > |VY|$.◁

We depict (X, Y)-transformations as seen in Figure 5.2. Instead of formally writing down vertex and edge sets, we use such illustrations to describe the transformations since they are more convenient and easier to understand. We mostly omit the labels of the outer vertices; their positioning shows which of these coincide.

This seems like a good opportunity to note that we have used such extensions and reductions before. The operations from Wormald's characterisation of cubic graphs in Figure 2.2, as well as our operations for the HIST-extension conjecture in Figure 3.1 actually fall perfectly into this framework. From these, we can recall that the outer vertices of the template graph do not necessarily correspond to distinct vertices of the graph G containing the core and, as a result, G need not have a subgraph isomorphic to X, just to $c\,X$. Additionally, the graphs $G[X \to Y]$ are not necessarily simple, even if

G, X, and Y are. Therefore, technically, we should have used the word 'multigraph' for the graph $G[X \to Y]$ above. However, we only intend to apply transformations which yield simple graphs, so this mainly serves as a reminder that this is a property we need to verify. A sufficient condition for ensuring simplicity is that no vertex in the core of Y has multiple adjacent outer vertices, or equivalently, that the minimum degree in the core of Y is 2. Another one is that $G[X \to Y]$ is 3-connected, since all 3-connected cubic graphs are simple.

Compatible extensions and reducible configurations. We can now provide the definitions we need to actually show that certain graphs do not occur as subgraphs of minimum counterexamples.

5.3 Definition. A *reducible configuration* is a (subcubic) graph R that is not part of a 3-connected minimum counterexample to the 3-decomposition conjecture. In this case we also say that R is *reducible*. ◁

Our goal is to describe (X, Y)-extensions that allow us to extend 3-decompositions. For ease of use, we provide the following definition as well.

5.4 Definition. An (X, Y)-extension is *3-compatible* if, for every cubic graph G with a 3-decomposition and for every extension $H \coloneqq G[X \to Y]$ of G, the graph H also has a 3-decomposition. ◁

In fact, in our proofs that certain transformations are 3-compatible we show how to construct a 3-decomposition of H from one of G.

Finding a 3-compatible extension (X, Y) is very helpful for proving that the core of Y is reducible. The following lemma exhibits a property that implies reducibility if it is satisfied by such an extension.

(5.10) (5.13) (5.16) (5.18) **5.5 Lemma.** Let (X, Y) be a 3-compatible extension. The core cY is reducible if for every 3-connected cubic graph H containing (a subgraph isomorphic to) cY

- H has a 3-decomposition, or,
- cY is induced and the reduction $G \coloneqq H[Y \to X]$ is a 3-connected graph. ◁

Proof. Suppose H is a minimum counterexample containing cY as an induced subgraph and let $G \coloneqq H[Y \to X]$ be a 3-connected graph. Since H is a minimum counterexample, G has a 3-decomposition and, because (X, Y) is 3-compatible, $H = G[X \to Y]$ also has a 3-decomposition, which is a contradiction. □

Note that this proof would have worked just as well if we had required 3-compatibility to yield extensions only for 3-connected graphs. We opted for this more inclusive definition since it strengthens the results for the extensions we prove to be 3-compatible.

Whenever possible, we want to apply this lemma. Therefore, in order to prove that a certain configuration R is reducible, we proceed as follows:

(1) Prove a certain (X, Y)-extension to be 3-compatible where $R = cY$.
(2) Check that in any 3-connected cubic graph H containing a subgraph R' isomorphic to R, R' is actually induced and a (Y, X)-reduction of H yields another 3-connected graph.

The first option of Lemma 5.5 lets us assume that the graph obtained from the reduction is still sufficiently large by excluding some small graphs that already have 3-decompositions. Furthermore, the procedure described here is actually entirely constructive in the sense that, given a graph H, we can perform reductions using the reducible structures as long as they are present. Let G be the resulting graph. If we can find a 3-decomposition for this smaller graph G, then we can undo the reductions step by step and extend our decompositions to the larger graphs, finally obtaining one for H.

To use this procedure, we regularly need to check whether subgraphs are induced and whether reductions yield 3-connected graphs. To facilitate this, we end this section with two lemmas that are helpful in this respect. Their proofs use Menger's theorem when verifying 3-connectivity, which requires simple graphs to be applicable.

5.6 Lemma. Let H be a 3-connected cubic graph that contains the core of a template graph Y with $|OY| = 3$ and let X be the template graph of a single vertex. Then cY is induced and, if $H - IY$ consists of more than one vertex, then the reduction $G \coloneqq H[Y \to X]$ is 3-connected. $\triangleleft$ [2.8] (5.10) (5.13) (5.16)

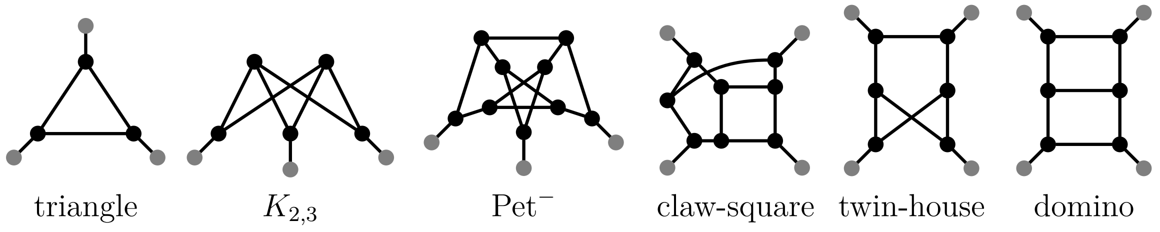
Proof. The 3-connectivity of H implies that there are at least three edges joining $H - IY$ with cY. Since Y has three outer vertices, there are exactly three such edges and cY is induced. Furthermore, if $H - IY$ has more than one vertex, then the three neighbours of cY are distinct: if all three of the neighbours coincide, then H is not connected, which is a contradiction. If exactly two of them coincide, then we find a 2-edge cut in H, which contradicts to H being 3-connected. Consequently, G is simple in this case.

To see that G is 3-connected, consider two distinct vertices u and v of G. If neither u nor v is the vertex in cX, then they are in H and we obtain three internally vertex-disjoint paths linking them in H. At most one of these paths can use vertices of cY since $|E(VH \setminus IY, IY)| = 3$. Thus, by potentially replacing the path segment through cY by the vertex in cX, we obtain three paths in G.

If u is the vertex in cX, then let $u' \in IY$. We obtain three vertex-disjoint $u'v$-paths in H, each of which uses one of the edges from cY to the rest of H. Thus we can replace these initial path segments by the edges incident to u in G and obtain the required paths there. $\square$

5.7 Lemma. Let H be a 3-connected cubic graph that contains the core of a template graph Y with $|OY| = 4$ and $|IY| \geq 5$. Moreover, let X be the template graph of the square. Then cY is induced and the reduction $G \coloneqq H[Y \to X]$ is 3-connected. $\triangleleft$ [2.8] (5.13) (5.16) (5.18)

Proof. The 3-connectivity of H implies that there are at least three edges between cY and $H - IY$. Since Y only has four outer vertices, cY is induced in H and G is simple because the square has minimum degree 2.



A repetition of Figure 5.1.

To verify the 3-connectivity of G, we consider two vertices u and v of G. If both are not vertices of the square, then we obtain three internally vertex-disjoint paths linking them in H. At most two of these paths use vertices of cY since Y only has four outer vertices and all other paths exist in G. If a single path passes cY, then it can be replaced by one in the square and so can two paths unless they both need to connect non-adjacent vertices of the square. In this case, we can cross the paths by pairing the start of one with the end of the other and vice versa. In all cases we obtain the three desired disjoint paths.

Next, we consider the case that exactly one of the two vertices, say v, is in the square. Let v' be the vertex in cY that is adjacent to the same outer vertex in Y as v is in X. By 3-connectivity of H, we obtain three paths between u and v' in H. Of these, one path only meets cY in v' while the remaining two start with a path segment in cY. However, the square contains such path segments as well, so we can replace the initial part of the paths by ones in the square to get three paths in G. We remark that v' may have degree 1 in cY, in which case two paths in H meet cY only in v'. This causes no problems, however, it just means that one of the two remaining paths segments we needed above is very short.

Finally, if both vertices are in the square, replace u by its neighbour x outside of the square. By the previous case, we obtain a path from x to v that only meets the square in v. We extend this path to u and take the two disjoint paths the square provides. Thus G is also 3-connected. $\square$

5.2. New reducible configurations

In this section, we prove the following theorem:

(5.22) **5.8 Theorem.** The six graphs in Figure 5.1 on Page 67 are reducible. $\triangleleft$

For convenience, we repeat Figure 5.1 on this page.

The first two were shown to be reducible in [Hei19, Hei20] but we include them for completeness. (Actually, [Bac15] also contains the triangle.) All but one of these proofs adhere to the general proof structure described in the previous section and we illustrate it using the triangle. For the remaining graphs we focus on the more challenging cases

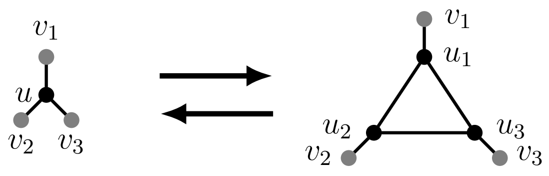


Figure 5.3.: Transforming a vertex into a triangle.

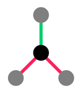


Figure 5.4.: Possible behaviours of a 3-decomposition at a single vertex.

and refer to Appendix A for the others. The first step for the triangle is to find an (X, Y)-extension that is 3-compatible and where $c\,Y$ is the triangle. For this, we choose X such that $c\,X$ is a single vertex.

5.9 Lemma. The extension shown in Figure 5.3 is 3-compatible. ◁

Proof. Let G be a cubic graph with a 3-decomposition (T, C, M) and let H be an (X, Y)-extension of G, where the extension is the one from Figure 5.3. We wish to extend (T, C, M) to H. To this end, we start by determining the possible behaviours at u. As T is a spanning tree, at least one of the edges incident to u is in T. If it is exactly one, then the other two are part of a cycle. If there are exactly two, then the missing edge is in the matching and, otherwise, there are three. These options are shown in Figure 5.4, up to rotational symmetry. Recall that edges of T are green, those of C are red, and those in M are blue.

We now need to turn the decomposition (T, C, M) of G into one of H, which we do by distinguishing between the possible behaviours described above. The extensions obtained are shown in Figure 5.5. We describe how these depictions are to be read, using the first case as an example. The remaining ones should then be self-explanatory.

If we only have a single T-edge, say uv_1, incident to u, we make the following extension: we begin by setting $T' := T - I\,X + I\,Y$, $C' := C - I\,X + I\,Y$, and $M' := M - E\,X$. These are a spanning forest, a disjoint union of cycles together with a v_2v_3-path, and a matching. Technically, C' also contains u_1 as an isolated vertex, which we drop. The forest has four components, three of them being the vertices $I\,Y$ of the triangle.

By adding the edges u_1v_1, u_1u_2, and u_1u_3 to T' we obtain a spanning tree of G and adding the remaining edges incident to u_2 and u_3 to C' yields a disjoint union of cycles. As a result (T', C', M') is a 3-decomposition of G.

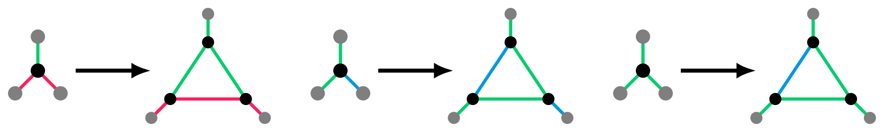
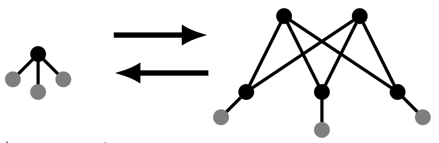
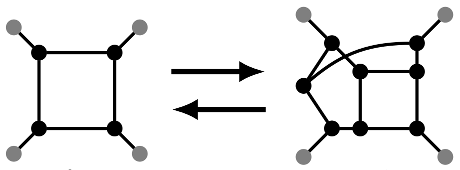


Figure 5.5.: Extending a 3-decomposition from a vertex to a triangle.


(a) Transforming a vertex into a $K_{2,3}$.


(b) Transforming a square into a claw-square.

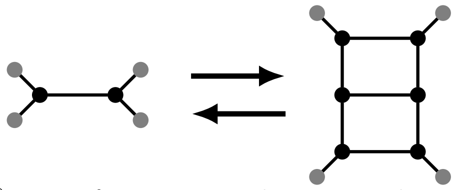

(c) Transforming an edge into a domino.

Figure 5.6.: Three straightforward 3-compatible extensions.

The figure illustrates how to extend the decomposition for the remaining two behaviour types, giving us a 3-decomposition of H in all cases, and thus showing that the extension is 3-compatible. □

We can now complete the second step to obtain the reducibility of the triangle.

[5.5] **5.10 Corollary.** The triangle is reducible. ◁

[5.6] [6.8] *Proof.* Here we consider the inverse (Y, X)-reduction also shown in Figure 5.3. Let H be a 3-connected cubic graph containing a triangle $u_1u_2u_3u_1$. By Lemma 5.5 it suffices to prove that the triangle is induced and that a (Y, X)-reduction $G = H[Y \to X]$ is 3-connected. In fact, we may assume that $H \neq K_4$ as the K_4 has a 3-decomposition (for example, by Theorem 6.8). However, all required properties follow directly from Lemma 5.6. □

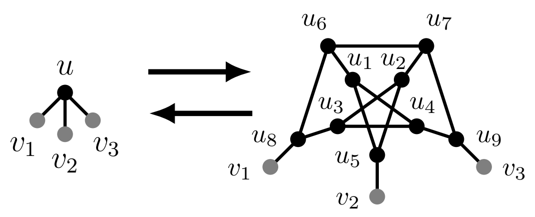
Now that we have illustrated the general concept, let us fix some notation for the remainder of this section.

5.11 Convention. Whenever we consider a transformation, we denote the smaller graph by X and the larger one by Y, unless explicitly stated otherwise. Furthermore, when we prove that an extension is 3-compatible, we write G for the smaller graph containing the core of X and $H := G[X \to Y]$ is an (X, Y)-extension of G. The 3-decomposition of G is (T, C, M) and we want to construct a 3-decomposition (T', C', M') for H. ◁

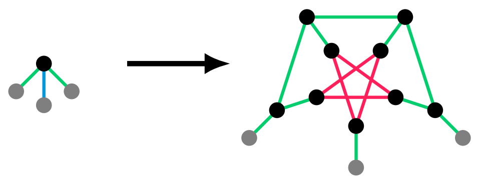
Next, we look at three further examples of 3-compatible extensions that are straightforward. As such, the proof of the following lemma can be found in Appendix A (Figures A.1, A.4, and A.8 on Pages 173, 174, and 177).

[A.1] **5.12 Lemma.** The extensions shown in Figures 5.6a to 5.6c are 3-compatible. ◁

[A.4] [A.8] (5.18) By completing the second step for the $K_{2,3}$ and the claw-square, we get reducibility for the next two configurations.



(a) Transforming a vertex into a Pet$^-$.



(b) Extending a local behaviour.

Figure 5.7.: A transformation for the Pet$^-$.

5.13 Corollary. The $K_{2,3}$ and the claw-square are reducible. ◁ [5.5] [5.6] [5.7] [6.8]

Proof. Regard the (Y, X)-reductions in Figures 5.6a and 5.6b. By Theorem 6.8 and Lemma 5.5, it suffices to prove that, given a 3-connected cubic graph H with $|H| > 6$ containing the core of Y, cY is induced and the reduction $G := H[Y \to X]$ is simple and 3-connected. This holds for the $K_{2,3}$ by Lemma 5.6 (since we excluded the $K_{3,3}$ by requiring $|H| > 6$) and for the claw-square by Lemma 5.7. □

We could not include the domino here since reducing it to a single edge could harm 3-connectivity. We postpone completing the domino until the end of this section since its proof is the most complex and having seen more examples should make it easier to follow. Therefore, we continue with the Pet$^-$ and the twin-house.

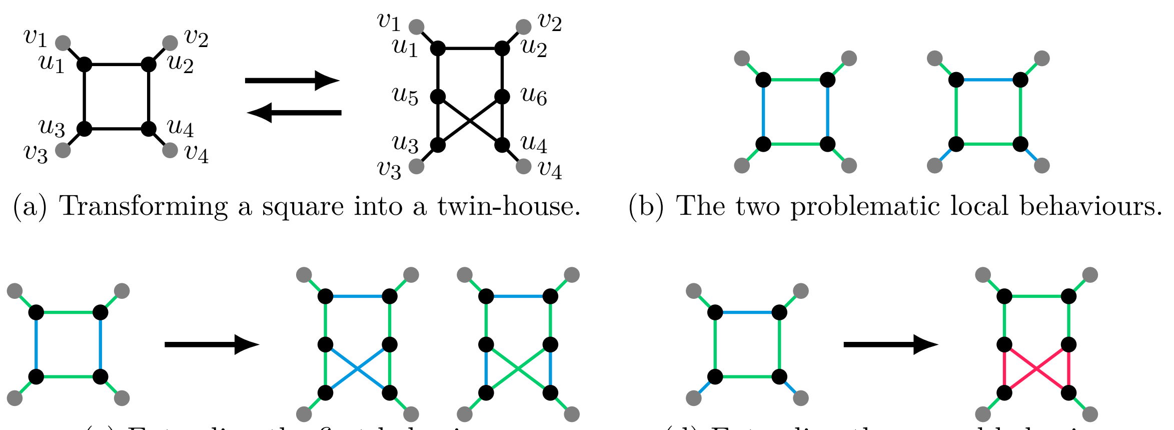
5.14 Lemma. The extension shown in Figure 5.7a is 3-compatible. ◁ [A.2]

Proof. We note that the Pet$^-$, like the triangle, is symmetric. Thus, we only need to check the three possible behaviours seen in Figure 5.4 on Page 73. Of these, the first and the last are straightforward and can be found in Appendix A (Figure A.2 on Page 173). This leaves the second one, where an M-edge is at the single vertex, and we extend the decomposition as shown in Figure 5.7b.

Note that the removal of the edges uv_1 and uv_3 from T (where the vertex names are taken from Figure 5.7a) creates three connected components, one containing only u and one with v_2 and v_3, respectively. By adding all edges of H (including v_2u_5) except those on the cycle $u_1u_4u_3u_2u_5u_1$ to this forest we obtain a spanning tree T' and adding the excluded cycle to C yields C'. Now $(T', C', M \setminus \{v_2u_5\})$ is a 3-decomposition of H. □

The twin-house is reduced to the square, which has significantly more possible local behaviours than the K_1 or the K_2. In order to reduce the number of cases that need to be extended, we show that some can be transformed into one another. Again, this is found in Appendix A (resulting in the behaviours shown in Figure A.6 on Page 175) and we only show how to deal with the problematic cases in the proofs here.

5.15 Lemma. The extension shown in Figure 5.8a is 3-compatible. ◁ [A.9]



(a) Transforming a square into a twin-house. (b) The two problematic local behaviours.

(c) Extending the first behaviour. (d) Extending the second behaviour.

Figure 5.8.: A transformation for the twin-house.

Proof. For this extension, we consider the local behaviours in Figure 5.8b and refer to Appendix A (Figure A.9 on Page 178) for the remaining ones.

For the first forest, we note that the removal of the edge u_1u_2 (using the terminology from Figure 5.8a) from T creates two connected components, one containing u_1 and one containing u_2. By symmetry we may assume that the component containing u_2 also contains u_3 and u_4. Once we remove the edge u_3u_4 as well, we end up with three components. One of these three contains u_1 whereas the other two either contain u_2, u_3, and u_4 or u_2, u_4, and u_3. The assignments we now make are illustrated in Figure 5.8c. By replacing the square by the twin-house and adding the edges

$$u_1u_5,\ u_5u_3,\ u_2u_6,\ u_6u_4 \text{ or } u_1u_5,\ u_5u_4,\ u_2u_6,\ u_6u_3$$

to the forest, we obtain a spanning tree T' of H. The missing edges are

$$u_1u_2,\ u_5u_4,\ u_6u_3 \text{ or } u_1u_2,\ u_5u_3,\ u_6u_4$$

which form a matching M' together with $M \setminus \{u_1u_3, u_2u_4\}$. Thus (T', C, M') is a 3-decomposition of the extended graph.

For the second forest, removing the three edges u_1u_3, u_3u_4, u_4u_2 from T yields a spanning forest with four components, two of which are the isolated vertices u_3, u_4. By replacing the square by the twin-house and adding the edges u_1u_2, u_1u_5, u_2u_6 to the forest, we obtain a spanning tree of $G - \{u_3, u_4\}$. We can connect these last two vertices by using their incident edges u_3v_3, u_4v_4 in M to obtain a spanning tree T' of G as shown in Figure 5.8d. We then take M' to be the set $M \setminus \{u_1u_2, u_3v_3, u_4v_4\}$, which is still a matching. The remaining edges in the twin-house form a C_4 which we add to C to obtain C'. Then (T', C', M') is a 3-decomposition of G. □

[5.5] **5.16 Corollary.** The Pet$^-$ and the twin-house are reducible. ◁
[5.6]
[5.7] *Proof.* The (Y, X)-reductions in Figures 5.7a and 5.8a yield 3-connected graphs by Lemmas 5.6 and 5.7. By Lemma 5.5 both graphs are reducible. □

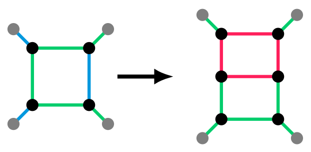


Figure 5.9.: The bad local behaviour for the domino.

This just leaves the domino, whose reducibility proof is the most complex. If we regard the reduction to the square again, then all but one case is straightforward, which is the one shown on the left in Figure 5.9. However, this case is difficult to remedy and requires us to know more about the structure of the graph G in which this occurs. To obtain this information, we first attempt to reduce the domino to a single edge, for which the corresponding extension is 3-compatible by Lemma 5.12. If this reduction yields a 3-connected graph, then we are done. Otherwise, we have acquired enough information to deal with the only problematic case occurring in the reduction to the square.

5.17 Lemma. Let (X, Y) be the extension and (X, Z) be the reduction of the square shown in Figure 5.10. Moreover, let G be a 3-connected graph containing the core of X such that its (X, Z)-reduction is not 3-connected. If G has a 3-decomposition, then the (X, Y)-extension of G also has a 3-decomposition. [2.9] [A.10] (5.18)

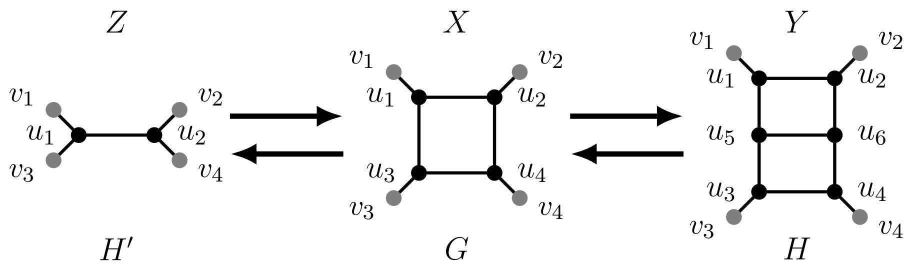


Figure 5.10.: Transforming a square into a domino or an edge.

Proof. Let G be a graph as in the claim and let (T, C, M) be a 3-decomposition of G. Furthermore, let H be the extension $G[X \to Y]$ and H' the reduction $G[X \to Z]$. If any local behaviour other than the one in Figure 5.9 occurs, then we can extend the decomposition of G to H, see Figure A.10. Thus, we restrict ourselves to the forest T_X with $E\,T_X := \{v_2u_2, u_2u_1, u_1u_3, u_3u_4, u_4v_4\}$.

If we remove the three edges of T_X in the square, then we end up with four components: the vertices u_1, u_3 and components $H_2 \ni u_2$, $H_4 \ni u_4$. If $v_3 \notin VH_4$, then we take all edges of Y except those on the cycle $u_1u_2u_6u_5u_1$ to be part of the tree component, as shown in Figure 5.9. This results in a spanning tree since each of the five edges added connects different components and its complement is the matching $M \setminus \{u_1v_1, u_3v_3, u_2u_4\}$ and the cycles in C together with $u_1u_2u_6u_5u_1$. This lets us assume that $v_3 \in VH_4$. Note that a symmetric assignment (leaving the cycle $u_5u_6u_4u_3u_5$) lets us assume that $v_1 \in VH_2$. Now let C_1 and C_3 be the unique cycles in $T + u_1v_1$ and $T + u_3v_3$, which are disjoint in the situation we are in now. We illustrate this in Figure 5.11.

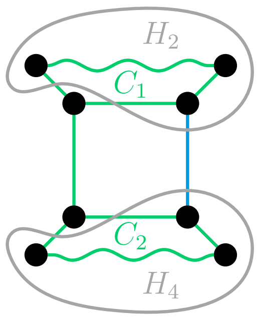


Figure 5.11.: The bad case in the proof on Lemma 5.17. In it, C_1 and C_2 are two disjoint cycles in the graph $T + \{u_1v_1, u_3v_3\}$, whose edges are coloured green.

We claim this shows that H' is 3-connected, contrary to the assumption of the lemma. The graph H' is simple since the neighbours of u_1 in H' are distinct, they are in different components of $T - u_1u_3$. The same holds for the neighbours of u_2. If H' were not 3-connected, then, by the max-flow min-cut theorem, a partition (A', B') of VH' exists such that $E_{H'}(A', B')$ contains at most two edges. We now consider the partition (A, B) of VG obtained by adding u_3 to the set containing u_1 and u_4 to the set containing u_2. Next, we compare the sets $E_{H'}(A', B')$ and $E_H(A, B)$. An edge $xy \in E_{H'}(A', B')$ is also in $E_H(A, B)$ if neither of its ends is u_1 or u_2. If just one of them is, say x, then it corresponds to a unique edge in $E_H(A, B)$, where x is potentially replaced by the vertex u_3 or u_4. Only the edge u_1u_2 corresponds to multiple edges in $E_H(A, B)$, namely to the edges u_1u_2 and u_3u_4.

Since G is 3-connected, we can conclude that $|E_{H'}(A', B')| = 2$, $|E_H(A, B)| = 3$, and $u_1u_2 \in E_{H'}(A', B')$, u_1u_2, $u_3u_4 \in E_H(A, B)$. This lets us assume that u_1, $u_3 \in A$ and u_2, $u_4 \in B$. Both cycles C_1 and C_3 contain an element of A and one of B, resulting in a cut of these cycles. These cuts have at least two edges crossing, giving us a total of at least four since the cycles are disjoint, which is a contradiction. □

[5.5] **5.18 Corollary.** The domino is reducible. ◁
[5.7]
[5.12] *Proof.* In light of the new situation, we cannot just apply Lemma 5.5. Let H be a
[5.17] minimum counterexample containing the domino, that is, H is a 3-connected cubic graph
[6.8] without a 3-decomposition. Since all cubic graphs on six vertices have a 3-decomposition by Theorem 6.8, we may assume that H has more vertices than the ones in the domino. Also, by Lemma 5.7, we may assume that the domino is induced.

If we can replace the domino by a single edge without harming 3-connectivity as shown in Figure 5.6c, then we obtain a 3-decomposition of H by Lemma 5.12, a contradiction.

So we may assume that this is not the case. By replacing the domino by a square as shown in Figure 5.10 on Page 77, we obtain a new graph G which is 3-connected by Lemma 5.7. By minimality of H, G has a 3-decomposition and satisfies the premise of Lemma 5.17 (since the reduction of the square to an edge corresponds to the reduction of the domino in H to an edge directly). This gives us a contradiction. □

5.3. Properties of minimum counterexamples

In this section, we prove properties of minimum counterexamples to the 3-decomposition conjecture (under the assumption that such a counterexample exists). To do so, we exploit our new reducible configurations.

We prove that all 3-connected cubic graphs of path-width at most 4 have a 3-decomposition. Therefore minimum 3-connected counterexamples have path-width at least 5. This proof makes use of the reducible configurations we determined in Section 5.2 by showing that the restriction of the path-width causes one of them to appear. Hence, the proof follows an analogous structure to the one for tree-width 3 in [Hei20], which shows the following (slightly modified) lemma:

5.19 Lemma ([Hei20, Lemma 8.9]). *Every cubic graph of tree-width at most 3 contains at least one isomorphic copy of the following graphs as a subgraph: the triangle, the $K_{2,3}$, or the domino.* $\triangleleft$ (5.20)

Actually, [Hei20] proved this result for a larger graph than the domino (which behaves more nicely than the domino with respect to 3-decompositions) and requires the graphs to be 3-connected, but this assumption can be replaced by requiring them to be simple. For path-width we obtain the following result:

5.20 Lemma. Every cubic graph of path-width at most 4 contains at least one isomorphic copy of the following graphs as a subgraph: the triangle, the $K_{2,3}$, the domino, the twin-house, or the claw-square. $\triangleleft$ [2.4] [5.19] [7.10] [7.25] (5.21)

Proof. A graph of path-width at most 3 is also of tree-width at most 3 and, hence, the lemma follows from Lemma 5.19 in this case. Consequently, we assume G has path-width 4 and consider a smooth path-decomposition $(P, \mathcal{V})$ of G with $P := 1 \ldots n'$ and $P' := P - n'$, which exists by Theorem 2.4. Thus, all bags contain exactly five vertices of G and $|V_i \cap V_{i+1}| = 4$ for all $i \in P'$. For each $i \in P$, we denote the vertex entering V_i by v_i and the vertex leaving V_i by w_i (if they exist). In this proof, we make strong use of the following property, which is a direct consequence of the definition of path-decompositions:

$$N_G\, w_i \subseteq \bigcup_{j=1}^{i} V_j \text{ for all } i \in P'. \tag{5.1}$$

Since the proof involves quite a few case distinctions, we refer to Figure 5.12 for an orientation of which edges are present in each case. Moreover, we use a lot of symmetry arguments and omit stating that these are without loss of generality every time.

Let $V_1 = \{v, u_1, u_2, u_3, u_4\}$ and $w_1 = v$. (This is a typical example of the symmetries we use: we can always rename the vertices such that this is true.) By (5.1), we obtain $Nv \subseteq \{u_1, u_2, u_3, u_4\}$, say $Nv = \{u_1, u_2, u_3\}$. We get two cases for w_2, namely

$$w_2 = v_2 \text{ or } w_2 = u_1.$$

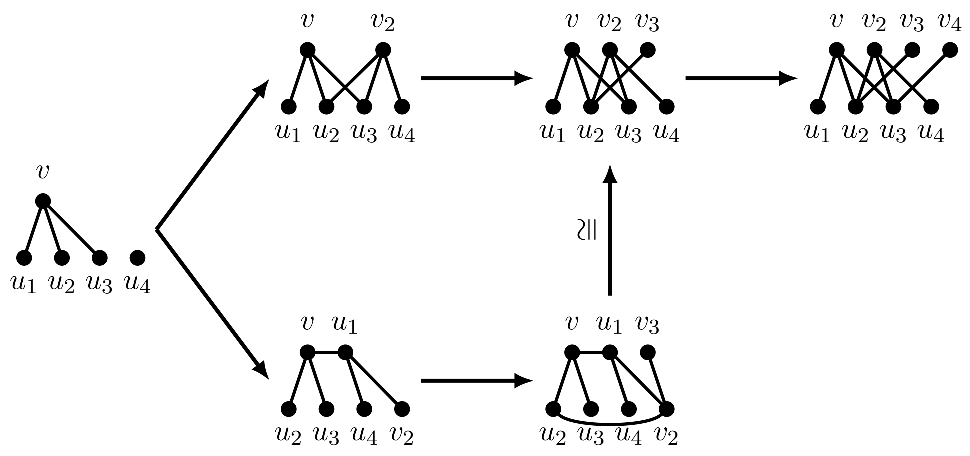


Figure 5.12.: A roadmap of the case distinctions made in the proof of Lemma 5.20. Column i shows the options that occur when the ith bag is considered.

In the first case, $Nv_2 = Nv$ gives us a $K_{2,3}$. Thus, $Nv_2 = \{u_2, u_3, u_4\}$. This time, we have three options for w_3: u_1, u_2, or v_3. The first and last case yield a desired subgraph: if $w_3 = u_1$, then Nu_1 contains a vertex in $\{u_2, u_3, u_4\}$. The first two options give us a triangle, the last a twin-house. For $w_3 = v_3$, we get a $K_{2,3}$ or $Nv_3 = \{u_1, u_2, u_4\}$ and a domino is present.

This puts us in the case that $w_3 = u_2$, which must have a neighbour in $\{v_3, u_1, u_3, u_4\}$. All but v_3 yield a triangle, so we are done or $u_2v_3 \in EG$. We now get four cases for w_4: u_1, u_3, v_3, or v_4. If $w_4 = u_1$, then u_1 has a neighbour in $\{v_3, u_3, u_4\}$, yielding a domino, a triangle, or a twin-house. If $w_4 = v_3$, then v_3 has a neighbour in $\{u_1, u_3, u_4\}$, yielding a domino, a $K_{2,3}$, or another domino. If $w_4 = v_4$, then v_4 has three neighbours in $\{u_1, u_3, u_4, v_3\}$. From $u_3 \in Nv_4$, we get a domino if the edge v_4u_1 or v_4u_4 is present, one of which must be. Otherwise, $Nv_4 = \{u_1, u_4, v_3\}$ and the graph contains a claw-square.

The last option is $w_4 = u_3$, in which we have $u_3v_4 \in EG$ since u_3u_4 and u_3u_1 result in triangles and u_3v_3 yields a $K_{2,3}$. By considering w_5, for which only the two options $w_5 = v_3$ or $w_5 = v_5$ remain, we complete this branch of the proof. We get that v_3v_4, v_3u_1, and v_3u_4 give us a twin-house and two dominoes, finishing the case $w_5 = v_3$. For $w_5 = v_5$, choosing any three of the possible neighbours (all choices are symmetric) creates a claw-square.

Now we go back to our first split and deal with the case that $w_2 = u_1$. We get a triangle or $Nu_1 = \{v, v_2, u_4\}$. This gives two choices for w_3 here, namely v_3 or v_2. The first case gives $Nv_3 = \{u_2, u_3, u_4\}$ and a twin-house is present. If $w_3 = v_2$, then we need two neighbours in $\{u_2, u_3, u_4, v_3\}$. We can exclude u_4 as it forms a triangle, and both u_2 and u_3 yield a $K_{2,3}$. Thus, $Nv_2 = \{u_1, u_2, v_3\}$ and the obtained case is actually one we have already seen before, see Figure 5.12. ☐

Using Lemma 5.20 and the reductions from Section 5.2 we can prove the following theorem.

5.21 Theorem. Every 3-connected cubic graph of path-width at most 4 satisfies the 3-decomposition conjecture. ◁ [5.20]

Proof. Suppose towards a contradiction that the class of all 3-connected cubic graphs of path-width at most 4 does not satisfy the 3-decomposition conjecture. Let H be a counterexample of minimum order amongst all graphs of this class. By Lemma 5.20, at least one of the graphs listed there is a subgraph of H. Observe that all the corresponding reductions (as discussed in Section 5.2) produce a minor of H and are, hence, path-width preserving. Moreover, these transformations preserve 3-connectivity. Altogether, H can be reduced to a 3-connected graph G of path-width at most 4 with $|VG| < |VH|$. Since H was a minimum counterexample, the graph G satisfies the 3-decomposition conjecture and, because the transformations used are 3-compatible, so does H, yielding a contradiction. (Note that for the domino, we only use the reduction to the edge if it preserves 3-connectivity and that we can extend a decomposition from the square to the domino in the case we do not reduce to the edge.) □

Next, we prove the following properties of minimum counterexamples.

5.22 Theorem. If G is a minimum counterexample to the 3-decomposition conjecture amongst all 3-connected cubic graphs, then it does not contain any of the graphs in Figure 5.1 on Page 67 as a subgraph. In particular, [5.8]

(a) the girth of G is at least 4,
(b) every cycle of length 4, 5, or 6 in G is induced, and
(c) every edge of G is the centre edge of an induced P_6. ◁

Proof. Suppose that there exists a 3-connected counterexample to the 3-decomposition conjecture. Choose G to be minimum amongst all such counterexamples. By Theorem 5.8, the six graphs listed in the claim are reducible and hence not subgraphs of G. We refer to the property that the triangle is not a subgraph of G by (Δ) and use ($K_{2,3}$), (Pet^-), (cs), (th), and ($\boxminus$) for the respective properties corresponding to the other subgraphs. Properties (a) and (b) follow immediately from (Δ) and ($\boxminus$). We prove Property (c). Let wx be some edge of G.

Claim 1. If wx is in a cycle of length 6, then Property (c) is satisfied for wx. ◁

Proof. Let $C := uvwxyzu$ be a cycle which contains wx. By Property (b) the cycle C is induced and, consequently, there exists a vertex $v' \in N_G\, v \setminus VC$. Consider the path $P := v'vwxyz$ as shown on the left of Figure 5.13. If P is induced, then Claim 1 holds. Therefore, we may assume that there is an edge in G between two vertices of P that are non-adjacent in P. For simplicity, we call such an edge a chord of P. Since C is induced, one end of this chord is v' and by (Δ) the other end is neither w nor u, leaving the cases that

$$v'x,\ v'y,\ \text{or}\ v'z$$

is an edge of G.

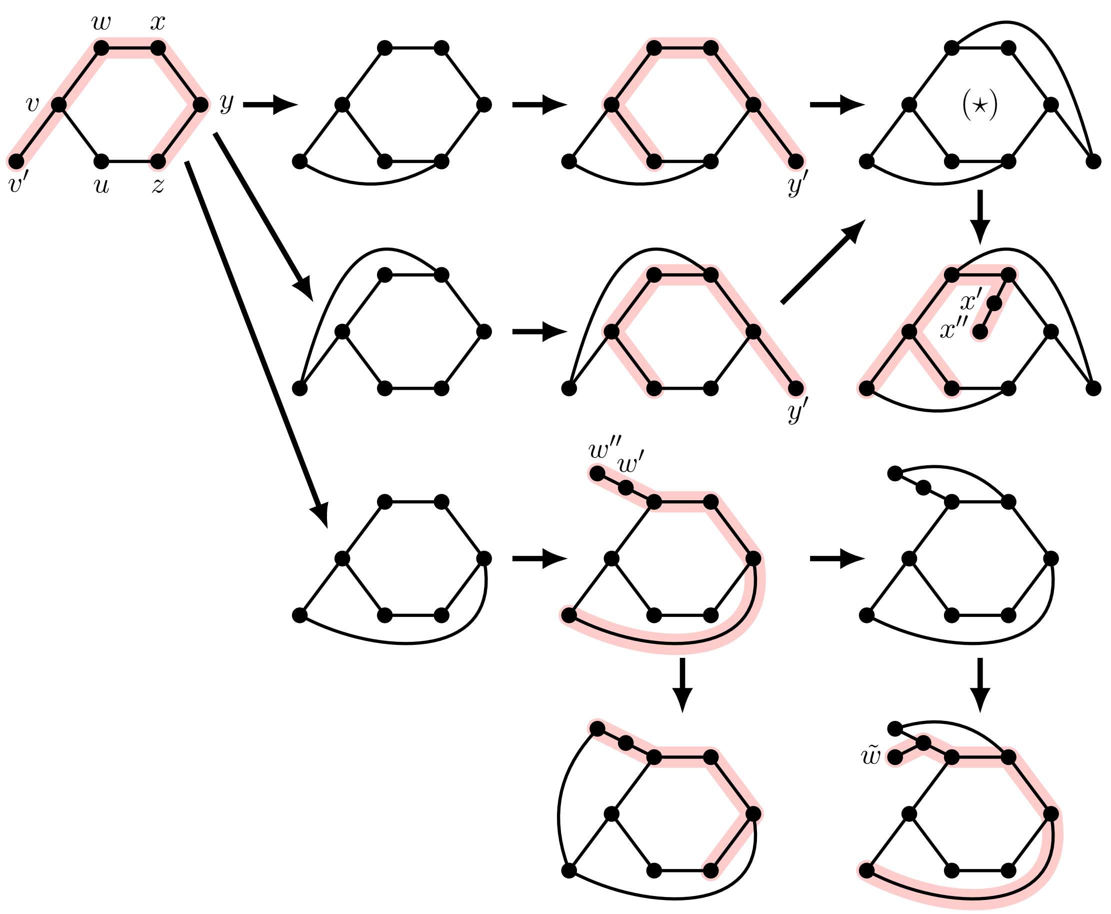


Figure 5.13.: The graphs occurring in the proof of Claim 1. By starting with a C_6 containing wx on the top left and using the properties available in this case, the local view of the graph is expanded until an induced P_6 is found that contains wx as its centre edge.

Assume that $v'z \in E\,G$. The vertex y has a neighbour $y' \notin V C \cup \{v'\}$ by (Δ). Consider the path $uvwxyy'$. If this path is not induced, then $y'w \in E\,G$ by (Δ) and $(\boxminus)$. We label this situation by $(\star)$ so we can refer to it later. From (Δ) and $(\boxminus)$ we obtain that x has a neighbour $x' \notin V C \cup \{v', y'\}$. We prove that both remaining neighbours of x' are not in $V C \cup \{v', y'\}$: since G is cubic, x' is not adjacent to v, w, y, or z. By $(K_{2,3})$ we obtain that $x'y' \notin E\,G$ and with (cs) we obtain $x'u$, $x'v' \notin E\,G$. Hence, there exists a vertex x'' in $N_G\,x' \setminus (V C \cup \{v', y'\})$. Consider the paths $uvwxx'x''$ and $v'vwxx'x''$. At least one of the two paths is induced since the only possible chords are ux'' or $v'x''$, respectively. However, if both chords exist, then $(K_{2,3})$ is violated. Altogether, if $v'z \in E\,G$, then G contains a path of the desired form.

Now assume that $v'x \in E\,G$. There exists $y' \in N_G\,y \setminus (V C \cup \{v'\})$ since C is chordless and (Δ) holds. Consider the path $uvwxyy'$. We may assume that this path is not induced, that is, $y'u \in E\,G$ (the other potential chord wy' violates $(\boxminus)$). Observe that the vertices can be relabelled such that we are in the same situation as $(\star)$, which includes the fact

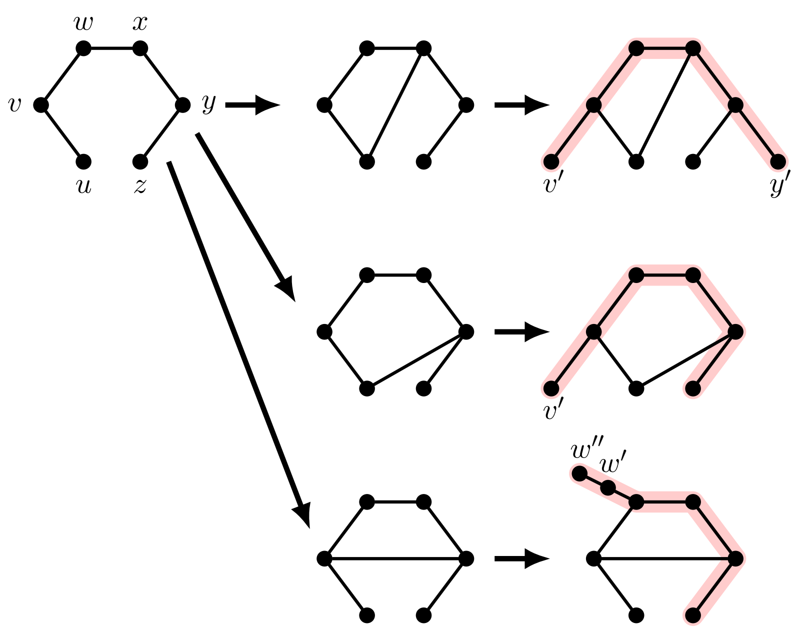


Figure 5.14.: The graphs occurring in the proof of Claim 2. Again, the initial P_6 containing wx is expanded until an induced version is found.

that the edge wx is identified, even though its ends are 'swapped'. Hence, there exists an induced P_6 with wx at its centre.

Finally, let $v'y \in E\,G$. Since C is chordless and (Δ) holds, we obtain that there exists a vertex $w' \in N_G\,w \backslash (V C \cup \{v'\})$. The only possible neighbour of w' in $V C \cup \{v'\}$ is z by (Δ), (th), and (cs). Since $d_G\,w' = 3$ there exists another vertex $w'' \in N_G\,w' \setminus (V C \cup \{v'\})$. If $w''w'wxyv'$ is induced, then Claim 1 is true. Therefore, assume there is a chord of $w''w'wxyv'$ in G. By (Δ), the only possible chords are

$$w''x \text{ and } w''v'.$$

If $w''v' \in E\,G$, then the path $w''w'wxyz$ is induced: the edges $w''x$ and xz violate (Δ), the edges $w''x$ and $w''z$ are prohibited by (th), and the last potential chord $w'z$ contradicts (Pet$^-$). Otherwise, if $w''x \in E\,G$, then there exists $\tilde{w} \in N_G\,w' \setminus (V C \cup \{w'', v'\})$ by ($\boxminus$) and (th), and the path $\tilde{w}w'wxyv'$ is induced by (cs). $\diamond$

Claim 2. If wx is the centre edge of a P_6 and wx is not contained in a cycle of length 6, then Property (c) is satisfied for wx. $\triangleleft$

Proof. Let $P := uvwxyz$ be a P_6 contained as a subgraph G as seen in Figure 5.14. If P is induced, then Claim 2 is satisfied. Therefore, we may assume that P has a chord. By (Δ) and since wx is not contained in a C_6, the following chords are possible: ux, uy, vy, vz, and wz. By symmetry, it suffices to consider the chords

$$ux,\ uy, \text{ and } vy.$$

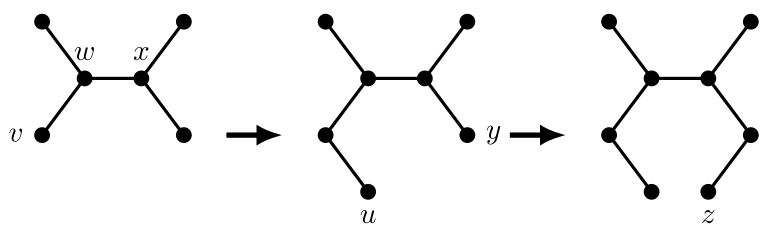


Figure 5.15.: The graphs occurring in the proof of Claim 3. Using the forbidden subgraphs, the graph around wx is explored until a P_6 with wx at its centre is found.

First assume that $ux \in E\,G$. By $(K_{2,3})$ and (th), v has a neighbour $v' \notin V P$. Similarly, we get that y has a neighbour $y' \notin V P \cup \{v'\}$ by (Δ), $(K_{2,3})$, and (th). The path $v'vwxyy'$ is induced since a chord would either violate (Δ) or $(\boxminus)$ or lead to a C_6 containing wx.

Now assume that $uy \in E\,G$. There is a neighbour v' of v with $v' \notin V P$ by (Δ) and (th). The path $v'vwxyz$ is induced since a chord of this path would either violate (Δ) or (th) or create a C_6 with wx as an edge.

Finally assume that $vy \in E\,G$. From (Δ) and $(K_{2,3})$, we obtain that w has a neighbour $w' \notin V P$ and w' has a neighbour $w'' \notin V P$ by (Δ), $(\boxminus)$, and (th). A chord of $w''w'wxyz$ does not exist due to the same three properties and the assumption that no C_6 contains wx. $\diamond$

Claim 3. There exists a P_6 whose centre edge is wx. $\triangleleft$

Proof. By (Δ), we have $N_G\,w \cap N_G\,x = \varnothing$. Let $v \in N_G\,w \setminus \{x\}$ as seen in Figure 5.15. There exists a neighbour u of v with $u \notin N_G\,w \cup N_G\,x$ due to (Δ) and $(K_{2,3})$. Let y be a vertex in $N_G\,x \setminus \{w\}$. By (Δ), $(K_{2,3})$, and (th), we obtain that y has a neighbour $z \notin N_G\,w \cup N_G\,x \cup \{u\}$. Altogether $uvwxyz$ is the desired P_6. $\diamond$

This completes the proof. $\square$

Part II.

Algorithmic Proof Support

In this part, we readdress two of the proof techniques we used in Chapter 5. We target those that deal with exhaustive case analyses and show how to formalise these proofs such that they can be checked automatically. Specifically, we revisit the proofs for 3-compatibility of extensions in Chapter 6 and provide a method for finding all the ‘straightforward’ extensions we relegated to Appendix A. We also verify that all small cubic graphs have a 3-decomposition. In Chapter 7, we automate the proof of Lemma 5.20 where we showed that certain subgraphs are part of every cubic graph of path-width at most 4. More precisely, we develop an algorithm that checks whether a set $\mathcal{U}$ of graphs is unavoidable for a class of graphs $\mathcal{G}$ if we restrict their path-width. This problem is undecidable in general (so the algorithm need not terminate in case $\mathcal{U}$ is unavoidable), but we give a criterion that ensures termination. Finally, in Chapter 8, we use the developed techniques to provide upper bounds on the girth of cubic graphs with small path-width.

Chapter 6.

Naive Extensions

We formalise and automate the part of the reducibility proofs of Section 5.2 that we discuss in Appendix A. Moreover, we determine that the process described here leads to an NP-complete problem. Finally, we note that all cubic graphs of order at most 20 satisfy the 3-decomposition conjecture.

At the core of our reducibility proofs in Section 5.2 lies the task of checking that *every* behaviour, which a 3-decomposition might exhibit on a (smaller) graph S, can be extended to R. This leads to a lot of cases, many of which are straightforward. We identify these *naively extendable* cases and demonstrate how they can be handled algorithmically in Section 6.1. This provides an alternative proof for the contents of Appendix A. The algorithm we use for checking the extendability is an exhaustive search, which suffices for our purposes since the configurations we check are small. Nevertheless, we prove that checking for naive extendability is an NP-complete problem in Section 6.2. We also use the same exhaustive search (together with a heuristic) to show that all cubic graphs of order at most 20 satisfy the 3-decomposition conjecture in Section 6.3, meaning a potential counterexample must have order at least 22.

This chapter is joint work with Irene Heinrich and is the last part of [BH21] that was missing.

6.1. Algorithmically checking 3-compatibility

Recall that we wanted to prove that certain (X, Y)-extensions are 3-compatible for template graphs X and Y. We refer to Definitions 5.1, 5.2, and 5.4 for the corresponding definitions. To make it possible to check the 3-compatibility of an extension in an automated fashion, we start by taking a look at what the tree-component of a valid behaviour looks like when restricted to a template graph, which is exactly what the following definition describes.

6.1 Definition. A spanning forest T_X of a template graph X is *3-consistent* if

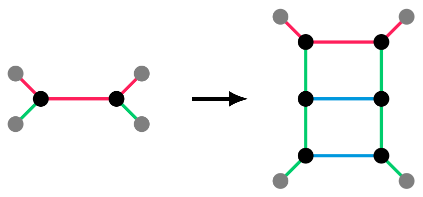


Figure 6.1.: A naively extendable 3-consistent forest. The green edges in both graphs form a 3-consistent forest. Moreover, the forest for the template graph of the edge is naively extendable to the template graph of the domino.

- every component of T_X contains an outer vertex and
- every component of $X - E\,T_X$ is a single vertex, a single edge, a cycle, or a path joining two outer vertices of X.

We write M_X for the set of edges uv for which $\{u, v\}$ is a component of $X - E\,T_X$ and denote the union of the cycles and paths between outer vertices in $X - E\,T_X$ by C_X. ◁

See Figure 6.1 for two examples of 3-consistent forests, where we employ our usual colour scheme: the edges of T_X are green, those in C_X are red, and the ones in M_X are blue.

(6.5) **6.2 Observation.** The requirements for a forest to be 3-consistent are all necessary for it to be a restriction of a 3-decomposition. In other words, if a cubic graph G contains $c\,X$ as an induced subgraph and has a 3-decomposition (T, C, M), then the restriction T_X of T to X is 3-consistent. ◁

Recall that X is not necessarily a subgraph of G (just the core of X is), but we treat it as one since its edges can be identified with ones of G. Thus, the restriction of T to X is the subgraph of X containing the edges that correspond to ones in T. Further, we remark that the restriction of T to X is not necessarily connected, and the restrictions of C and M to X are C_X and M_X, where C_X may now contain paths in addition to cycles.

An *assignment* is a function $f\colon OX \to \{\mathtt{t}, \mathtt{c}, \mathtt{m}\}$. We say that T_X *realises* f if, for each vertex $v \in OX$, its incident edge e satisfies that $e \in T_X$ ($e \in C_X$, $e \in M_X$) if $f\,v = \mathtt{t}$ ($f\,v = \mathtt{c}$, $f\,v = \mathtt{m}$).

6.3 Definition. A 3-consistent forest T_X is *naively extendable* to Y if there exists a 3-consistent forest T_Y of Y such that

- T_Y realises the same assignment as T_X (in particular, $OX = OY$) and
- two outer vertices of X are in the same component of T_X if and only if they are in the same component of T_Y. ◁

An example of a naively extendable forest is shown in Figure 6.1.

We show that a naively extendable forest T_X allows us to extend a 3-decomposition that locally behaves like T_X.

6.4 Lemma. If G has a 3-decomposition (T, C, M) such that the restriction T_X of T to X is naively extendable to Y with forest T_Y, then the extension $H := G[X \to Y]$ has a 3-decomposition. $\lhd$ (6.5)

Proof. We verify that the decomposition (T', C', M') of H given by

$$\begin{aligned} E\,T' &:= (E\,T \setminus E\,T_X) \cup E\,T_Y, \\ E\,C' &:= (E\,C \setminus E\,C_X) \cup E\,C_Y, \text{ and} \\ M' &:= (M \setminus M_X) \cup M_Y \end{aligned}$$

is, indeed, a 3-decomposition of H.

We first prove that T' is a spanning tree, where we assume that T' is spanning. Therefore, we only need to verify that T' is connected and acyclic. To show that T' is connected, fix a vertex $v \in H - I\,Y = G - I\,X$. There exists a path P in T from this vertex to every other vertex in $G - I\,X$. If P does not use vertices of $c\,X$, then it is present in T'. Otherwise, P contains a subpath that links two outer vertices of X. This places them in the same component of T_X and they are connected in T_Y. Thus, we can replace every such subpath in T_X by one in T_Y to obtain a path in T'. Any vertex $w \in c\,Y$ is in the same component of T_Y as an outer vertex, which is connected to v, making T' connected.

Suppose now that T' contains a cycle K. This cycle cannot be contained in $H - I\,Y$ and, hence, K consists of paths using edges in $E\,T \setminus E\,T_X$ and paths using edges in $E\,T_Y$. By choosing a cycle that consists of a minimal number of such segments, we can ensure that it enters every component of T_Y at most once. Any such path can be replaced by one in T_X, giving us a cycle in T which is a contradiction.

The set M' is a matching since the edges that are in $M \setminus M_X$ and those in M_Y are independent. So if e, $e' \in M'$ share an end v, we may assume that $e \in M \setminus M_X$ and $e' \in M_Y$. Therefore, their common end is a vertex in OX. In this case $e' \notin M_X$, giving us $e' \notin M_Y$ since $f\,v \neq \mathtt{m}$, where f is the assignment realised by T_X, which is a contradiction.

Finally, for $v \in H$, we take a look at the number of edges in C' incident to v. If $v \in c\,Y$ or v is in H but not adjacent to $c\,Y$, then this degree is either 2 or 0. This just leaves the vertices OY. Since T_X and T_Y realise the same assignment, any edge from a vertex of $G - c\,X$ to $c\,X$ is in C_X if and only if the corresponding edge from $H - c\,Y$ to $c\,Y$ is in C_Y. Consequently, the degree of these vertices remains the same in C and C', yielding a degree of 2 or 0 here as well. $\square$

The proof of Lemma 6.4 is constructive in the sense that, given a 3-decomposition for G as in the statement, we can construct a 3-decomposition for H. We can now automate the proofs we relegated to Appendix A in Chapter 5 by verifying that 3-consistent forests are naively extendable.

[6.2] [6.4] **6.5 Theorem.** To prove that an (X, Y)-extension is 3-compatible, it suffices to show that any 3-consistent forest T_X of X satisfies

(i) T_X is naively extendable to Y or

(ii) every (X, Y)-extension $H \coloneqq G[X \to Y]$ of a graph G has a 3-decomposition if G has a 3-decomposition (T, C, M) for which the restriction of T to X is T_X. $\triangleleft$

Proof. This is true as any 3-decomposition (T, C, M) of a graph G containing an induced $c\,X$ has a 3-consistent restriction T_X of T to X by Observation 6.2. If T_X is naively extendable, then H has a 3-decomposition by Lemma 6.4 and otherwise such a decomposition exists by Condition (ii). $\square$

The task of determining all 3-consistent forests and checking whether they are naively extendable can be accomplished algorithmically. We have implemented this[1]. With the exception of our analysis of the triangle in Lemma 5.9, the local behaviours covered in Section 5.2 are exactly the ones that are not naively extendable, that is, where we are required to use Condition (ii) (disregarding symmetric cases).

6.2. The complexity of naive extensions

We now prove that determining whether a graph has a naive extension is an NP-complete problem. We first restrict ourselves to the subproblem of finding a 3-consistent forest T_Y that realises some assignment f. As in Theorem 3.4, this reduction is also based on the one for the integral multi-commodity flow problem [EIS76].

(6.7) **6.6 Theorem.** Given a template graph Y and an assignment f, checking whether a 3-consistent forest T_Y for Y exists that realises f is NP-complete. $\triangleleft$

Proof. The problem is in NP since a spanning forest T_Y of Y can be checked to be 3-consistent and realise f in polynomial time. To prove NP-hardness we use a reduction from 3-SAT, see Problem 2.15. Let φ be a formula with variables $x_1, \ldots, x_n$ and clauses $X_1, \ldots, X_m$. We assume that, for every variable x, both x and $\overline{x}$ occur equally often in the clauses. This is without loss of generality since we can add additional clauses of the form $x \vee x \vee \overline{x}$ or $x \vee \overline{x} \vee \overline{x}$ as required to reach the desired state.

We first note that we can force edges uv to end up in the tree part T_Y by subdividing them and adding an outer vertex to the subdivision vertex whose incident edge is in the matching. Formally, we remove uv and add edges uw, vw, wy, where w, y are new vertices and y is an outer vertex. By setting $f\,y \coloneqq \mathtt{m}$, wy needs to be in M_Y and both uw and vw are necessarily in T_Y. For simplicity, we refer to such a gadget as a *forced tree-edge.*

[1]GitLab repository containing 3-decompositions for all connected cubic graphs of order at most 20 and code to check for naive extensions, `https://gitlab.rlp.net/obachtle/reductions-for-the-3-decomposition-conjecture`, August 2022.

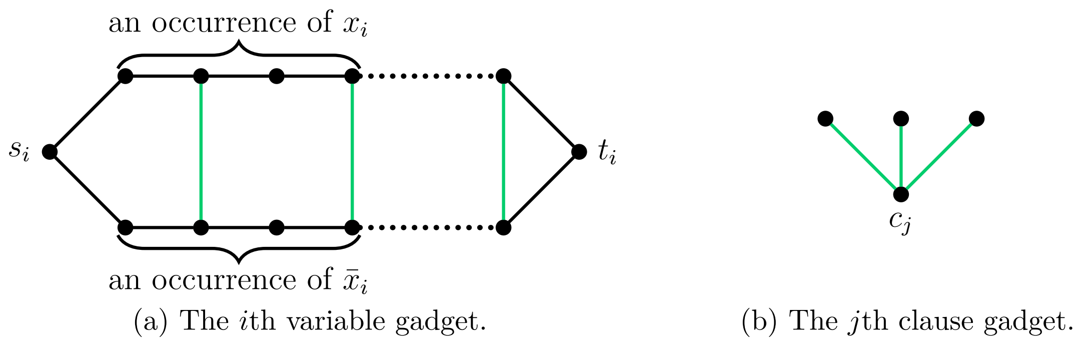

an occurrence of x_i
s_i
t_i
an occurrence of $\bar{x}_i$
(a) The ith variable gadget.
c_j
(b) The jth clause gadget.

Figure 6.2.: The gadgets used in the proof of Theorem 6.6. Here, the forced tree-edges are simply drawn by a green edge (without depicting the additional vertices they contain).

With this gadget at hand, we can describe the clause and variable gadgets we construct. Clause gadgets are very simple and shown in Figure 6.2b. They just consist of a vertex c_j incident to three forced tree-edges, whose ends are in the variable gadgets (we describe where momentarily). Each such edge corresponds to one of X_j's literals.

The variable gadget for x_i consists of two paths of length $4l_i + 1$ which coincide in their ends s_i and t_i and are disjoint otherwise. Here, l_i is the number of occurrences of x_i (or $\bar{x}_i$) in φ. We call one of these paths the *upper path* and the other the *lower* one. In this gadget, we partition the $4l_i$ distinct vertices on the upper and lower paths into l_i blocks of four consecutive vertices and each block on the upper path corresponds to an occurrence of x_i while each block on the lower path corresponds to an $\bar{x}_i$ in some clause. Furthermore, the lth vertices of the upper and lower path are connected by a forced tree-edge for all even l. This is visualised in Figure 6.2a.

The entire graph now concatenates all the variable gadgets by adding edges from s_i to t_{i+1} for $i \in \{1, \dots, n-1\}$. It adds an outer vertex to s_1 and t_n, where the edges are required to be part of C_Y (by setting the f-value of the outer vertices to `c`). A forced tree-edge in a clause gadget corresponding to a literal L is connected to the third vertex in the block that corresponds to L. Finally, all the first vertices in a block are connected by a forced tree-edge to a path consisting of only forced tree-edges that ends in two more leaves. These two leaves are assigned an f-value of `t`. The construction of Y runs in polynomial time, so we just need to prove that φ is satisfiable if and only if there exists a 3-consistent forest T_Y realising f.

First assume that a 3-consistent T_Y exists that realises f. The cycle component C_Y contains a path P from s_1 to t_n in the graph $Y - E\,T_Y$. Since T_Y contains all forced tree-edges, P either uses the upper or lower path in each variable gadget. If it uses the lower path in the gadget for x_i, then we set x_i to `True` and otherwise we set it to `False`. Suppose this assignment would not satisfy φ. This yields a clause X_j for which all three of its literals are not satisfied, meaning that we use the upper path for the positive literals and the lower path for the negated ones. Consequently, the vertex c_j

and its neighbours in the variable gadgets form a component of T_Y without an outer vertex, contradicting the fact that T_Y is 3-consistent.

Conversely, assume that we are given a satisfying assignment for φ. Let T_Y consist of the following edges: T_Y contains all forced tree-edges. Additionally, if x_i is set to `True`, T_Y uses the upper path and otherwise it uses the lower one. Here, uses a path means that T_Y contains the path's edges incident to s_i and t_i and in each block it contains the first and third edge. As a result, after the first edge, the edges on the path alternate between being and not being in T_Y.

This is a forest: it has a component for each clause gadget X_j which subsumes the third vertices of all blocks corresponding to its literals. Disregarding the components consisting of a single outer vertex, there is one other component containing the path of forced tree-edges and the first (and second) vertex of all blocks. The cycle component C_Y consists of a path which links the outer vertices at s_1 and t_n and all other components are single edges, placing them in M_Y. As a result, T_Y realises f and we can make it 3-consistent by choosing a satisfied literal in every clause X_j and connecting the component of X_j to the one containing the path by adding the edge from the second to the third vertex in the corresponding block to T_Y. □

Using the same notation as in the proof above, we note that the reduction presented in Theorem 6.6 can also be used to prove the following corollary.

[6.6] **6.7 Corollary.** Determining whether a forest is naively extendable is NP-complete. ◁

Proof. To get this result, one simply needs to define a graph X and a 3-consistent forest T_X that also realises the assignment f specified in Theorem 6.6 and in which the only two outer vertices with an incident edge in T_X are in the same component of T_X. Such a graph X can be constructed by taking a path P that has an inner vertex for every forced tree-edge. All these inner vertices have degree 2 and are assigned an outer vertex as their final neighbour. The ends of P receive one outer vertex and one further neighbour in the core of X. These final two vertices are connected by an edge and each of them is incident to another outer vertex. The forest T_X now uses P and all edges incident to the ends of P, yielding a path as C_X and leaving M_X with all edges incident to inner vertices of P that are not in P. □

6.3. 3-decompositions for small graphs

In this section, we just briefly explain how we obtained the following result.

(5.10) (5.13) (5.18) **6.8 Theorem.** All graphs of order at most 20 satisfy the 3-decomposition conjecture. In particular, a minimum counterexample has order at least 22. ◁

The cubic graphs of order at most 20 are known, see [Bri96, Mer99, Bri+13]. We used the complete list of cubic graphs of order at most 20 as provided by [Bri+13]. For each graph in this list, we computed a tree that leads to a 3-decomposition for this graph, i.e., removing the edges of this tree from its host graph results in the disjoint union of a 2-regular graph, a 1-regular graph, and some isolated vertices. To obtain such a tree for a given cubic graph, we first employed a heuristic. If the heuristic did not lead to a desired tree, then our algorithm tackled the problem by an enumeration approach. The code as well as the computed trees can be found in our GitLab repository[2].

We complete this section with a brief overview of the heuristic employed. When presented with a graph G, it first searches for non-separating cycles in the graph and removes them. It does so heuristically, by computing a spanning tree in the graph and seeing if the cycle obtained by adding one of the remaining edges to this tree is non-separating. (Recall that finding such cycles in NP-complete by Theorem 3.4.) After removing a non-separating cycle, the degree 1 vertices are recursively pruned, their incident edges must be part of the tree. Since we know, by the equivalence to the 2-decomposition conjecture, that decompositions continue to exist in the new graph unless the conjecture is false, we continue with the preprocessed version.

In it, we then compute a spanning tree T, which yields a decomposition into this spanning tree, a 2-regular subgraph C, and paths P. Note that the edges left over when removing T can contain cycles as well, we combine those with the non-separating cycles we found. If no path in P has length greater than 1, a 3-decomposition has been found and is returned. Otherwise, to reduce the number of long paths, an edge e at the end of such a path is taken. By adding it to T, we obtain a cycle C' in $T + e$. If this cycle is non-separating, we include it in the cycle component and compute a new spanning tree. Otherwise, we check whether the cycle contains an edge $e' = uv$ where both u and v have degree 0 in P (and thus degree 3 in $T + e$). In the case that it does, we remove e' from T and add it to P, where it forms a path of length 1. Should we manage to remove all excessively long paths this way, we obtain a 3-decomposition and otherwise the heuristic fails.

When the heuristic fails, we revert to the preprocessed graph and check whether it contains a spanning tree with the desired behaviour by iteratively extending assignments to the three components in a consistent way. In essence, we use the possible local behaviours at a vertex, as displayed in Figure 5.4, and use these to colour all edges. We keep track of components to detect when adding edges to the tree creates a cycle.

[2]GitLab repository containing 3-decompositions for all connected cubic graphs of order at most 20 and code to check for naive extensions, `https://gitlab.rlp.net/obachtle/reductions-for-the-3-decomposition-conjecture`, August 2022.

Chapter 7.

Unavoidable Structures

Let $\mathcal{G}$ be a class of graphs with a membership test, $k \in \mathbb{N}$, and let $\mathcal{G}^k$ be the class of graphs in $\mathcal{G}$ of path-width at most k. We present an interactive framework that finds an *unavoidable set* for $\mathcal{G}^k$, that is, a set of graphs $\mathcal{U}$ such that any graph in $\mathcal{G}^k$ contains an isomorphic copy of a graph in $\mathcal{U}$. At the core of our framework is an algorithm that verifies whether a set of graphs is, indeed, unavoidable for $\mathcal{G}^k$. If it is not, it provides a counterexample that can be used to extend the set $\mathcal{U}$.
In general, it is undecidable whether a finite set of graphs is unavoidable for a given graph class, even if the graphs in the class are all of path-width 2 and the class itself is decidable. However, we give a criterion for termination: our algorithm terminates whenever $\mathcal{G}$ is highly local, of bounded maximum degree, and $\mathcal{U}$ is a finite set of connected graphs. We put special emphasis on the case that $\mathcal{G}$ is the class of cubic graphs and tailor the algorithm to this case. In particular, we introduce the new concept of high-degree-first path-decompositions, which enable highly efficient pruning techniques.

A set of graphs is *unavoidable* for a graph class if every graph in the class contains an isomorphic copy of a graph in the set. Unavoidable sets are extensively used in

- recursive algorithms and inductive proofs, for example [Bod+19, FGH20],
- structural graph theory, where unavoidable sets give insights into the possible composition of graphs, see [Chu+16] for an example, and
- interactive proofs, for example Appel and Haken's proof of the famous four colour theorem, see [AH77, AHK77].

While discharging [CW17] is a tool to find unavoidable structures for colouring problems and Ramsey theory [CFS15] studies unavoidable sets in extremal graph theory, there is no generic approach for finding or checking whether a given set is unavoidable. Frequently, tedious case distinctions are necessary to prove that some set is indeed unavoidable for a considered class, see [Bod+19, FGH20]. We contribute a new automated method to this sparse list of techniques for finding unavoidable sets.

We present an interactive framework for the automatic construction and testing of unavoidable sets parametrised by path-width. More precisely, let a class $\mathcal{G}$ with a membership test and a number $k \in \mathbb{N}$ be given and let $\mathcal{G}^k$ be the class of graphs in $\mathcal{G}$ of path-width at most k. At the core of our framework is an algorithm which investigates the hypothesis that a finite set of graphs $\mathcal{U}$ is unavoidable for $\mathcal{G}^k$. It can be adapted to serve as a check for unavoidable induced subgraphs or minors, as we see in Section 7.5. We prove that the algorithm terminates for a large variety of graph classes, for example if $\mathcal{G}$ is the class of all l-regular graphs for some $l \in \mathbb{N}$ and the graphs in $\mathcal{U}$ are connected. The framework uses the algorithm as follows: it starts with a (potentially empty) set of graphs and runs the algorithm on it. If the set is not unavoidable, the provided counterexample can be used to extend the set of structures, either by adding the graph itself or a subgraph. This is the interactive part of the framework, since the desired or useful unavoidable structures often depend on the application. This process is repeated until the set of structures is unavoidable.

We now give a high-level description of the algorithm we develop in this chapter and describe the challenges that occur. Roughly speaking, our algorithm searches for a minimum counterexample to the hypothesis that $\mathcal{U}$ is unavoidable for $\mathcal{G}^k$. We face two major challenges on our way to a practically applicable algorithm: a very large search space and the fact that, in general, it is undecidable whether a finite set of graphs is unavoidable for a given graph class, even if the class is decidable and of path-width 2 (see Lemma 7.13). In order to reduce the search space we exploit a natural linear vertex-ordering which is given by a path-decomposition and we make strong use of isomorphism rejection (described very vaguely below and in detail in Section 7.2). Concerning the undecidability, which prevents the algorithm from terminating in general, we prove a criterion which allows us to guarantee termination whenever $\mathcal{G}$ is highly local and of bounded maximum degree and the graphs in $\mathcal{U}$ are connected in Section 7.3. Recall that, amongst many others, the following graph classes are highly local: bipartite graphs, l-colourable graphs, and H-free graphs, where H is some fixed graph. We also analyse the running time of the algorithm when applied to highly local graph classes.

Let us now give a slightly more precise description of the algorithm, which runs in phases. After the ith phase, it either

- returns a counterexample of order $k + i$, or
- returns `None`, guaranteeing that no counterexample exists and $\mathcal{U}$ is unavoidable, or
- ensures that no counterexample of order $k + i$ exists and proceeds with phase $i + 1$.

In essence, the algorithm checks graphs $G \in \mathcal{G}^k$ by simulating the traversal of a smooth path-decomposition from left to right. It manages a queue that, at the beginning of phase i, contains all pairs of the form (U, H) which satisfy:

- H is a subgraph of some graph $G \in \mathcal{G}^k$ which has a smooth path-decomposition with U as its ith bag. In particular, $|VH| = k + i$.
- H contains all information provided by the bags preceding U.
- H is a potential subgraph of a counterexample to $\mathcal{U}$ being unavoidable for $\mathcal{G}^k$.

The base algorithm, presented in Section 7.1, checks for the current pair (U, H) whether adding edges to H results in a graph in $\mathcal{G}^k$ that avoids all graphs in $\mathcal{U}$. If this is the case, then the obtained graph is returned as a certificate that $\mathcal{U}$ is not unavoidable for $\mathcal{G}^k$. Otherwise, (U, H) is replaced by multiple pairs (U', H'), which correspond to potential next steps in our simulation of the traversal of a smooth path-decomposition. Here, the sets U' are candidates for the next bag in the decomposition and H' is a supergraph of H of order $|H| + 1$. If no pairs remain in the queue, without a counterexample being found, then the algorithm guarantees that $\mathcal{U}$ is unavoidable for $\mathcal{G}^k$.

To drastically limit the number of additional pairs created, we heavily rely on isomorphism rejection to prune the resulting search tree. In order to avoid a combinatorial explosion, we introduce the new concept of *high-degree-first* path-decompositions. These turn out to be invaluable when tailoring the algorithm to cubic graphs in Section 7.4 since they allow us to impose strong restrictions on the next bag in the path-decomposition (U' in the above notation).

We illustrate the algorithm (and the effects of the optimisation) in Section 7.6 where we use it to reprove Lemma 5.20. Another example, with larger path-width values, is presented in the next chapter and the results can be seen in Table 8.1. Note that the last two rows contain the number of pairs considered by the base algorithm and the one employing both isomorphism rejection and high-degree-first path-decompositions. Our implementation of the algorithm which produced the results in this table can be found on GitLab[1].

Before we start, note that Courcelle's theorem [Cou90, CM93] implies the existence of an algorithm like the one we describe here. However, it does not provide one explicitly since encoding a bound on the path-width in monadic second order logic uses access to the forbidden minors, which exist by [RS04], but are unknown for values larger than 2. (Even if they were known, there are at least $(k!)^2$ many forbidden minors for path-width k [TUK94], resulting in a very large formula.) Moreover, our algorithm has more desirable runtime properties than the one which exists according to [Cou90]: the decision procedure of Courcelle's theorem has non-elementary complexity [FG04].

The content of this chapter is joint work with Irene Heinrich and a preliminary version is available as a preprint on arXiv [BH20].

7.1. The base algorithm

We start this section with a toy example that illustrates the by-hand method we automate and is illustrated in Figure 7.1. We have already seen one such example in the proof of

[1] GitLab repository containing an implementation of our algorithm for checking whether a set $\mathcal{U}$ is unavoidable for $\mathcal{G}^k$, `https://gitlab.rlp.net/obachtle/testing-unavoidable-sets-for-small-path-width`, October 2022.

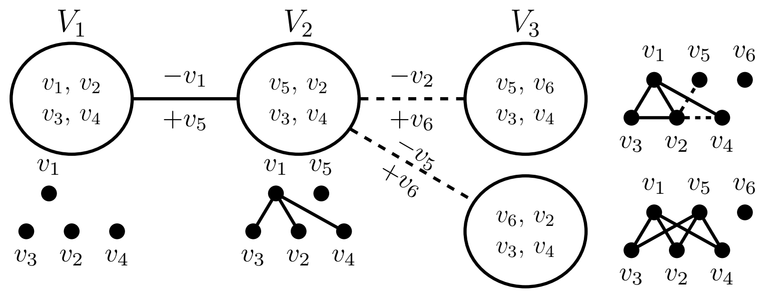


Figure 7.1.: The by-hand method for cubic graphs of path-width at most 3. Here, the bags of the path-decomposition are shown, together with their associated graphs (see Definition 7.2).

Lemma 5.20, but the smaller example presented here should serve as a more clear guide to what is going on as it directly illustrates the behaviour of our algorithm.

Let $\mathcal{G}$ be the class of cubic graphs. We prove that the set $\mathcal{U} := \{C_3, C_4\}$ is unavoidable for the class $\mathcal{G}^3$ of cubic graphs of path-width at most 3. To this end, let $(P, \mathcal{V})$ be a smooth width 3 path-decomposition of a graph $G \in \mathcal{G}^3$. Let $v = v_1$ and v' be the vertices leaving the bags V_1 and V_2, respectively. Since v leaves V_1 (and has degree 3), we obtain $N_G\, v = V_1 \setminus \{v\}$. If v' is a neighbour of v, then we may assume that $v' = v_2$ and v' has two other neighbours in $V_2 \setminus \{v'\}$. Consequently, v and v' have a common neighbour, say v_3, and G contains a C_3. Otherwise, v' is the vertex v_5 entering V_2. In this case, $N_G\, v' = V_2 \setminus \{v'\} = V_1 \setminus \{v\}$ and we obtain a C_4.

Before we can describe our base algorithm, we need some definitions and notation.

7.1 Convention. Barring few exceptions, all path-decompositions occurring in the remainder of this chapter are smooth and adhere to the following naming convention: their path is $P := 1 \dots n'$ and their bags are $\mathcal{V} := (V_i)_{i \in P}$. ◁

Moreover, we need the next definition, which will play an essential role in the remainder of this and the next chapter.

7.2 Definition. The *graph associated with* V_i is

$$G_i := G\left[\bigcup_{j \le i} V_j\right] - E(G\,[V_i]).$$ ◁

Intuitively, G_i contains all information provided by the bags preceding V_i as it has (exactly) the edges incident to vertices that have left already. Note that if v leaves V_i, then $G_{i+1} = G_i + E\,v + v_{i+1}$ for some new vertex v_{i+1}. For convenience, the following term is introduced.

7.3 Definition. A vertex u is a *new neighbour* of v if v is the vertex leaving V_i and $u \in N_{G_{i+1}}\, v \setminus N_{G_i}\, v$. ◁

Thus, the new neighbours of v are exactly those vertices u in G_{i+1} for which uv is an edge of G that was not already present in G_i.

Additionally, we introduce notation to make the statement 'every graph in $\mathcal{G}$ of path-width at most k has a subgraph in $\mathcal{U}$' concise.

7.4 Definition. Let $\mathcal{G}$ be a class of graphs, $\mathcal{U}$ be a finite set of graphs, and let $k \in \mathbb{N}$. We write

- $\mathcal{G}^k$ for the class containing all graphs in $\mathcal{G}$ of path-width at most k, and
- $\text{super}\,\mathcal{U}$ for the class containing those graphs with a subgraph isomorphic to a graph in $\mathcal{U}$, that is, $\text{super}\,\mathcal{U}$ contains all graphs G for which an $S \in \mathcal{U}$ exists with $S \cong S' \subseteq G$. $\triangleleft$

With this, the question whether $\mathcal{U}$ is unavoidable for $\mathcal{G}^k$ translates to: is $\mathcal{G}^k \subseteq \text{super}\,\mathcal{U}$?

Given this setup, we can describe our algorithm that answers the question whether $\mathcal{G}^k \subseteq \text{super}\,\mathcal{U}$, by implementing the proof technique illustrated in the toy example with which we started this section. The only requirement on $\mathcal{G}$ is that it is given together with a membership test, which allows checking whether a specific graph is in $\mathcal{G}$. By checking all graphs of order at most k explicitly, we may assume that $\mathcal{G}$ only contains graphs with at least $k + 1$ vertices. This ensures that all graphs in $\mathcal{G}$ of path-width at most k have a smooth path-decomposition of width k.

Next, we define the pairs we talked about in the outline of the chapter.

7.5 Definition. For each smooth path-decomposition $(P, \mathcal{V})$ of a graph G and each $i \in \{1, \dots, n'\}$ we say that G *contains* (V_i, G_i). A pair (U, H) is *good* if every $G \in \mathcal{G}^k$ containing it satisfies $G \in \text{super}\,\mathcal{U}$. $\triangleleft$

We remark that any pair (U, H) with $H \in \text{super}\,\mathcal{U}$ is good. This holds as every graph G containing (U, H) has H as a subgraph. As a preparation for the formal description and the correctness proof of our algorithm, we prove the following lemma.

7.6 Lemma. Let G be a graph with path-decomposition $(P, \mathcal{V})$ that contains the pair (U, H) and let φ be a bijection with domain VG. (7.7) (7.9) (7.18)

(a) Renaming the vertices of G according to φ, in both G and its path-decomposition, yields an isomorphic graph $\varphi\, G \cong G$ with path-decomposition $(P, (\varphi\, V_i)_{i \in P})$.
(b) The graph $\varphi\, G$ contains the pair $(\varphi\, U, \varphi\, H)$.
(c) If $(V_1, (V_1, \varnothing))$ is a good pair, then $\mathcal{G}^k \subseteq \text{super}\,\mathcal{U}$. $\triangleleft$

Proof. Parts (a) and (b) are immediate from the respective definitions. For Part (c), consider a graph $H \in \mathcal{G}^k$ with a smooth path-decomposition of width k. After renaming the vertices, we may assume that the first bag is V_1. In particular, the graph H_1 associated with V_1 is $(V_1, \varnothing)$. Thus, the graph H contains $(V_1, (V_1, \varnothing))$ and, hence, $H \in \text{super}\,\mathcal{U}$. $\square$

Algorithm 7.1: Base algorithm for checking whether $\mathcal{G}^k \subseteq \text{super}\mathcal{U}$.

Input: A class of graphs $\mathcal{G}$ with a membership test, a finite set of graphs $\mathcal{U}$, and a path-width value k.

Output: An element of $\mathcal{G}^k \setminus \text{super}\mathcal{U}$ or `None` if no such graph exists.

```
def TestUnavoidability($\mathcal{G}$, $\mathcal{U}$, $k$):
    $V_1 \leftarrow \{u_1, \dots, u_k, v_1\}$
    $Q \leftarrow [(V_1, (V_1, \varnothing))]$
    if $(V_1, \varnothing)$ is good then  $Q$.dequeue()
    while $Q \neq \varnothing$ do
        $(U, H) \leftarrow Q$.dequeue()
        for $E' \subseteq \{xy\colon\ x, y \in U,\ x \neq y\}$ do          # search for counterexamples
            if $H + E' \in \mathcal{G} \setminus \text{super}\mathcal{U}$ then
                return $H + E'$
        $i \leftarrow |H| - k$
        for $u \in U$ do                                                  # explore next bag
            $U' \leftarrow U \setminus \{u\} \cup \{v_{i+1}\}$
            for $Y \subseteq U \setminus \{u\}$ do
                $H' \leftarrow H + v_{i+1} + \{uy\colon\ y \in Y\}$
                if $H' \notin \text{super}\mathcal{U}$ and there exists a $G \in \mathcal{G}$ containing $(U', H')$ then
                    $Q$.append($(U', H')$)
    return None
```

We are now ready to describe the algorithm, whose pseudocode can be found in Algorithm 7.1. Given a graph class $\mathcal{G}$ with a membership test, $k \in \mathbb{N}$, and a finite set of graphs $\mathcal{U}$, the algorithm manages a queue Q of pairs (U, H). By Lemma 7.6 (a) we may assume that the vertex set VG is exactly the set $V \coloneqq \{u_1, \dots, u_k, v_1, \dots v_{n'}\}$ and the first bag of the path-decomposition consists of the vertices in $V_1 \coloneqq \{u_1, \dots, u_k, v_1\}$. Therefore, the queue is initialised with the pair $(V_1, (V_1, \varnothing))$. We maintain the following invariant.

$$\text{If all pairs in the queue are good, then } \mathcal{G}^k \subseteq \text{super}\mathcal{U}. \tag{7.1}$$

This holds initially by Lemma 7.6 (c) and, once the queue is empty, we may return that $\mathcal{U}$ is unavoidable, or more precisely, we return `None` since no counterexample exists. During the execution of the algorithm, it iteratively removes pairs (U, H) from the queue. To maintain the invariant (7.1), the algorithm needs to ensure that every graph $G \in \mathcal{G}$ containing (U, H) is in super$\mathcal{U}$. Every such graph has a smooth path-decomposition with U as a bag whose associated graph is H. If this is the last bag in the path-decomposition, then G is obtained from H by adding edges between vertices of U and the algorithm checks all such augmentations to see whether one of them avoids $\mathcal{U}$. Should this occur, the algorithm returns this augmentation as a counterexample. Otherwise, the decomposition does not end with the bag U and the algorithm checks all possible options for the next bag and its associated graph, adding these new pairs to the queue for processing. If any option for a subsequent pair is good, then the original must be as well.

7.7 Theorem. If Algorithm 7.1 terminates, then the returned result is correct. ◁ [7.6] (7.8) (7.16) (7.25)

Proof. We first prove that $VH = \left\{u_1, \ldots, u_k, v_1, \ldots, v_{|H|-k}\right\}$ for every pair (U, H) in the queue. Furthermore, we define the *path-decomposition of* (U, H) for such pairs, which is a smooth path-decomposition $(P, \mathcal{V})$ of H of width k with last bag U. For $(V_1, (V_1, \varnothing))$ the claim on the vertex set holds and we use a path of length 0 with bag V_1 as our decomposition. Now let (U', H') be a pair added in the iteration in which (U, H) was removed. As $VH' = VH \cup \{v_{i+1}\}$ for $i = |H| - k$ and $v_{i+1} \notin VH$, we get $|H'| = |H| + 1$ and VH' satisfies the claim. To obtain the decomposition of (U', H'), we extend the one of (U, H) by adding an additional vertex to the end of the path whose corresponding bag is U'. This decomposition has width k and is smooth as $|U'| = k+1$ and the bags U and U' differ in exactly one vertex.

Next, we prove that the results returned are correct if the algorithm terminates. If the algorithm returns a graph $H+E'$ in Line 9, then $H+E' \in \mathcal{G} \setminus \operatorname{super}\mathcal{U}$. It has path-width at most k, since the path-decomposition of (U, H) has width k with last bag U, which is also a path-decomposition for $H + E'$. Therefore, it suffices to check that $\mathcal{G}^k \subseteq \operatorname{super}\mathcal{U}$ in case the algorithm returns `None`.

We verify this by inductively proving the following invariant: before any iteration of the while loop, (7.1) holds. We have already argued that (7.1) holds for the initialisation of Q, by renaming the vertices and using Lemma 7.6 (c). If, before the first iteration, the single pair in Q is removed, then this pair is good and (7.1) holds. So it is true before the first iteration of the while loop.

Now assume that the invariant holds up to iteration l and consider the queue before iteration $l + 1$. In iteration l only a single element (U, H) was removed from Q. Consequently, it suffices to prove that (U, H) is good if all newly added pairs are. To verify this, let $G \in \mathcal{G}^k$ be a graph containing (U, H) with corresponding decomposition $(P, \mathcal{V})$. First assume that U is the last bag of this decomposition, that is,

$$G = H + E' \text{ for some } E' \subseteq \{xy \colon\ x,\, y \in U,\, x \neq y\}\,.$$

The algorithm considers this graph in some iteration of the for loop in Line 7. Since it does not terminate in this iteration and $G \in \mathcal{G}$, we have $G \in \operatorname{super}\mathcal{U}$.

Now assume that the path-decomposition of G does not end with the bag U. Let U' be the subsequent bag. We know that, for $i = |H| - k$, $VH = \{u_1, \ldots, u_k, v_1, \ldots, v_i\}$. Thus v_{i+1} is not in VH and we can assume that the element that enters U' is v_{i+1} by Lemma 7.6 (b). (Simply choose a mapping φ that is the identity restricted to H and maps the vertex that enters U' to v_{i+1}.) Denote the vertex leaving U by u, then the graph H' associated with the bag U' has the form

$$H + E' + v_{i+1} \text{ with } E' \subseteq \{uy \colon\ y \in U \setminus \{u\}\}\,.$$

Note that E' is the set of edges between u and its new neighbours. The algorithm considers the set $U' \coloneqq U \setminus \{u\} \cup \{v_{i+1}\}$ in the for loop in Line 11 and also looks at the graph H' in Line 13. Since $G \in \mathcal{G}^k$, it is an element of $\mathcal{G}$ that contains (U', H'). If $H' \in \operatorname{super}\mathcal{U}$, then $G \in \operatorname{super}\mathcal{U}$ and if it is not, then (U', H') is added to the queue. In this case, we have assumed it is a good pair and G contains it, meaning that G is in $\operatorname{super}\mathcal{U}$. □

The algorithm remains correct if we remove the check whether there exists a graph $G \in \mathcal{G}$ containing (U', H') in Line 15. It is used to reduce the number of pairs added to the queue. Should this condition be hard to check, it can be omitted. Heuristics may be used instead as long as they are correct in case they return a negative answer. For example, if $\mathcal{G}$ is the class of cubic graphs, we can check whether H' is subcubic.

Since we need it again later, we note that we obtained the following result in the proof.

[7.7] (7.16) (7.17) **7.8 Corollary.** For every pair (U, H) added to Q in Algorithm 7.1 there is a path-decomposition $(P, \mathcal{V})$ of H of width k whose last bag is $V_l = U$ and for which the associated graph satisfies that $H_l = H$. ◁

Proof. We showed the existence of this path-decomposition in the proof of Theorem 7.7, we just did not remark that the graph associated with the last bag is, in fact, H. But this holds since H only contains edges with at most one end in U. □

7.2. Isomorphism rejection

Now that we have proved the base algorithm to be correct, we demonstrate how to drastically improve the running time by exploiting isomorphism rejection, which reduces the number of pairs added in each iteration (see also Table 8.1 on Page 122).

[7.6] (7.10) **7.9 Lemma.** For a bijection φ with domain VH, the pair (U, H) is good if and only if the pair $(\varphi U, \varphi H)$ is good. ◁

Proof. Let $(\varphi U, \varphi H)$ be a good pair and G be a graph containing (U, H). Let $\overline{\varphi}$ be the extension of φ to VG, where $\overline{\varphi} v = v$ for all $v \in VG \setminus VH$. By Lemma 7.6 (b) the graph $\overline{\varphi} G$ contains the pair $(\varphi U, \varphi H)$, which is good. Hence, $\overline{\varphi} G \in \operatorname{super}\mathcal{U}$ and $\overline{\varphi} G \cong G$, which implies $G \in \operatorname{super}\mathcal{U}$. This shows that (U, H) is good. The missing direction follows by considering φ^{-1}. □

As a consequence of Lemma 7.9, we can improve our base algorithm: we only need to add pairs (U', H') to Q for which the queue does not already have an element of the form $(\varphi U', \varphi H')$ for some bijection φ.

What we describe now are special cases of Lemma 7.9 that can be checked before Line 15 of Algorithm 7.1. Assume we are in the iteration in which the pair (U, H) is removed from the queue. We denote the set of automorphisms of H that stabilise U as a set by $\operatorname{Aut}(U, H)$, that is, $\operatorname{Aut}(U, H)$ contains all automorphisms φ of H with $\varphi U = U$. For convenience, the automorphisms φ in $\mathrm{Aut}(U, H)$ are called (U, H)-*maps* and we note that $\mathrm{Aut}(U, H)$ is a subgroup of the automorphism group of G. We use these maps to eliminate certain pairs without needing to consider them. To facilitate this, the set $\mathrm{Aut}(U, H)$ is computed directly after the pair (U, H) is removed.

Algorithm 7.2: An optimised version of Algorithm 7.1 using isomorphism rejection.

Input: A class of graphs $\mathcal{G}$ with a membership test, a finite set of graphs $\mathcal{U}$, and a path-width value k.

Output: An element of $\mathcal{G}^k \setminus \mathrm{super}\,\mathcal{U}$ or `None` if no such graph exists.

```
1  def TestUnavoidability($\mathcal{G}$, $\mathcal{U}$, $k$):
2      $V_1 \leftarrow \{u_1, \ldots, u_k, v_1\}$
3      $Q \leftarrow [(V_1, (V_1, \varnothing))]$
4      if $(V_1, \varnothing)$ is good then $Q$.dequeue()
5      while $Q \neq \varnothing$ do
6          $(U, H) \leftarrow Q$.dequeue()
7          Determine $\mathrm{Aut}(U, H)$
8          $F \leftarrow \{xy\colon\ x, y \in U, x \neq y\}$ and $\mathcal{F} \leftarrow 2^F$
9          while $\mathcal{F} \neq \varnothing$ do
10             Remove a set $E'$ from $\mathcal{F}$
11             if $H + E' \in \mathcal{G} \setminus \mathrm{super}\,\mathcal{U}$ then
12                 return $H + E'$
13             else
14                 $\mathcal{F} \leftarrow \mathcal{F} \setminus \{\varphi E'\colon\ \varphi \in \mathrm{Aut}(U, H)\}$
15         $i \leftarrow |H| - k$
16         Let $\overline{U} \subseteq U$ contain a vertex from every orbit
17         for $u \in \overline{U}$ do
18             $U' \leftarrow U \setminus \{u\} \cup \{v_{i+1}\}$
19             $\mathcal{Y} \leftarrow 2^{U \setminus \{u\}}$
20             while $\mathcal{Y}$ is not empty do
21                 Remove a set $Y$ from $\mathcal{Y}$
22                 $H' \leftarrow H + v_{i+1} + \{uy\colon\ y \in Y\}$
23                 if $H' \notin \mathrm{super}\,\mathcal{U}$ and there exists a $G \in \mathcal{G}$ containing $(U', H')$ and
24                 there is no bijection $\varphi$ with $(\varphi U', \varphi H') \in Q$ then
25                     $Q$.append($(U', H')$)
26                 $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{\varphi Y\colon\ \varphi \in \mathrm{Aut}(U, H), \varphi u = u\}$
27     return None
```

Our goal is to optimise all three for loops in Lines 7, 11, and 13 of Algorithm 7.1 by reducing the number of potential counterexamples, candidates for subsequent bags, and associated graphs considered. See Algorithm 7.2 for the pseudocode of the algorithm with these additions.

First, we improve the for loop in Line 7 in which the algorithm looks for counterexamples.

If we have checked an edge set $E' \subseteq \{xy\colon\ x, y \in U, x \neq y\}$, then we do not need to check the sets $\varphi E'$ for $\varphi \in \mathrm{Aut}(U, H)$. This holds as

$$H + E' \cong \varphi(H + E') = H + \varphi E'.$$

Second, we reduce the number of candidates for the next bag in Line 11. By using (U, H)-maps, we can reduce the number of vertices that need to be considered. Let

$$\varphi \in \mathrm{Aut}(U, H) \text{ with } \varphi v = u \text{ for some } u, v \in U \text{ and}$$

let $\overline{\varphi}$ be the extension of φ to $VH \cup \{v_{i+1}\}$ where $\overline{\varphi}(v_{i+1}) = v_{i+1}$. We set

$$U' \coloneqq U \setminus \{u\} \cup \{v_{i+1}\} \text{ and } U'' \coloneqq U \setminus \{v\} \cup \{v_{i+1}\}\,.$$

By Lemma 7.9 we know that the pair (U'', H'') is good if and only if $(\overline{\varphi}\, U'', \overline{\varphi}\, H'')$ is, where $\overline{\varphi}\, U'' = U'$. Therefore, if, for every $Y' \subseteq U \setminus \{u\}$, the pair (U', H') with $H' \coloneqq H + v_{i+1} + \{uy\colon\ y \in Y'\}$ is good, then the same holds for the pairs (U'', H'') where $H'' \coloneqq H + v_{i+1} + \{vy\colon\ y \in Y''\}$ with $Y'' \subseteq U \setminus \{v\}$. To see this, let H'' be of the form above, then

$$\overline{\varphi}\, H'' = \overline{\varphi}\, H + \overline{\varphi}(v_{i+1}) + \overline{\varphi}(\{vy\colon\ y \in Y''\}) = H + v_{i+1} + \{uy\colon\ y \in \overline{\varphi}\, Y''\}\,.$$

But as $\overline{\varphi}\, Y'' \subseteq \overline{\varphi}(U \setminus \{v\}) = U \setminus \{u\}$, the graph $\overline{\varphi}\, H''$ is a candidate for H' and this pair is good by assumption.

This means that we only need to consider the case where u leaves, so the pairs (U', H') above, and can disregard the pairs with U'' altogether. In other words, it suffices to consider one vertex from each orbit (where the group $\mathrm{Aut}(U, H)$ acts on VH). We may thus replace the for loop iterating over all $u \in U$ by one that iterates over the set $\overline{U} \coloneqq \left\{u^{\mathrm{Aut}(U,H)}\colon\ u \in U\right\}$. Notice that elements of $\mathrm{Aut}(U, H)$ stabilise U as a set, so any representative of this orbit can be chosen.

Third, we reduce the number of vertex sets that are checked in Line 13. Assume the next bag is $U' = U \setminus \{u\} \cup \{v_{i+1}\}$. We show that we do not need to add a pair (U', H') for a set Y' if we have already added the pair (U', H'') for a set Y'' and there exists a $\varphi \in \mathrm{Aut}(U, H)$ with

$$\varphi Y' = Y'' \text{ and } \varphi u = u.$$

To see this, observe that the pair (U', H') is good if and only if the pair $(\overline{\varphi}\, U', \overline{\varphi}\, H')$ is, where $\overline{\varphi}$ is, again, the extension of φ to v_{i+1} with $\overline{\varphi}(v_{i+1}) = v_{i+1}$. By assumption we have that $\overline{\varphi}\, U' = U'$. We set

$$E' \coloneqq \{uy'\colon\ y' \in Y'\} \text{ and } E'' \coloneqq \{uy''\colon\ y'' \in Y''\}\,.$$

But, since

$$\overline{\varphi}(H' - E') = \overline{\varphi}(H + v_{i+1}) = H + v_{i+1} = H'' - E''$$

and $\overline{\varphi}\, E' = E''$, we get that $\overline{\varphi}\, H' = H''$. Consequently, we may ignore all sets $\varphi Y'$, for any map φ in the stabiliser $\mathrm{Aut}(U, H)_u = \{\varphi \in \mathrm{Aut}(U, H)\colon\ \varphi u = u\}$, upon adding the pair (U', H') in Line 13.

Since we have only made allowed modifications, the algorithm remains correct.

7.10 Theorem. Algorithm 7.2 is correct if it terminates. ◁ [7.9] (5.20)

7.11 Remark. By the orbit-stabiliser theorem, $|\mathrm{Aut}(U, H)| = |\mathrm{Aut}(U, H)_u| \cdot |u^{\mathrm{Aut}(U,H)}|$. Hence, assuming a reasonable size of $\mathrm{Aut}(U, H)$, we know that either the orbit is large, reducing the number of iterations of the for loop in Line 11, or the number of graphs that can be eliminated by our optimisation of Line 13 of Algorithm 7.1 is large. ◁

7.3. Achieving termination

So far, we have seen that our algorithm is correct *if* it terminates. We begin our discussion of termination by showing that our problem is generally undecidable, so the algorithm need not terminate. Afterwards, we determine restrictions for $\mathcal{G}$ and $\mathcal{U}$ that are sufficient to guarantee termination after all, at least after we make some modifications to the algorithm so it uses the additional structure we impose. As a result, we obtain the following theorem.

7.12 Theorem. Let $\mathcal{G}$ be a highly local graph class of bounded maximum degree, $\mathcal{U}$ a finite set of connected graphs, and $k \in \mathbb{N}$. There is an algorithm that decides whether $\mathcal{U}$ is unavoidable for $\mathcal{G}^k$ or not. In the latter case, the algorithm returns a counterexample of smallest order. ◁ [7.20]

Undecidability of the general problem. As announced, we start by proving that the problem of deciding whether a finite set of graphs is unavoidable for a graph class is undecidable.

7.13 Lemma. It is undecidable to determine whether a set of graphs $\mathcal{U}$ is unavoidable for a graph class $\mathcal{G}$ even if $\mathcal{U}$ is a finite set of connected graphs and $\mathcal{G}$ is a decidable class containing graphs of path-width at most 2 and of maximum degree at most 3. ◁

Proof (sketch). We describe a reduction from the undecidable Post correspondence problem Problem 2.17. Let $x_1, \dots, x_N$ and $y_1, \dots, y_N$ be strings over $\{a, b\}$. Let I be the set of finite sequences in $\{1, \dots, N\}$ and $f \colon \mathbb{N} \to I$ be an enumeration of I.

For $n \in \mathbb{N}$, let $P_n \coloneqq v_1 \dots v_n$ and set $G_n \coloneqq P_n$ if fn is a solution to the Post correspondence problem. Otherwise, set $G_n \coloneqq P_{n+2} + v_{n+2}v_n$. Hence, all G_n have path-width at most 2, maximum degree at most 3, and they contain a C_3 if and only if the corresponding fn is not a solution. We define $\mathcal{G} \coloneqq \{G_n \colon n \in \mathbb{N}\}$, which is decidable. Consequently, the set $\mathcal{U} \coloneqq \{C_3\}$ is unavoidable for $\mathcal{G}$ if and only if no path is in $\mathcal{G}$, which is equivalent to there being no solution to the Post correspondence problem. □

Modified algorithm for highly local classes. To achieve termination, we need to modify our algorithm in accordance with the new restrictions on the input. Assume that

- $\mathcal{G}$ is a highly local class with verifier $\mathcal{A}$ of radius r and size s,
- all graphs $G \in \mathcal{G}$ have maximum degree at most Δ,
- $\mathcal{U}$ contains only connected graphs, and
- $D \coloneqq \max\{\operatorname{diam} S\colon\ S \in \mathcal{U}\}$.

Note that we use $\mathcal{A}$ for the verifier here instead of $\mathcal{V}$ since the latter is already in use.

Before we modify our algorithm, let us prove a few basic properties. For a pair (U, H), where $U \subseteq VH$, and $d \in \mathbb{N}$ we set

$$L^d_{(U,H)} \coloneqq \{v \in H\colon\ d_H(v, U) > d\}\,.$$

First, we show that if we have not found an unavoidable structure in a pair, then vertices at large distance to the last bag are irrelevant for $\mathcal{U}$ being unavoidable.

(7.20) **7.14 Lemma.** Let G be a graph containing a pair (U, H). If $H \notin \operatorname{super}\mathcal{U}$, then for every $S' \subseteq G$ with $S' \cong S \in \mathcal{U}$ we have that

$$S' \subseteq G - L^D_{(U,H)}. \qquad \triangleleft$$

Proof. Let $S' \subseteq G$ with $S' \cong S \in \mathcal{U}$. Suppose $VS' \cap L^D_{(U,H)} \neq \varnothing$, then there exists a vertex $v \in S'$ such that $d(v, U) > D$. Hence, every vertex $u \in S'$ satisfies

$$d(u, U) \geq d(v, U) - d(u, v) > 0$$

and, therefore, $VS' \cap U = \varnothing$. Since U separates $H - U$ and $G - VH$ in G and S' is connected, we get that $S' \subseteq H - U$, contradicting that $H \notin \operatorname{super}\mathcal{U}$. Hence, $S' \subseteq G - L^D_{(U,H)}$. $\square$

Next we prove a similar result for vertices that are needed to check containment in the class $\mathcal{G}$.

(7.16)
(7.20) **7.15 Lemma.** Let (U, H) be a pair, $G \in \mathcal{G}$ be a graph containing (U, H), and $\mathbf{P}$ be a proof for G. We have that

- for every $v \in L^r_{(U,H)}$ the verifier satisfies

$$\mathcal{A}(G, \mathbf{P}, v) = \mathcal{A}(H, \mathbf{P}[VH], v) \text{ and}$$

- for every $v \in VG \setminus L^r_{(U,H)}$ and $L \subseteq L^{2r}_{(U,H)}$ we obtain

$$\mathcal{A}(G, \mathbf{P}, v) = \mathcal{A}(G - L, \mathbf{P}[VG \setminus L], v). \qquad \triangleleft$$

Proof. Let $v \in H$ and $B \coloneqq B^r_G\, v$. If $v \in L^r_{(U,H)}$ it follows that $d(v,U) > r$ and, therefore, $B \subseteq H - U = G[VH \setminus U]$ since U separates $H - U$ and $G - VH$. Consequently, we have that

$$\begin{aligned}\mathcal{A}(G, \mathbf{P}, v) &= \mathcal{A}(G[B], \mathbf{P}[B], v)\\ &= \mathcal{A}(H[B], \mathbf{P}[B], v)\\ &= \mathcal{A}(H, \mathbf{P}[VH], v).\end{aligned}$$

Otherwise, if $v \in VG \setminus L^r_{(U,H)}$, then $v \in H$ and $d(v,U) \leq r$ or $v \notin H$. This implies that $B \subseteq G - L$ and we get

$$\begin{aligned}\mathcal{A}(G, \mathbf{P}, v) &= \mathcal{A}(G[B], \mathbf{P}[B], v)\\ &= \mathcal{A}((G - L)[B], \mathbf{P}[B], v)\\ &= \mathcal{A}(G - L, \mathbf{P}[VG \setminus L], v).\end{aligned}$$ □

Before we make adjustments, let us give an outline of how we intend to proceed.

(1) Algorithm 7.1 on Page 100 is extended from pairs (U, H) to triples $(U, H, \mathbf{P})$ that also contain a proof for H. We also adjust the checking of containment in $\mathcal{G}$ and introduce a heuristic to exclude new triples from being added to the queue.
(2) The isomorphism rejection is reintroduced, in a stronger version. Instead of considering the entire graph H' when applying a bijection, we show that we may forget about vertices that are too far away from U' to be relevant.

Let us start with Step (1), in which we work towards Algorithm 7.3. As a first step, let us generalise our notation from Section 7.1 to triples in a straightforward manner. A graph G *contains a triple* $(U, H, \mathbf{P})$ if it contains the pair (U, H) and there exists a proof $\overline{\mathbf{P}}$ for G that $\mathcal{A}$ accepts and which satisfies $\overline{\mathbf{P}}[VH] = \mathbf{P}$. Such a triple is *good* if any graph $G \in \mathcal{G}^k$ containing it is in super$\mathcal{U}$.

We now make the following alterations to Algorithm 7.1 resulting in Algorithm 7.3.

- The queue is initialised with all triples $(V_1, (V_1, \varnothing), \mathbf{P})$ in Line 3 where $\mathbf{P}$ is a proof for $(V_1, \varnothing)$, so some function $\mathbf{P}\colon V_1 \to \{0,1\}^s$.
- The verification of $H + E' \in \mathcal{G}$ when checking for counterexamples is replaced by checking that $\mathcal{A}$ accepts $\mathbf{P}$ at all vertices in H, see Line 8.
- We create all triples $(U', H', \mathbf{P}')$ from $(U, H, \mathbf{P})$ in Lines 11 to 16 where $\mathbf{P}'$ is a proof for H', that is, $\mathbf{P}'\colon VH' \to \{0,1\}^s$, with $\mathbf{P}'[VH] = \mathbf{P}$.
- We verify that $\mathcal{A}$ accepts $\mathbf{P}'$ at all vertices $v \in L^r_{(U',H')}$ in H' before adding the new triple $(U', H', \mathbf{P}')$ to the queue, see Line 17.

We note that thanks the heuristic specified in Line 17, it suffices to check that the verifier accepts $\mathbf{P}$ at the vertices of $H - L^r_{(U,H)}$ when checking for counterexamples, instead of checking all vertices of H as done in Line 8.

We now need to verify that this has not hurt the correctness.

Algorithm 7.3: Modified base algorithm for checking whether $\mathcal{G}^k \subseteq \operatorname{super}\mathcal{U}$.

Input: A highly local class of graphs $\mathcal{G}$ with bounded maximum degree, a finite set of connected graphs $\mathcal{U}$, and a path-width value k. Here, $\mathcal{A}$ is a verifier for $\mathcal{G}$ and s is the size of the prover.

Output: An element of $\mathcal{G}^k \setminus \operatorname{super}\mathcal{U}$ or `None` if no such graph exists.

```
def TestUnavoidability($\mathcal{G}$, $\mathcal{U}$, $k$):
    $V_1 \leftarrow \{u_1, \ldots, u_k, v_1\}$
    $Q \leftarrow [(V_1, (V_1, \varnothing), \mathbf{P})\colon \mathbf{P}$ is a proof for $(V_1, \varnothing)]$
    Remove all good triples from $Q$
    while $Q \neq \varnothing$ do
        $(U, H, \mathbf{P}) \leftarrow Q$.dequeue()
        for $E' \subseteq \{xy\colon x, y \in U, x \neq y\}$ do
            if $H + E' \notin \operatorname{super}\mathcal{U}$ and $\mathcal{A}(H + E', \mathbf{P}, v) = 1$ for all $v \in H$ then
                return $H + E'$
        $i \leftarrow |H| - k$
        for $u \in U$ do
            $U' \leftarrow U \setminus \{u\} \cup \{v_{i+1}\}$
            for $Y \subseteq U \setminus \{u\}$ do
                $H' \leftarrow H + v_{i+1} + \{uy\colon y \in Y\}$
                for $p \in \{0,1\}^s$ do
                    Let $\mathbf{P}'$ be the proof for $H'$ with $\mathbf{P}'[VH] = \mathbf{P}$ and $\mathbf{P}(v_{i+1}) = p$
                    if $H' \notin \operatorname{super}\mathcal{U}$ and $\mathcal{A}$ accepts $\mathbf{P}'$ at all vertices of $L^r_{(U',H')}$ then
                        $Q$.append($(U', H', \mathbf{P}')$)
    return None
```

[7.7] [7.8] [7.15] **7.16 Theorem.** The result returned by Algorithm 7.3 is correct if it terminates. ◁

Proof. This proof proceeds similarly to the one for Theorem 7.7. We know that Algorithm 7.1 only adds pairs (U, H) for which a path-decomposition of H exists that has width k and last bag U, by Corollary 7.8. This remains true here, since we only added a third component to these pairs, and it lets us conclude that a graph $H + E'$ returned in Line 9 has path-width at most k. It is also not in $\operatorname{super}\mathcal{U}$ and because the verifier accepts $\mathbf{P}$ at all vertices, it must be in $\mathcal{G}$, making it a counterexample as desired.

So again, we focus on the case that `None` is returned, for which we show the analogue of invariant (7.1) on Page 100.

$$\text{If all triples in the queue are good, then } \mathcal{G}^k \subseteq \operatorname{super}\mathcal{U}. \tag{7.2}$$

This holds initially: we add all possible proofs to the queue, so any graph in $\mathcal{G}$ contains a triple $(V_1, (V_1, \varnothing), \mathbf{P})$ for an appropriately chosen $\mathbf{P}$. In the iteration where $(U, H, \mathbf{P})$ is removed, we just need to show that it is good if all newly added triples are. So, once more, let $G \in \mathcal{G}^k$ be a graph containing $(U, H, \mathbf{P})$ with a path-decomposition having U as a bag. If U is the last bag, then G is in $\operatorname{super}\mathcal{U}$ since the verifier accepts $\mathbf{P}$ on $G = H + E'$ by assumption.

Otherwise, let U' be the next bag. As before

$$U' := U \setminus \{u\} \cup \{v_{i+1}\}, \; H' := H + E' + v_{i+1},$$

and both are considered by the algorithm. Since G contains $(U, H, \mathbf{P})$, it has a proof $\overline{\mathbf{P}}$ that $\mathcal{A}$ accepts and which satisfies $\overline{\mathbf{P}}[VH] = \mathbf{P}$. Hence G also contains the pair $(U', H', \mathbf{P}')$ where $\mathbf{P}' := \overline{\mathbf{P}}[VH']$. If $H' \in \operatorname{super}\mathcal{U}$, then we are done and, otherwise, we claim that $(U', H', \mathbf{P}')$ is added to the queue. This is true since it passes the additional condition we imposed in Line 17 by Lemma 7.15: the verifier accepts at these vertices since it does so for G and the results coincide. □

Before we proceed with Step (2), let us make a few easy observations about the current version that will serve us well shortly.

7.17 Observation. Every triple $(U, H, \mathbf{P})$ added to the queue satisfies that [7.8] (7.20)

- $H' \notin \operatorname{super}\mathcal{U}$,
- there exists a smooth path-decomposition $(Q, \mathcal{W})$ of H with last bag U whose associated graph is H, and
- $\mathcal{A}(H, \mathbf{P}, v) = 1$ for all $v \in L^r_{(U,H)}$. ◁

Proof. These first and last property follow directly from Line 17 and the second is just the result of Corollary 7.8, which carries over to Algorithm 7.3. □

Now that we have successfully extended our algorithm to triples, it is time to deal with Step (2). We shall only introduce the generic elimination technique, taking the place of Lemma 7.9. Note that the direct analogue of this lemma is easily obtained but sadly insufficient to guarantee termination. However, since we need it later, we prove it now to familiarise ourselves with the new terminology.

7.18 Lemma. For a bijection φ with domain VH, the triple $(U, H, \mathbf{P})$ is good if and only if the triple $(\varphi U, \varphi H, \varphi \mathbf{P})$ is. ◁ [7.6] (7.20)

Proof. Let $(\varphi U, \varphi H, \varphi \mathbf{P})$ be a good triple and G be a graph containing $(U, H, \mathbf{P})$. Let $\overline{\mathbf{P}}$ be a proof on G that $\mathcal{A}$ accepts with $\overline{\mathbf{P}}[VH] = \mathbf{P}$ and let $\overline{\varphi}$ be the extension of φ to VG, where $\overline{\varphi} v = v$ for all $v \in VG \setminus VH$.

By Lemma 7.6 (b) the graph $\overline{\varphi} G$ contains the pair $(\varphi U, \varphi H)$, $\mathcal{A}$ accepts $\overline{\varphi}\overline{\mathbf{P}}$ on $\overline{\varphi} G$ by definition of highly local, and $\overline{\varphi}\overline{\mathbf{P}}[V(\varphi H)] = \varphi \mathbf{P}$. Therefore, $\overline{\varphi} G$ contains the triple $(\varphi U, \varphi H, \varphi \mathbf{P})$ and, as this triple is good, $\overline{\varphi} G \in \operatorname{super}\mathcal{U}$. Hence, $\overline{\varphi} G \cong G$, $G \in \operatorname{super}\mathcal{U}$, and $(U, H, \mathbf{P})$ is good. The missing direction follows by considering φ^{-1} again. □

The stronger variant we need for termination allows us to neglect the parts of H that are ‘too far away’ from U. As Lemmas 7.14 and 7.15 suggest, all vertices in H whose distance to U exceeds both $2r$ and D are no longer relevant for the algorithm. They are neither helpful for finding subgraphs in $\mathcal{U}$ nor do we need to check whether the verifier accepts at these vertices, since we already know that it does. We formalise this observation in the following definition.

7.19 Definition. Let $(U, H, \mathbf{P})$ and $(U', H', \mathbf{P}')$ be two triples. Furthermore, let

$$d \coloneqq \max\{2r, D\}, \ L \coloneqq L^d_{(U,H)}, \text{ and } L' \coloneqq L^d_{(U',H')}.$$

A *restricted correspondence of two triples* $(U, H, \mathbf{P})$ *and* $(U', H', \mathbf{P}')$ is a bijective map $\varphi\colon\ VH \setminus L \to VH' \setminus L'$ satisfying

$$\varphi U = U', \ \varphi(H - L) = H' - L', \text{ and } \varphi\,\mathbf{P}[VH' \setminus L'] = \mathbf{P}'[VH' \setminus L']. \qquad \triangleleft$$

Our goal is to show that we can eliminate triples if there is a restricted correspondence between them. That this is allowed is implied by the theorem below.

[7.14] [7.15] [7.17] [7.18] (7.12) **7.20 Theorem.** Let $(U, H, \mathbf{P})$ and $(U', H', \mathbf{P}')$ be two triples. Assume that

- (U, H) was added to the queue by the algorithm and
- there exists a restricted correspondence φ of $(U, H, \mathbf{P})$ and $(U', H', \mathbf{P}')$.

If $(U, H, \mathbf{P})$ is good, then so is $(U', H', \mathbf{P}')$. $\triangleleft$

Proof. Let d, L, and L' be as in the definition of restricted correspondence above and let G' be a graph containing $(U', H', \mathbf{P}')$. By definition, we obtain a path-decomposition $(P', \mathcal{V}')$ of G' with $P' = 1 \ldots n'$ such that $V'_j = U'$ and $G'_j = H'$ for some j. Additionally, we know that there is a proof $\overline{\mathbf{P}'}$ such that $\mathcal{A}(G', \overline{\mathbf{P}'}, v) = 1$ for all $v \in G'$ and $\overline{\mathbf{P}'}[VH'] = \mathbf{P}'$.

We now want to use that $(U, H, \mathbf{P})$ is good to show that G' is in super$\mathcal{U}$. Intuitively, we achieve this by replacing H' (the left part of G') by $\varphi\,H$ and showing that the new graph G obtained this way is still in $\mathcal{G}^k$ and contains the triple $(\varphi\,U, \varphi\,H, \varphi\,\mathbf{P})$, which is good by Lemma 7.18. With these two properties, G must then be in super$\mathcal{U}$. But since the subgraphs in $\mathcal{U}$ cannot be contained in the left part of G, they must be on the right, which is where G concides with G'.

To obtain the graph described above, we first extend the bijection φ to H by choosing the images of L such that $\varphi\,L \cap VG' = \varnothing$ and then define

$$G \coloneqq \varphi\,H \cup (G' - L').$$

We see that

- $\varphi(VH) \cap V(G' - L') = V(H' - L')$,
- VG is the disjoint union of $\varphi(VH)$ and $V(G' - VH')$, and
- $G - \varphi\,L = G' - L'$.

Now that we have the graph we want, we can go about proving the two desired properties. To this end, we construct a smooth path-decomposition of G of width k that contains $\varphi\,U$ as a bag with associated graph $\varphi\,H$. Let $(Q, \mathcal{W})$ be a smooth path-decomposition of H

with last bag $W_l = U$, which exists by Observation 7.17. Define $P \coloneqq 1 \dots n' + l - j$ and let $\mathcal{V}$ contain the sets

$$V_i \coloneqq \begin{cases} \varphi\, W_i & \text{for } i \in \{1, \dots, l\}\,, \\ V'_{i+j-l} & \text{for } i \in \{l+1, \dots, l+n'-j\}\,. \end{cases}$$

Notice that $V_l = \varphi\, U = U' = V'_j$, giving us a smooth decomposition of width k if it is a path-decomposition. Moreover, $G_l = \varphi\, H$, so G contains the pair $(\varphi\, U, \varphi\, H)$.

We now verify the properties of a path-decomposition to get that G has path-width at most k and contains $(\varphi\, U, \varphi\, H)$. Let $v \in G$. If $v \in \varphi\, H$ then $v \in \varphi\, W_i = V_i$ for some $i \leq l$ and, otherwise, $v \in G' - V H'$ and v enters a bag V'_i for $i > j$, placing it in V_{i+l-j}. For an edge $uv \in E\, G$ we get that uv is covered by a bag $V_i = \varphi\, W_i$ with $i < l$ if u is in $\varphi(H - U)$, as it leaves one of the first $l-1$ bags. If this is not the case, we may assume that both u and v are in $(G' - V H') \cup U'$. This means that uv is an edge of G' and covered by a bag V'_i with $i \geq j$. Hence, uv is covered by the bag V_{i+l-j}. Finally, note that the set $\{i\colon\; v \in V_i\}$ can be written as

$$\{i \leq l\colon\; v \in \varphi\, W_i\} \cup \left\{i \geq l\colon\; v \in V'_{i+j-l}\right\},$$

making it the union of two paths. If both sets are non-empty, then $v \in \varphi(V H)$ as well as $v \in (V G' \setminus V H') \cup U'$, so $v \in U'$ and l is in both sets. Thus, $(P, \mathcal{V})$ is a path-decomposition.

We now verify that $G \in \mathcal{G}$ by exhibiting a proof $\overline{\mathbf{P}}$ for G that $\mathcal{A}$ accepts and which satisfies $\overline{\mathbf{P}}[\varphi\, H] = \varphi\, \mathbf{P}$. This shows that $G \in \mathcal{G}$ and that G contains the triple $(\varphi\, U, \varphi\, H, \varphi\, \mathbf{P})$. We define

$$\overline{\mathbf{P}}\, v \coloneqq \begin{cases} \varphi\, \mathbf{P}\, v & \text{if } v \in \varphi\, H, \\ \overline{\mathbf{P}'}\, v & \text{otherwise.} \end{cases}$$

Note that $\overline{\mathbf{P}}[V H' \setminus L'] = \varphi\, \mathbf{P}[V H' \setminus L'] = \mathbf{P}'[V H' \setminus L']$, from which we can deduce that $\overline{\mathbf{P}}[V G \setminus \varphi\, L] = \overline{\mathbf{P}}[V G' \setminus L'] = \overline{\mathbf{P}'}[V G' \setminus L']$. Using Lemma 7.15, we conclude that

$$\mathcal{A}(G, \overline{\mathbf{P}}, v) = \begin{cases} \mathcal{A}(\varphi\, H, \varphi\, \mathbf{P}, v) = \mathcal{A}(H, \mathbf{P}, \varphi^{-1}\, v) & \text{if } v \in L^r_{(\varphi\, U, \varphi\, H)}, \\ \mathcal{A}(G - \varphi\, L, \overline{\mathbf{P}}[V G \setminus \varphi\, L], v) = \mathcal{A}(G', \overline{\mathbf{P}'}, v) & \text{otherwise} \end{cases}$$

and in both cases $\mathcal{A}(G, \overline{\mathbf{P}}, v) = 1$.

In summary, G contains $(\varphi\, U, \varphi\, H, \varphi\, \mathbf{P})$, which is a good triple by Lemma 7.18. Hence, we can find a subgraph $S' \subseteq G$ with $S' \cong S \in \mathcal{U}$. By Lemma 7.14 we know that

$$S' \subseteq G - L^D_{(\varphi\, U, \varphi\, H)} \subseteq G - \varphi\, L = G' - L' \subseteq G'$$

and the triple $(U', H', \mathbf{P}')$ is good. $\square$

As a consequence of Theorem 7.20, we can augment Algorithm 7.3 by the condition that triples $(U', H', \mathbf{P}')$ are only added to Q in Line 18 if no restricted correspondence of $(U', H', \mathbf{P}')$ and a previously found triple exists.

Note that we only needed to check the queue in Algorithm 7.2, but this was due to the orders being increasing. This does not remain valid here, since we no longer need bijections on all of H.

This is now sufficient to guarantee termination.

Proof (of Theorem 7.12). Now that we eliminate triples which have a restricted correspondence to a triple previously added to the queue, we have limited the number of triples that can be added to the queue to a finite number. That this is allowed follows from Theorem 7.20. To see that only finitely many triples are added, we note that the bound Δ on the maximum degree of the graphs in $\mathcal{G}$ gives us a bound on the number of vertices in $H - L^d_{(U,H)}$. Hence, the number of such graphs is also constant, as is the number of choices for U, and the potential proofs on $H - L^d_{(U,H)}$, up to restricted correspondence. □

Running time analysis. We analyse the running time of our algorithm to show that it is elementary and, in particular, it is better than the running time obtained by Courcelle's theorem. Since this is our main objective, the estimations will be coarse and unoptimised.

The iterations of the outer loop in Line 5 are bounded by the number of triples $(U, H, \mathbf{P})$ we add to the queue. We had already seen that there are only finitely many of these and we now put a number to these amounts, which we neglected earlier. The graphs $H - L$ have at most

$$y \coloneqq (k+1)\Delta^{d+1}$$

vertices, bounding their number by $y2^{y^2}$. Here, and from here on, L denotes $L^d_{(U,H)}$. The set U is part of $H - L$, so there are at most $\binom{y}{k+1}$ possible choices for it. This just leaves the proofs on $H - L$, of which up to 2^{sy} different ones can exist. Thus, at most

$$x \coloneqq y2^{y^2}\binom{y}{k+1}2^{sy}$$

triples are added to the queue, which also bounds these iterations.

The cost of checking whether $H + E'$ has a subgraph in $\mathcal{U}$ in Line 8 can be bounded by $\mathcal{O}\left(|\mathcal{U}|y!y^2\right)$: simply try all possible locations of $S \in \mathcal{U}$ in $(H + E') - L$ and check whether they are correct. The loop in Line 7 goes through at most $2^{(k+1)^2}$ iterations. If we denote the time it takes to run the verifier $\mathcal{A}$ by $T\mathcal{A}$, we get a running time of $\mathcal{O}\left(yT\mathcal{A} + |\mathcal{U}|y!y^2\right)$ for each iteration of this loop. Here we did make use of the previously mentioned optimisation that checking the vertices $v \in H - L^r_{(U,H)}$ suffices.

Now we get to the next three nested loops. The one in Line 11 goes through at most $k+1$ iterations, the one in Line 13 brings another 2^k to the table, and the final loop in Line 15 comes with 2^s many. In these nested loops, the creation of $\mathbf{P}'$ is a cheap operation, but checking that $H \notin \operatorname{super}\mathcal{U}$ comes at cost $\mathcal{O}(|\mathcal{U}|y!y^2)$ again. Running the verifier on all elements of $L^r_{(U',H')}$ sounds like it could be expensive, but we need not do this for vertices also in $L^r_{(U,H)}$. Therefore, we can bound the time required for this generously by $\mathcal{O}(y\,T\mathcal{A})$.

This just leaves the time required to determine whether there exists a restricted correspondence of $(U', H', \mathbf{P}')$ and some previously found triple. For this, we have to compare the current triple to at most the $\mathcal{O}(x)$ many found before. For each we can try the $\mathcal{O}(y!)$ bijections φ to see whether they are a restricted correspondence, where checking takes $\mathcal{O}(y^2)$ time.

We use this opportunity to note that $T\mathcal{A}$ cannot be arbitrarily bad. Since the verifier is only ever faced with a finite number of local views, we can implement it as a lookup table. The bound on the number of views it can ever see is obtained in analogy to the bound x for the triples, and strictly smaller, since fewer vertices are present. Once we have the lookup table, we only need to analyse the neighbourhood to see which value we must return, letting us (generously) assume that $T\mathcal{A} \in \mathcal{O}(y^2)$.

Altogether, by collecting everything, we get a total running time of

$$\mathcal{O}\left(x \cdot \left(2^{(k+1)^2} \cdot \left(y\,T\mathcal{A} + |\mathcal{U}|y!y^2\right) + (k+1) \cdot 2^k \cdot 2^s \cdot \left(y\,T\mathcal{A} + |\mathcal{U}|y!y^2 + xy!y^2\right)\right)\right),$$

which we can simplify to

$$\mathcal{O}\left(x^2 y! y^2 \cdot 2^{(k+2)^2+s}\right).$$

Necessity of the adjustments to the algorithm. To wrap up this section, we show that our adjustments to the algorithm are necessary to achieve termination. To this end, we provide examples in which the algorithm does not terminate and that adhere to all but one of our additional conditions.

If we drop the highly local requirement, we are back in a case where the problem is undecidable and termination cannot be guaranteed.

If we remove the requirement on bounded maximum degree, we can still use our modified algorithm, since we only needed the bound on the degree to estimate the number of triples that are added to the queue. But there can now be infinitely many such triples, as illustrated by the following example. Regard the class $\mathcal{G}$ which contains all graphs that consist of two stars connected by a single edge between their centre vertices. Every graph in this class has diameter 3, making it locally checkable. By choosing $k = 1$ and letting the set $\mathcal{U}$ consist of a path of length 3, we obtain an example for which the algorithm does not terminate. Every path-decomposition of a graph $G \in \mathcal{G}$ starts by covering the edges in one of the two stars, followed by the edge connecting it to the second, and ending with the edges of this star. Consequently, the path of length 3 can appear arbitrarily late in the path-decomposition.

A similar problem occurs when one drops the connectivity requirement for the graphs in $\mathcal{U}$. This is what guarantees a finite diameter in the proof, so again this makes infinitely many triples possible. An example of this can be obtained as follows: let the class $\mathcal{G}$ contain those graphs which are the union of a path with two triangles where the each of the triangles intersects the path in one of its ends. We set $\mathcal{U} := \{2K_3\}$ and $k = 2$. Again, every path-decomposition starts with one triangle and then traverses the path and ends at the other, meaning that the $2K_2$ can appear arbitrarily late. This is an example of a class with bounded maximum degree and it is also locally checkable: roughly speaking, a proof simply accepts at degree 2 vertices and the degree 3 ones must see a triangle.

7.4. Tailoring the algorithm to cubic graphs

We begin this section by proving that we can make strong assumptions on the path-decompositions of (cubic) graphs without loss of generality. These will then serve us well for both tailoring our algorithm to cubic graphs and for proving results on the girth of cubic graphs in the next chapter.

(7.22) (7.24) (8.14) (8.15) **7.21 Lemma.** Let $(P, \mathcal{V})$ be a smooth path-decomposition of G and $i \in \{1, \dots, n' - 1\}$. If $N_G\, v \subseteq VG_i$ for some $v \in V_i$, then there exists a smooth path-decomposition $(P, \mathcal{W})$ of G such that

$$V_j = W_j \text{ for } j \leq i \text{ and } v \text{ leaves } W_i. \quad \triangleleft$$

Proof. Let $v \in V_i$ be such a vertex and assume that some other vertex $u \neq v$ leaves V_i. Replacing v by u in all bags of index greater than i yields another path-decomposition $(P, \mathcal{W})$ of G as all edges with v as an end are already covered by the bags up to V_i. It is also smooth, satisfies $V_j = W_j$ for $j \leq i$, and v leaves the bag W_i as required. □

[7.21] (7.24) (7.25) (8.14) (8.15) **7.22 Lemma.** Let $(P, \mathcal{V})$ be a smooth path-decomposition of G, let $i \in \{1, \dots, n' - 1\}$. If G_i contains a vertex v with $|N_G\, v \setminus N_{G_i}\, v| \leq 1$, then there is a smooth path-decomposition $(P, \mathcal{W})$ of G with

$$V_j = W_j \text{ for } j < i,\ V_{i-1} \setminus V_i = W_{i-1} \setminus W_i, \text{ and } v \text{ leaves } W_i. \quad \triangleleft$$

Proof. Let v be as in the statement and assume that $u \neq v$ is the vertex leaving V_i. If $N_G\, v \subseteq VG_i$, then we apply Lemma 7.21 to obtain the claim.

Hence, we assume there exists a vertex $w \in N_G\, v \setminus VG_i$ and let V_j be the bag of lowest index containing w. By assumption, we have $j > i$ and Lemma 7.21 lets us assume that v leaves V_j or V_j is the last bag. We describe how to obtain the desired path-decomposition $(P, \mathcal{W})$ in case the bag V_{j+1} exists, making note of what would change if it does not in parentheses. The process is illustrated in Figure 7.2.

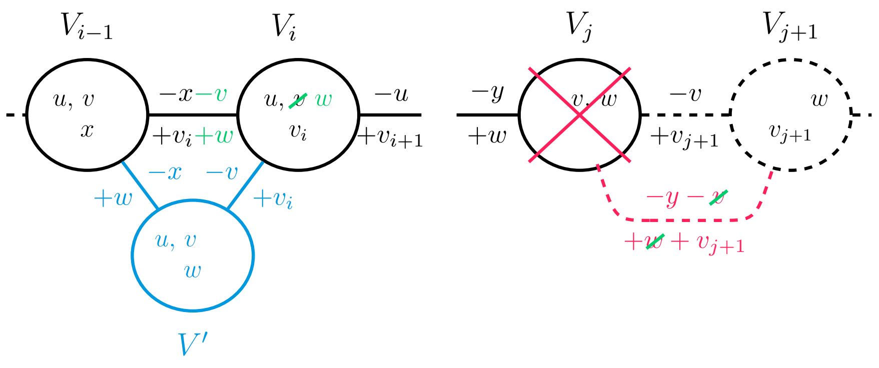


Figure 7.2.: The path-decomposition constructed in the proof of Lemma 7.22. First, the bag V_j is deleted, in the red step, resulting in the red modifications. Then v is replaced by w, as marked in green, and finally, the blue bag V' is inserted.

First, we delete the bag V_j, connecting V_{j-1} to V_{j+1} (if it exists). Note that if a bag uniquely covers an edge of G, then the vertex entering and the one leaving it are an end of this edge, otherwise the bag before or after would have done this as well. Therefore, the result is a path-decomposition of $G - vw$ $(G - w)$. This step is marked in red in Figure 7.2.

Next, we replace all occurrences of v in the bags $V_i, \dots, V_{j-1}$ by w, which is marked in green in Figure 7.2. As all these bags contain v but do not contain w, so they continue to have $k+1$ elements. Also the now neighbouring bags V_{j-1} and V_{j+1} have k vertices in common. Since we only removed v from bags and all its neighbours but w already shared a bag before V_i, this is a path-decomposition of $G - vw$ (now also in the case that V_j was the last bag, where V_{j-1} contains all neighbours of w).

Finally, we insert the bag $V' = V_{i-1} \cup \{w\} \setminus \{x\}$, where x is the vertex in $V_{i-1} \setminus V_i$, between V_{i-1} and V_i to make the decomposition smooth and turn it into a path-decomposition of G as this bag contains both v and w, see the blue part of Figure 7.2. This completes the construction and the proof. □

By iteratively applying this lemma to a smooth path-decomposition of a cubic graph G, we may assume that the vertex leaving bag V_i has degree at least 2 in G_i, whenever such a vertex exists. If, additionally, a degree 3 vertex is chosen to leave whenever present, we obtain a decomposition as defined below.

7.23 Definition. Let $(P, \mathcal{V})$ be a path-decomposition of a cubic graph G as in Convention 7.1 and, for $i \in \{1, \dots, n'\}$, let

$$d_i \coloneqq \max\{d_{G_i} v\colon\ v \in V_i\}\,.$$

Then $(P, \mathcal{V})$ is called *high-degree-first (hdf)* or an *hdf-decomposition* if the vertex v_i leaving the ith bag satisfies

$$d_{G_i} v_i = 3 \text{ if } d_i = 3 \text{ and } d_{G_i} v_i = 2 \text{ if } d_i = 2. \qquad \triangleleft$$

Such hdf-decompositions always exist.

[7.21] [7.22] **7.24 Theorem.** Let G be a cubic graph of path-width k. Then G has an hdf-decomposition of width k. ◁

Proof. The theorem follows from iteratively applying Lemmas 7.21 and 7.22 as outlined above. □

Aside from making use of general properties of cubic graphs, such as that $|G|$ is even and $\|G\| = \frac{3}{2}|G|$, hdf-decompositions are immensely helpful in speeding up the algorithm. More precisely, when considering a pair (U, H), Lemma 7.22 lets us choose the vertex to leave U if H has a vertex of degree at least 2, eliminating the need to iterate over $\overline{U}$ entirely.

[7.7] [7.22] (5.20) **7.25 Theorem.** Instead of iterating over the set $\overline{U} \coloneqq \left\{u^{\mathrm{Aut}(U,H)} \colon\ u \in U\right\}$, it suffices to choose a single vertex of degree at least 2, if such a vertex is present, in the case that $\mathcal{G}$ contains only cubic graphs. ◁

Proof. To see that this holds, recall the proof of Theorem 7.7. There we proved the invariant that $\mathcal{G}^k \subseteq \operatorname{super}\mathcal{U}$ if all pairs in the queue are good. More precisely, we showed that in the iteration where we remove the pair (U, H), this pair is good if all the newly added ones are. Consequently, we need to show that a pair (U, H) is already good if we only add pairs of form (U', H') for which $U' = U \setminus \{u\} \cup \{v_{i+1}\}$ for some vertex u of degree at least 2 in H. If H has no such vertex, the algorithm remains unchanged.

In this situation, where u and U' are defined as above, let $G \in \mathcal{G}^k$ be a graph containing (U, H) with corresponding path-decomposition $(P, \mathcal{V})$ and $U = V_i$. Assume, without loss of generality, that v_j is the vertex entering bag j for $j \in \{2, \dots, i+1\}$. We know that $G \in \operatorname{super}\mathcal{U}$ if U is the last bag of the decomposition, so we may assume there is a subsequent bag V_{i+1}. By applying Lemma 7.22, we get a smooth path-decomposition $(P, \mathcal{W})$ of G such that $V_j = W_j$ for all $j < i$, $V_{i-1} \setminus V_i = W_{i-1} \setminus W_i$, and u leaves W_i. Let φ be a bijection on VG with $\varphi_{|V(G_{i-1})} = \mathrm{id}_{V(G_{i-1})}$ that maps the unique vertices entering W_i and W_{i+1} to v_i and v_{i+1}. This gives us that φG contains the pair $(\varphi(W_{i+1}), \varphi(H_{i+1}))$ where H_{i+1} is the graph associated with W_{i+1}. Since the vertices leaving V_{i-1} and W_{i-1} coincide and $\varphi u = u$, we get that $\varphi W_i = U$ and $\varphi(W_{i+1}) = U'$. Note that $\varphi u = u$ follows from the fact that $d_H u > 0$, which implies that it is not the vertex entering V_i.

The graph $H' \coloneqq \varphi(H_{i+1})$ is of the form $H + v_{i+1} + \{uy \colon\ y \in Y\}$, where $Y \subseteq U \setminus \{u\}$. If $H' \in \operatorname{super}\mathcal{U}$, then $G \in \operatorname{super}\mathcal{U}$ and we may assume this is not the case. As the set Y is in $\mathcal{Y}$, it is considered by the algorithm and added to the queue unless there is already an element $(\psi U', \psi H')$ contained in it. In either case, the pair (U', H') is good by assumption and thus both φG and G are in $\operatorname{super}\mathcal{U}$. □

7.5. Unavoidable minors and induced subgraphs

Our algorithm can easily be extended to a test for unavoidable minors or induced subgraphs. We only need to adapt the definition of super$\mathcal{U}$ to denote the set that contains all graphs with a minor or an induced subgraph in $\mathcal{U}$. In these cases a pair (U, H) is good if H has a minor in $\mathcal{U}$, respectively if $H - U$ has an induced subgraph in $\mathcal{U}$ since this subgraph is inevitably induced every graph G containing (U, H). Our termination proof carries over to the induced subgraph case, without further modification. Note that this is due to Lemma 7.14 considering $H - U$ already.

7.6. Revisiting Lemma 5.20

In this section we want to revisit the proof of Lemma 5.20 and show how our algorithm determines that every cubic graph of path-width at most 4 has one of the five graphs in Figure 5.1 as a subgraph. To this end, we apply Algorithm 7.2 with the optimisations from Section 7.4.

Proof (of Lemma 5.20 using Algorithm 7.2). As announced, we execute Algorithm 7.2 with $\mathcal{G}$ being the class of all cubic graphs, $\mathcal{U}$ containing the triangle, $K_{2,3}$, domino, twin-house, and claw-square, and $k = 4$. We may also assume that the graphs in $\mathcal{G}$ have at least five vertices as the only cubic graph with less is the K_4, which contains a triangle. The progress of the algorithm is visualised if Figure 7.3.

The algorithm begins with the pair $(V_1, (V_1, \varnothing))$ where $V_1 = \{u_1, \dots, u_4, v_1\}$. This does not contain any of the subgraphs specified above, so it is removed in the first iteration of the while loop (and not before). In this iteration, we can find no counterexample as no cubic graph has five vertices. Also, any bijection on V_1 can be extended to a (U, H)-map, so all vertices are in the same orbit. Thus we only need to consider the case that v_1 leaves and $U' = U \setminus \{v_1\} \cup \{v_2\}$. As v_1 has degree 0 in H, we consider sets $Y \subseteq U \setminus \{v_1\}$ that have cardinality 3. We remove the set $\{u_1, u_2, u_3\}$ and add the pair (U', H') it creates to the queue. The update of Y removes all other sets.

Again the queue only consists of a single element, which we remove in the next iteration. For sake of simplicity, we call this pair (U, H) now, to coincide with the names used in the algorithm. For this pair, the maps in $\mathrm{Aut}(U, H)$ are those maps that fix v_1 and map the set $\{v_2, u_4\}$ to itself. We thus set $\overline{U}$ to $\{u_1, v_2\}$. We again find no counterexamples as the only two cubic graphs on six vertices are the $K_{3,3}$, which contains a $K_{2,3}$, and the prism on six vertices, which has a triangle.

For the set $U' = U \setminus \{v_2\} \cup \{v_3\}$ we again only consider sets $Y \subset U \setminus \{v_2\}$ of cardinality 3. If u_4 is not in Y, then the graph H' in the algorithm contains a $K_{2,3}$. Thus, Y consists of two elements of the set $\{u_1, u_2, u_3\}$ and u_4. But as we have elements in $\mathrm{Aut}(U, H)$ that arbitrarily permute first three vertices while fixing v_2, it suffices to consider the

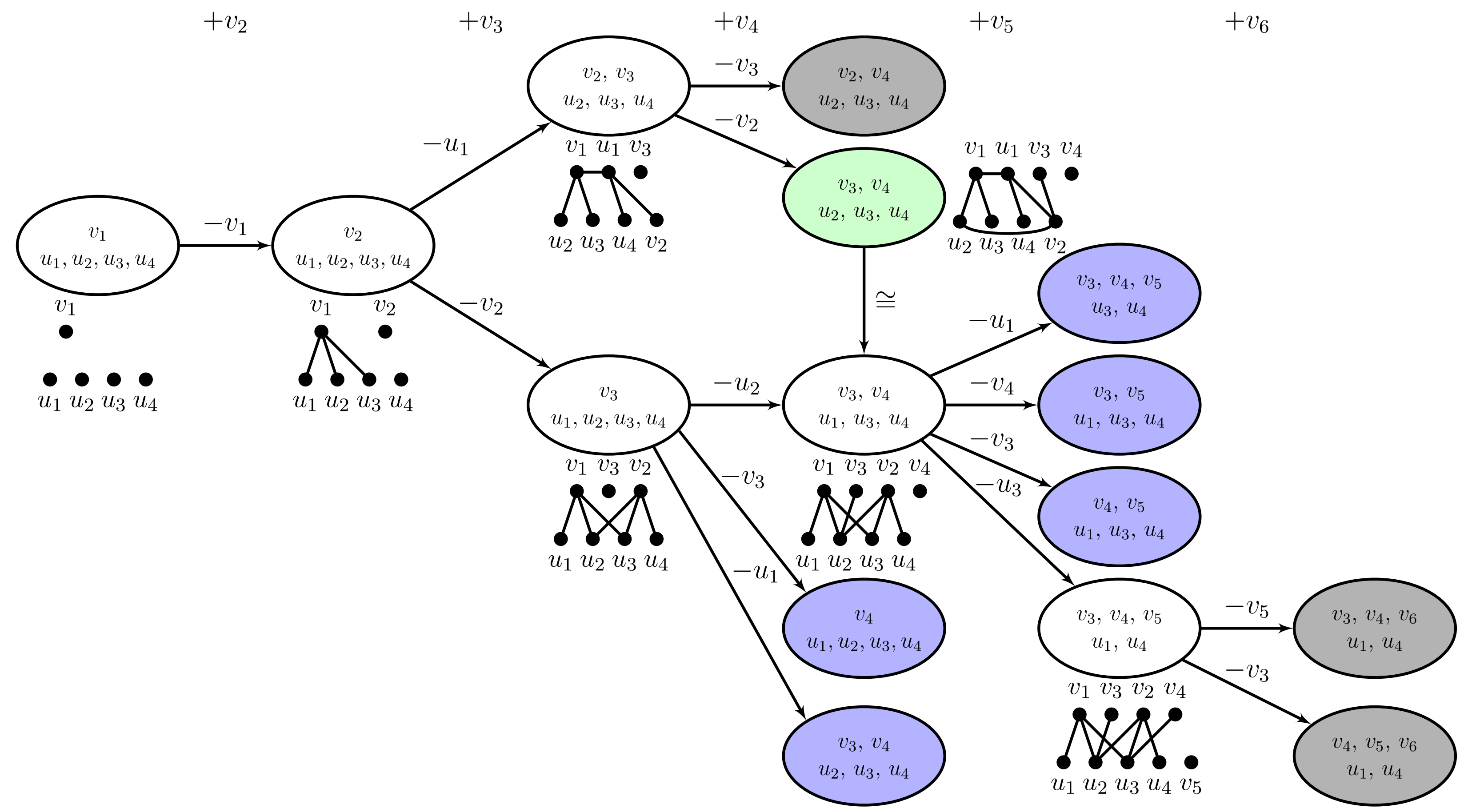


Figure 7.3.: A roadmap of the case distinctions made in the proof of Lemma 5.20. The nodes in this figure represent bags and their associated graphs are drawn close to them. A bag is white if it causes additional pairs to be added to the queue, grey if it does not, light green if it is not added because of the last symmetry (in Line 24), and light blue if it would be optimised away by Theorem 7.25.

set $Y = \{u_2, u_3, u_4\}$. We now obtain multiple pairs, so we call this one (U_1, H_1) (and recommend using Figure 7.3 to keep track of the algorithm's progress).

The final set we consider here is $U' = U \setminus \{u_1\} \cup \{v_3\}$. This time we only need to check sets Y of cardinality 2 as u_1 has degree 1. If u_2 or u_3 are in Y, G contains a triangle and otherwise $Y = \{u_4, v_2\}$. Consequently, only the pair (U_2, H_2) is added, where $U_2 = U'$ and $H_2 = H + v_3 + \{u_1u_4, u_1v_2\}$.

We continue and remove the pair (U_1, H_1). We cannot extend H_1 to a cubic graph as it has odd order, so we can proceed to choosing $\overline{U} = \{u_1, u_2, v_3\}$. Note that $\mathrm{Aut}(U_1, H_1)$ is generated by the permutation that exchanges u_2 and u_3 and the one that exchanges u_1, v_1 with u_4, v_2. Here, Theorem 7.25 would let us use $\overline{U} = \{u_2\}$ and we could skip the next two paragraphs.

We begin with $U' = U_1 \setminus \{v_3\} \cup \{v_4\}$. In this case, the set Y has cardinality 3, so v_3 has exactly one non-neighbour in $\alpha \in \{u_1, u_2, u_3, u_4\}$. If $\alpha \in \{u_1, u_4\}$, then G contains a $K_{2,3}$ and otherwise it contains a domino, so nothing new needs to be added here.

Next up is $U' = U_1 \setminus \{u_1\} \cup \{v_4\}$. Candidates for the set Y must have two elements, giving u_1 a neighbour in $\{u_2, u_3, u_4\}$. If u_1 is adjacent to u_2 or u_3, then G contains a triangle and otherwise G contains a twin-house.

Finally, let $U' = U_1 \setminus \{u_2\} \cup \{v_4\}$ and consider the one-element sets Y. If it is not the set $\{v_3\}$, then G contains a triangle and otherwise we add the corresponding pair to the queue and call it (U_3, H_3).

This leaves us with two pairs again, of which (U_2, H_2) is removed next. By the same cardinality argument, we can skip looking for a counterexample and, instead, directly look at the set $\mathrm{Aut}(U_2, H_2)$. Its elements are generated by the three maps exchanging u_2 and u_3, u_4 and v_2, and v_1, u_2, u_3 and u_1, u_4, v_2. As a result, the orbits are $\{v_3\}$ and $U_2 \setminus \{v_3\}$ and we can choose $\overline{U} = \{v_2, v_3\}$.

We begin with the option that $U' = U_2 \setminus \{v_3\} \cup \{v_4\}$. Again, we consider sets Y with three vertices, starting with $Y = \{u_3, u_4, v_2\}$. As this can be mapped to any other three element set with a map in $\mathrm{Aut}(U_2, H_2)$, it also suffices to look at this Y. But here the graph $(H_2 - u_2) + E\, v_3$ is isomorphic to twin-house and nothing new is added here.

This leaves $U' = U_2 \setminus \{v_2\} \cup \{v_4\}$ in which case we need sets Y with two elements. If u_4 is in Y, then we obtain a triangle and if u_2 and u_3 are in Y, then we have found a $K_{2,3}$. Nothing is added in either case and we are left with the two sets Y containing v_3 and either u_2 or u_3. We only need to consider $Y = \{v_3, u_2\}$ as the other is removed after this is processed. The resulting pair (U', H') need not be added to the queue as $(\varphi\, U', \varphi\, H') = (U_3, H_3)$ where φ is the following permutation of VG: it maps

$$v_3 \mapsto u_4,\ u_1 \mapsto u_2,\ u_2 \mapsto u_3,\ u_3 \mapsto u_1,\ u_4 \mapsto v_3, \text{ and identifies all other vertices.}$$

We now remove the last element (U_3, H_3) from our queue. This graph has only one non-trivial element in $\mathrm{Aut}(U_3, H_3)$, which exchanges v_1, u_1 and v_2, u_4. Also we find no counterexample as every cubic graph $G' = H_3 + E'$ contains an edge incident to u_1 with other end in $\{v_3, u_3, u_4\}$. But the presence of this edge gives us a domino, a triangle, or a twin-house, in that order. Proceeding to the set $\overline{U}$, we choose $\{v_4, v_3, u_1, u_3\}$. Once more, Theorem 7.25 yields a reduction of the set $\overline{U}$ to just the single element set $\{u_3\}$.

Start with $U' = U_2 \setminus \{u_1\} \cup \{v_5\}$, which we have already handled. In it, we need sets Y with two elements, putting at least one of then in the set $\{v_3, u_3, u_4\}$ and giving us a graph that contains an element of $\mathcal{U}$.

The case $U' = U_2 \setminus \{v_3\} \cup \{v_5\}$ is analogous, here v_3 has a neighbour in $\{u_1, u_4, u_3\}$, resulting in two dominoes and a $K_{2,3}$.

For $U' = U_2 \setminus \{v_4\} \cup \{v_5\}$, we need three vertices in Y. If Y contains u_3, then it also contains either u_1 or u_4, giving us a domino in both cases. Otherwise $Y = \{u_1, u_4, v_3\}$ and the resulting graph is claw-square.

This just leaves $U' = U_2 \setminus \{u_3\} \cup \{v_5\}$. The sets Y that are of interest are those with just a single element and of those, the ones containing just u_1, u_4, v_3 lead to two triangles and a $K_{2,3}$. So we consider the missing set $Y = \{v_4\}$, which yields a new pair that is added to the queue.

As this is the final iteration, we go back to calling this pair (U, H). As H has an odd number of vertices we need not look for a counterexample and there are two orbits in H, namely $\{v_5\}$ and $U \setminus \{v_5\}$. We choose $\overline{U} = \{v_3, v_5\}$ and start with $U' = U \setminus \{v_3\} \cup \{v_6\}$. For a set Y of cardinality 2, we get that Y contains a vertex $\alpha \in \{u_1, u_4, v_4\}$. The resulting edge $v_3\alpha$ either forms a domino, in the first two cases, or gives us a twin-house. Finally, let $U' = U \setminus \{v_5\} \cup \{v_6\}$, and Y be a set with three vertices. The graph H' obtained this way contains a claw-square. Consequently, nothing new is added and the algorithm terminates. $\square$

Chapter 8.

Relating the Girth and Path-Width of Cubic Graphs

We exploit our framework from the previous chapter to prove new lower bounds on the path-width of cubic graphs. Moreover, we determine the extremal girth values of cubic graphs of path-width k for all $k \in \{3, \dots, 10\}$ and all smallest graphs which take on these extremal girth values. Further, we present a new constructive characterisation of the extremal cubic graphs of path-width 3 and girth 4.

Inductive proofs in structural graph theory are our main motivation for the research on unavoidable subsets and, indeed, our new framework from Chapter 7 already provided us with (an alternative proof for) one structural result, to which we add in this chapter.

Regard (or recall) the following example: let $\mathcal{G}$ be the class of cubic graphs and write $\mathcal{U}_i$ for the set $\{C_3, \dots, C_i\}$. We saw in Section 7.1 that $\mathcal{U}_4$ is unavoidable if we restrict the path-width to at most 3. To investigate the maximal girth values of cubic graphs in relation to their path-width, we define $\xi\, k$ as the maximal girth of a cubic graph of path-width k, that is,

$$\begin{aligned}\xi \colon \mathbb{N}_{\geq 3} &\to \mathbb{N},\\ k &\mapsto \max\{g \colon \text{ there is a cubic graph of girth } g \text{ and path-width } k\}\,.\end{aligned}$$

For the case $k = 3$, we then know that $\xi\, 3 \leq 4$ by the example above. By running the algorithm on $\mathcal{U}_3$ with $k = 3$ we obtain the $K_{3,3}$ as a cubic graph of girth 4 and path-width at most 3. We can apply the algorithm in this fashion for larger values of k and, with this method, we find the value of $\xi\, k$ for $k = 3, \dots, 7$, as shown in Table 8.1.

Thus we have obtained a lower bound on the path-width based on the girth of the graph. Bodlaender and Koster [BK11] survey lower bounds for tree-width, amongst them is the following result of Chandran and Subramanian [CS05]. If G is a cubic graph of girth at least g, path-width k, and tree-width t, then $k \geq t \geq (e(g+1))^{-1} 2^{\lfloor (g-1)/2 \rfloor - 2} - 2$, where e denotes Euler's number. We note that the bound in [CS05] is more general since graphs

k	3	3	4	5	5	6	6	7
$\mathcal{U}$	$\mathcal{U}_3$	$\mathcal{U}_4$	$\mathcal{U}_4$	$\mathcal{U}_4$	$\mathcal{U}_5$	$\mathcal{U}_5$	$\mathcal{U}_6$	$\mathcal{U}_6$
Result	$K_{3,3}$	`None`	`None`	Petersen	`None`	Heawood	`None`	`None`
Pairs base	6	5	81	12484	3841	–	–	–
Pairs cubic	3	2	3	7	5	15	9	19

Table 8.1.: Computational results for cubic graphs of path-width k and unavoidable structures $\mathcal{U}$. The row 'Pairs base' contains the number of pairs considered by the base version of the algorithm while the row 'Pairs cubic' shows how many are left after the use of isomorphism rejection and tailoring the algorithm to cubic graphs. The three dashed cells contain no values as the algorithm did not terminate in a reasonable time on our machines.

3	4	5	6	7	8	9	10
$K_{3,3}$	cube twisted cube	Petersen	Heawood	Pappus	McGee	Tutte-Coxeter	G_{10}

Table 8.2.: All smallest graphs of path-width k and girth $\xi\, k$. The top row specifies the value of k while the bottom one contains the graphs. Here G_{10} denotes the unique cubic graph of path-width 10 and girth 8. We refer to Figure 8.1 for drawings of these graphs.

of average degree d are considered. We inserted $d = 3$ and added the inequality $k \geq t$ to arrive at the situation we consider. Lower bounds of the path-width are useful, for example when trying to compute the path-width of a graph.

Combining our algorithmic formalisms with combinatorial techniques, we prove two new (linear) bounds on ξ in Sections 8.1 and 8.2 (see Theorems 8.6 and 8.11). These improve the bound in [CS05] whenever the path-width is less than 26, that is, in all practically relevant cases when it comes to computing the path-width.

We also determine the precise values of $\xi\, k$ for $k \leq 10$: these are obtained by noting that the bounds from Theorem 8.11 are tight in these cases, which we show by exhibiting graphs that have the required girth. In fact, we can give a complete list of the minimal graphs of path-width k and girth $\xi\, k$ for all $k \in \{3, \dots, 10\}$. To do so, we used the complete list of small cubic graphs [Bri+13] and checked their path-width using SageMath [Sage20].

[Bri+13] [Sage20] [8.11] **8.1 Theorem.** For $k \in \{3, \dots, 10\} \setminus \{4\}$ there is a unique smallest cubic graph of path-width k and girth $\xi\, k$. There are two smallest graphs of path-width 4 and girth 4. See Table 8.2. ◁

A (d, g)-*cage* is a minimal d-regular graph of girth g. The study of cages dates back to [Tut47]. A recent survey on this topic is [EJ12]. If $d = 3$, then there is a unique $(3, g)$-cage for all $g \in \{3, \dots, 8\}$. Clearly, if a cubic cage of girth $\xi\, k$ has path-width k, then it appears in the table. Thus, it is not surprising that several of the graphs above

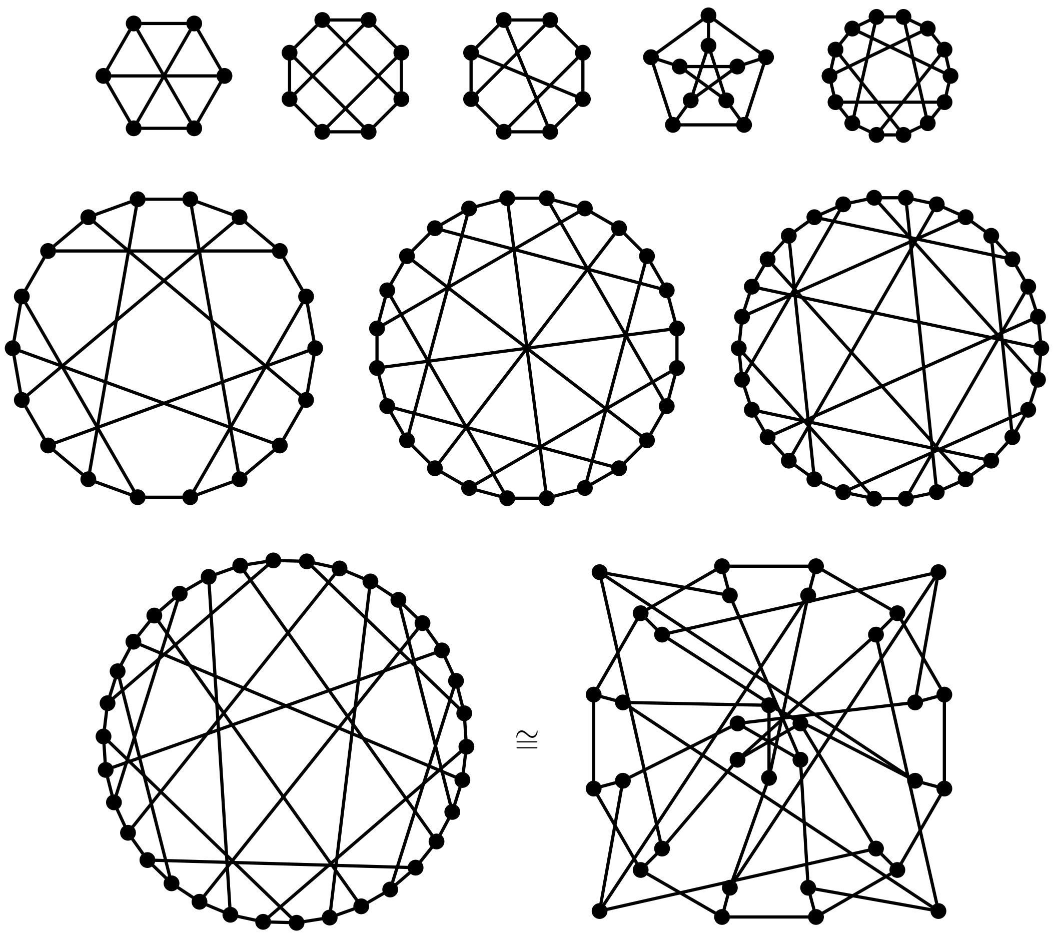


Figure 8.1.: All graphs of path-width k and girth $\xi\, k$ for $k \in \{3, \ldots, 10\}$. First row: the $K_{3,3}$, the cube, the twisted cube, and the Petersen and Heawood graph. Middle row: the Pappus, McGee, and Tutte-Coxeter graph. Last row: two drawings of the unique graph of girth 8 and path-width 10, which we denote by G_{10}.

are cages: the $K_{3,3}$, the Petersen graph, the Heawood graph, and the Tutte-Coxeter graph.

As a final application of our algorithm we obtain a constructive classification for the cubic graphs of path-width 3 and girth 4 in Section 8.3.

The content of this chapter is joint work with Irene Heinrich and makes up the second part of [BH20].

8.1. A first bound on the girth

Let G be a cubic graph of path-width k and girth g. We show that large girth necessitates large path-width. Before we start formally proving Theorem 8.6 (and Theorem 8.11), let us briefly sketch how these proofs work. First, we gather information about the associated graphs G_i for the initial bags, by counting both the connected components of these graphs as well as their number of vertices of degree 0 to 3. To limit the possible cases and to obtain particularly simple ones with few degree 2 and 3 vertices, we use hdf path-decompositions. From this we can deduce that a cycle is present by the kth bag and at least two exist by bag $k+2$. Theorem 8.6 is obtained by considering the last bag l for which G_l contains at most one cycle and use the small number of vertices that can lie on cycles at all to deduce that G_{l+1} has a short cycle. For Theorem 8.11 we show that G_i is a forest for all $i \leq g-2$ and then combine this knowledge with our degree and component counts to go through the graphs G_{g-1} to G_{g+3} to find a short cycle.

We now begin with the formal proof. Recall that hdf-decompositions exist and allow us to assume that degree 2 or 3 vertices, if present, leave a bag, see Definition 7.23 and Theorem 7.24. These are useful when determining the structure of the graphs G_i for the initial bags of the decomposition. For these proofs, we also need the *extended graph* of G, which is defined as

$$G^{\text{ext}} := G + E_{k+1}.$$

This lets us extend the path-decomposition of G by letting the vertices in the last bag leave while adding the new degree 0 ones. We call such a path-decomposition an *extended decomposition of G*. By starting with an hdf-decomposition of G, this can easily be made hdf as well. The sole purpose of this extension is to ensure that the decomposition has enough bags so that sufficiently many associated graphs exist.

For the remainder of this chapter, we adhere to the following conventions.

8.2 Convention. The graph G is cubic and has girth g and path-width k. We denote the extended graph G^{ext} by H and H comes with a path-decomposition $(P, \mathcal{V})$ that is an extended decomposition of G and also hdf. Moreover, in terms of notation concerning the path-decomposition, we refer to Convention 7.1. ◁

Recall the definition of new neighbours of a vertex v, Definition 7.3, which are exactly those $3 - d_{H_i} v$ many vertices that are connected to v by an edge in the transition to H_{i+1}.

We begin by describing the degree distribution in the associated graphs. To this end, we set

$$d_j^i := |\{v \in H_i \colon \; d_{H_i} v = j\}| \text{ for } j \in \{0,1,2,3\} \text{ and } i \in \{1,\dots,n'\}\,.$$

We write t_i for the number of non-trivial components of H_i, where non-trivial means not of order 1. To determine the degrees, we proceed by induction on i. We consider H_{i+1}, assuming that the claim holds for H_i and H_{i+1} is of the form $H_i + E\,v + v_{i+1}$.

8.3 Lemma. Let $i \in \{1,\dots,n'\}$. If H_i is a forest, then: [2.2] (8.4) (8.5) (8.6) (8.8) (8.10) (8.11)

(a) $d_0^i \geq 1$.
(b) $d_1^i = i - 1 + 2t_i$.
(c) $d_2^i \leq 3$, at most one component of H_i contains vertices of degree 2, and no subpath of H_i contains more than two degree 2 vertices.
(d) $d_3^i = i - 1$. ◁

Proof. To show the claim, we replace the subpath part of Property (c) by the following stronger property: there exists a vertex z whose removal separates all degree 2 vertices of H_i.

For $i = 1$ the claim holds as $H_1 = G_1 \cong E_{k+1}$. Assume that the claim holds up to some index $i \geq 1$ and let $H_{i+1} := H_i + E\,v + v_{i+1}$ be acyclic. We note that $d_{H_{i+1}}(v_{i+1}) = 0$ and Property (a) follows. Furthermore, any vertex that has left a prior bag has degree 3 in H_i and by Property (d) there are no others. Hence, $d_{H_i} v < 3$. Since H_{i+1} is acyclic, v and all new neighbours of v are in different components of H_i. Thus, no new neighbour of v has degree 2 in H_i: if this were the case, then by the hdf property $d_{H_i} v = 2$ as well and both these vertices lie in the same component of H_i by Property (c), creating a cycle. So $d_3^{i+1} = d_3^i + 1 = i$ and Property (d) holds.

Next, we remark that the handshaking lemma yields the following equation, where $\kappa(H_{i+1})$ denotes the number of components of H_{i+1}:

$$d_1^{i+1} + 2d_2^{i+1} + 3d_3^{i+1} = 2\|H_{i+1}\| = 2\left(|H_{i+1}| - \kappa(H_{i+1})\right) = 2d_1^{i+1} + 2d_2^{i+1} + 2d_3^{i+1} - 2t_i.$$

From this we deduce that $d_1^{i+1} = d_3^{i+1} + 2t_i = i + 2t_{i+1}$ and Property (b) is satisfied.

This only leaves Property (c). First assume $d_2^i > 0$, in which case $d_{H_i} v = 2$ by hdf and we get a degree 2 vertex z that separates all degree 2 vertices in H_i. Let u be the unique new neighbour of v. If $d_{H_i} u = 1$, then $d_{H_{i+1}} u = 2$ but u is part of the same component as the remaining degree 2 vertices of H_i. Moreover z still separates the vertices of degree 2 in H_{i+1} as u is a neighbour of the prior degree 2 vertex v. If u does not have degree 1, the claim follows directly.

If $d_2^i = 0$, then all degree 2 vertices of H_{i+1} are new neighbours of v that had degree 1 in H_i. Consequently, there are at most three such vertices, they all lie in the same component of H_{i+1}, and they are separated by v. □

[8.3] (8.5) (8.6) **8.4 Lemma.** Let $i \in \{1, \ldots, n'\}$. If H_i contains a unique cycle, then there exists an index $j \in \{i, i+1\}$ such that H_j contains a unique cycle and satisfies the following properties:

(a) $d_0^j \geq 1$.
(b) $d_1^j = j - 3 + 2t_j$.
(c) $d_2^j \leq 3$ and at most one component of H_j contains vertices of degree 2.
(d) $d_3^j = j - 1$. ◁

Proof. Property (a) is satisfied because v_j has degree 0 in H_j, for all j. The claim holds for $i = 1$ since H_1 is always acyclic. Assume that it holds up to some $i \geq 1$ and let $H_{i+1} \coloneqq H_i + E\,v + v_{i+1}$ contain a unique cycle. The graph H_i thus contains at most one cycle and we first assume it is acyclic, letting us apply Lemma 8.3.

If $d_{H_i}\, v = 2$, then the new neighbour u of v is in the same component as v in H_i and $t_{i+1} = t_i$. The claim holds for $j = i + 1$ when $d_{H_i}\, u = 1$: here

$$d_3^{i+1} = d_3^i + 1 = i,\ d_1^{i+1} = d_1^i - 1 = i - 2 + 2t_i,\ d_2^{i+1} = d_2^i \leq 3, \text{ and}$$

all degree 2 vertices are in the same component. This leaves the case that $d_{H_i}\, u = 2$, where H_{i+1} does not satisfy the properties as it has $i + 1$ vertices of degree 3. However, by hdf, u is the next vertex to leave and $H_{i+2} \coloneqq H_{i+1} + v_{i+2}$. Here, H_{i+2} satisfies

$$d_3^{i+2} = d_3^{i+1} = d_3^i + 2 = i + 1,\ d_1^{i+2} = d_1^{i+1} = d_i^i = i - 1 + 2t_i, \text{ and } d_2^{i+2} = d_2^{i+1} = d_2^i - 2 \leq 1,$$

completing this case.

We may now assume that $d_{H_i}\, v \leq 1$ and $d_2^i = 0$. As H_{i+1} has a unique cycle, either one new neighbour is in the same component as v in H_i and the remaining ones are in different components or no new neighbour is in the same component as v and exactly two of them share a component. In either case, $d_3^{i+1} = d_3^i + 1 = i$ and at most three degree 2 vertices are present in H_{i+1}, which share a component. For the degree 1 vertices, note that v has one new neighbour of degree 1 that creates the cycle and the remaining new neighbours are either of degree 0 or 1. In the first case, the number of degree 1 vertices increases by one and in the second it decreases by one, but so does the number of non-trivial trees. This shows Property (b).

We are left with the case that H_i is not acyclic, which means it contains a unique cycle and the induction hypothesis applies. This gives us that either H_{i+1} satisfies the degree properties, and we are done, or H_i does. We may assume the latter. The only new degree 3 vertex is v since connecting two vertices of degree 2 would result in a new cycle. If $d_{H_i}\, v = 2$, its neighbour has degree 0 or it has degree 1 and is in a different component.

This increases the number of degree 1 vertices by one or decreases their number and the number of non-trivial trees by one each. Otherwise, $d_{H_i} v \leq 1$ and H_i has no degree 2 vertices. The new neighbours now have degree 1 and are in different components of H_i or degree 0 and the properties hold once more. □

8.5 Observation. The graph H_k is not a forest and H_{k+2} has more than one cycle. ◁ [8.3] [8.4] (8.6)

Proof. Suppose H_k is acyclic. Lemma 8.3 states that

$$2k = |H_k| \geq d_3^k + d_1^k + d_0^k \geq 2k - 1 + 2t_k,$$

which is a contradiction since $k \geq 3$ and, hence, $t_k > 0$. Similarly, if H_{k+2} has at most one cycle, then it has exactly one and either H_{k+2} or H_{k+3} satisfy the degree properties of Lemma 8.4. Using these we obtain

$$2k + 2 = |H_{k+2}| \geq d_3^{k+2} + d_1^{k+2} + d_0^{k+2} \geq 2k + 2t_{k+2} + 1$$

or

$$2k + 3 = |H_{k+3}| \geq 2k + 2t_{k+3} + 3,$$

which is a contradiction in both cases. □

We are now ready to prove the following bound.

8.6 Theorem. For all $k \in \mathbb{N}_{\geq 3}$ the following inequality is satisfied: [8.3] [8.4] [8.5]

$$\xi\, k \leq \tfrac{2}{3}k + \tfrac{10}{3}. \quad ◁$$

Proof. To prove this result, we need to show that $g \leq \frac{2}{3}k + \frac{10}{3}$ for any graph G of path-width k and girth g. Regard the extended graph H of G. Let l be the maximal index such that H_l contains at most one cycle. By Observation 8.5 we have that $l \leq k + 1$. Since H_l contains either no cycle, and Lemma 8.3 applies, or it contains exactly one, and Lemma 8.4 can be used, we get that $d_3^l = l - 1$ and $d_2^l \leq 3$. Note that H_{l+1} contains multiple cycles, so Lemma 8.4 actually applies to H_l. This lets us estimate the number of vertices of degree at least 2 in H_{l+1}: if $d_2^l > 0$, then at most one edge is added in the transition to H_{l+1} by hdf. Since at least one end of such an edge has degree 2, we obtain $d_3^{l+1} + d_2^{l+1} \leq d_3^l + d_2^l + 1$. On the other hand, if $d_2^l = 0$, then at most one new degree 3 and three new degree 2 vertices are created. This yields $d_3^{l+1} + d_2^{l+1} \leq d_3^l + 4$. Combined these give us that

$$d_3^{l+1} + d_2^{l+1} \leq d_3^l + 4 = l + 3.$$

Let C and C' be two distinct cycles in H_{l+1}. If they are disjoint, then $|C| + |C'| \leq l + 3$, yielding

$$g \leq \frac{l+3}{2} \leq \frac{k}{2} + 2.$$

Otherwise, C' contains a path Q between two non-adjacent vertices of C. Using this path and C, we obtain two cycles of which one has length at most $\frac{|C|}{2} + \|Q\|$. We estimate

$$|Q| \leq d_3^{l+1} + d_2^{l+1} - |C| + 2 \leq l + 5 - |C|$$

to get

$$g \leq |C| \text{ and } g \leq \tfrac{|C|}{2} + l + 4 - |C| = l + 4 - \tfrac{|C|}{2}.$$

Consequently, since $|C| = l + 4 - \frac{|C|}{2}$ holds for $|C| = \frac{2}{3}l + \frac{8}{3}$ and $l \leq k + 1$

$$g \leq \frac{2}{3}k + \frac{10}{3}.$$

The bound from the disjoint cycle case is strictly better than this one and, hence, the result follows. □

8.2. A second bound on the girth

In this section, we provide the (tight) upper bounds on $\xi\, k$ for small values of k, which we need for the proof of Theorem 8.1. In preparation, we make the following observations.

(8.8)
(8.11) **8.7 Observation.** The neighbours of a vertex v of degree at most 2 in H_i are of degree 3. ◁

Proof. Note that H_i only contains edges incident to vertices that have left one of the first $i-1$ bags. Therefore, at least one end of any edge is of degree 3. □

[8.3]
[8.7]
(8.9)
(8.10)
(8.11) **8.8 Observation.** If H_i is a forest and Q is a path in H_i, then

$$|Q| \leq i + 2 + \min\left\{d_2^i, 2\right\} - t_i.$$

If an end of Q has degree 2, then this bound improves by 2 to

$$|Q| \leq i + \min\left\{d_2^i, 2\right\} - t_i.$$ ◁

Proof. The path Q has at most two vertices of degree 1, the remaining ones are of degree 2 or 3. By Lemma 8.3, H_i has $d_3^i = i - 1$ and at most two of degree 2 lie on Q. Also, any non-trivial component contains at least one degree 3 vertex by Observation 8.7. This gives us that $|Q| \leq 2 + \min\{d_2^i, 2\} + [d_3^i - (t_i - 1)]$, which is the first inequality.

If one end, say v has degree 2, then at most one vertex of degree 1 is in Q. Additionally, both neighbours of v have degree 3 by Observation 8.7 and at most one of them is on Q. Consequently $|Q| \leq 1 + \min\{d_2^i, 2\} + [d_3^i - 1 - (t_i - 1)]$, completing the proof. □

[8.8]
(8.10)
(8.11) **8.9 Observation.** If $H_{i+1} = H_i + E\, v + v_{i+1}$, H_i is a forest, and $i \leq g - 2$, then no new neighbour of v is in the same component as v in H_i. ◁

Proof. Suppose that u is a new neighbour of v and in the same component of H_i as v. Let Q be the path joining v and u in H_i. If $d_2^i = 0$, then $|Q| \leq i + 2 - t_i \leq g - 1$ by Observation 8.8. If $d_2^i \geq 1$, then $d_{H_i} v = 2$ by hdf. Again Observation 8.8 yields $|Q| \leq i + \min\{d_2^i, 2\} - t_i \leq g - 1$. In both cases, the edge vu causes a cycle of length at most $g - 1 < g$, which is a contradiction. □

Next, we show that high girth necessitates many acyclic associated graphs.

8.10 Lemma. For $i \leq g - 2$, H_i is a forest. ◁ [8.3] [8.8] [8.9] (8.11)

Proof. For $i = 1$ the claim holds since $H_1 \cong E_{k+1}$. Assume the claim holds for some $i \leq g - 3$. If $i = 1$, then $|H| > |G| \geq k + 1 = |H_1|$ and, hence, H_2 exists. If $i \geq 2$, then $d_1^i \geq 1$ by Lemma 8.3 and since any vertex of H is isolated or of degree 3, the graph H_{i+1} exists. Let v be the vertex that leaves V_i. We have $H_{i+1} = H_i + E\,v + v_{i+1}$.

First note that, by Observation 8.9, no new neighbour of v is part of the same component as v in H_i. In particular, if $d_{H_i} v = 2$, then H_{i+1} is acyclic. This leaves the case that $d_{H_i} v < 2$. Since the path-decomposition is hdf, we know that $d_2^i = 0$ and there are multiple new neighbours of v. In case two of them are in the same component of H_i, we get a short cycle by noticing that the path between these neighbours has order at most $i + 2 - t_i \leq g - 2$ by Observation 8.8, which is a contradiction. Thus H_{i+1} is a forest.□

With these tools at hand, we can now prove the bound we need. We note that this is an improvement on the bound in Theorem 8.6 for $k \leq 13$.

8.11 Theorem. The values of ξ for small values of k are shown in the table below: [8.3] [8.7] [8.8] [8.9] [8.10] (8.1)

k	3	4	5	6	7	8	9	10
$\xi\,k$	4	4	5	6	6	7	8	8

Additionally, $\xi\,k \leq k - 2$ holds for all $k \geq 10$. ◁

Proof. The graphs found in Table 8.2 show that $\xi\,k$ takes at least the value specified above and it suffices to prove that it is not larger. To this end we show that

$$g \leq k + 1 \text{ in general, } g \leq k \text{ if } k \geq 4,\ g \leq k - 1 \text{ if } k \geq 7, \text{ and } g \leq k - 2 \text{ if } k \geq 10.$$

This proves the equalities for the values specified above and shows that $\xi\,k \leq k - 2$ for $k \geq 10$.

We know that $H_1, \ldots, H_{g-2}$ are acyclic by Lemma 8.10. In this proof, we look at the possibilities that arise for the subsequent associated graphs, starting with H_{g-1}. Since H_{g-2} is a forest, Lemma 8.3 implies $d_3^{g-2} = g - 3$, $d_1^{g-2} = g - 3 + 2t_{g-2}$, and $d_0^{g-2} \geq 1$. Therefore

$$1 \leq d_0^{g-2} + d_2^{g-2} = k + g - 2 - (g - 3) - (g - 3 + 2t_{g-2}) = k + 4 - g - 2t_{g-2}. \quad (8.1)$$

Rearranging yields $g \leq k + 3 - 2t_{g-2} \leq k + 1$ proving the first of the four inequalities.

For the remainder of this proof, we may assume

$$g \geq k - 1 \text{ and } k \geq 4,\ g \geq 5$$

since for smaller values the claimed inequalities are already true. Using the former and $g \leq k + 3 - 2t_{g-2}$, we get $t_{g-2} \leq 2$ and $t_{g-2} = 1$ if $g \geq k$. We now go through the graphs H_{g-1} to H_{g+3} and see what forms they can have. While doing so, we recommend using Figure 8.2 as a guide, which depicts the major steps.

Let $H_{g-1} := H_{g-2} + E\,v + v_{g-1}$. Since H_{g-2} has $g - 3$ vertices of degree 3, v is not one of them and it has new neighbours. By Observation 8.9, any new neighbour is in a component different from v. In particular, if $d_{H_{g-2}}\,v = 2$, then H_{g-1} is acyclic. Assume that v has degree at most 1 in H_{g-2}, which implies $d_2^{g-2} = 0$ by hdf. If all new neighbours of v are in different components of H_{g-2}, then H_{g-1} is acyclic and we obtain, by Lemma 8.3, that

$$\begin{aligned}&H_{g-1} \text{ is acyclic,}\\ &t_{g-1} = 1,\ d_3^{g-1} = g - 2,\ d_1^{g-1} = g, \text{ and } d_0^{g-1} + d_2^{g-1} = k + 1 - g.\end{aligned} \tag{8.2}$$

Note that we obtained $t_{g-1} = 1$ and $d_0^{g-1} + d_2^{g-1} = k + 1 - g$ by counting arguments: H_{g-1} has $k + g - 1 \leq 2g$ vertices in total and $d_0^{g-1} \geq 1$. The last inequality is true for all associated graphs, which is why we omit writing it each time. We refer to the left graph in the top row of Figure 8.2 for an illustration of this situation.

This just leaves the case that two neighbours of v share a component. In analogy to the proof of Lemma 8.10, the path between two such neighbours has order at most $g - t_{g-2}$ by Observation 8.8. This implies $t_{g-2} = 1$ and the path has length exactly $g - 1$ and contains all degree 3 vertices. We conclude that $d_{H_{g-2}}\,v = 0$ in this case and H_{g-1} must be acyclic if $d_{H_{g-2}}\,v = 1$.

Thus, if H_{g-1} is not acyclic, then v is isolated and there is either a unique cycle of length g in H_{g-1} or the third new neighbour of v is also in the same component of H_{g-2} as the other two. In the first case, H_{g-1} has one non-trivial component and satisfies $d_3^{g-1} = d_3^{g-2} + 1 = g - 2$, $d_2^{g-1} = 2$, and $d_1^{g-1} = d_1^{g-2} - 1 = g - 2$. By the same counting argument as before, we are in the following situation, which is illustrated on the top right of Figure 8.2,

$$\begin{aligned}&H_{g-1} \text{ has a unique cycle,}\\ &t_{g-1} = 1,\ d_3^{g-1} = d_1^{g-1} = g - 2,\ d_2^{g-1} = 2, \text{ and } d_0^{g-1} = k + 1 - g.\end{aligned} \tag{8.3}$$

Otherwise, if v has all three neighbours u_1, u_2, u_3 in the same component T, the paths u_iTu_j are of order at least $g - 1$ for $1 \leq i < j \leq 3$. Consequently, all three of them contain all $g - 3$ vertices of degree 3 in H_{g-2} and differ only in their ends. Hence, H_{g-1} contains a cycle of length at most 4 and $g \leq 4$, contradicting our assumption.

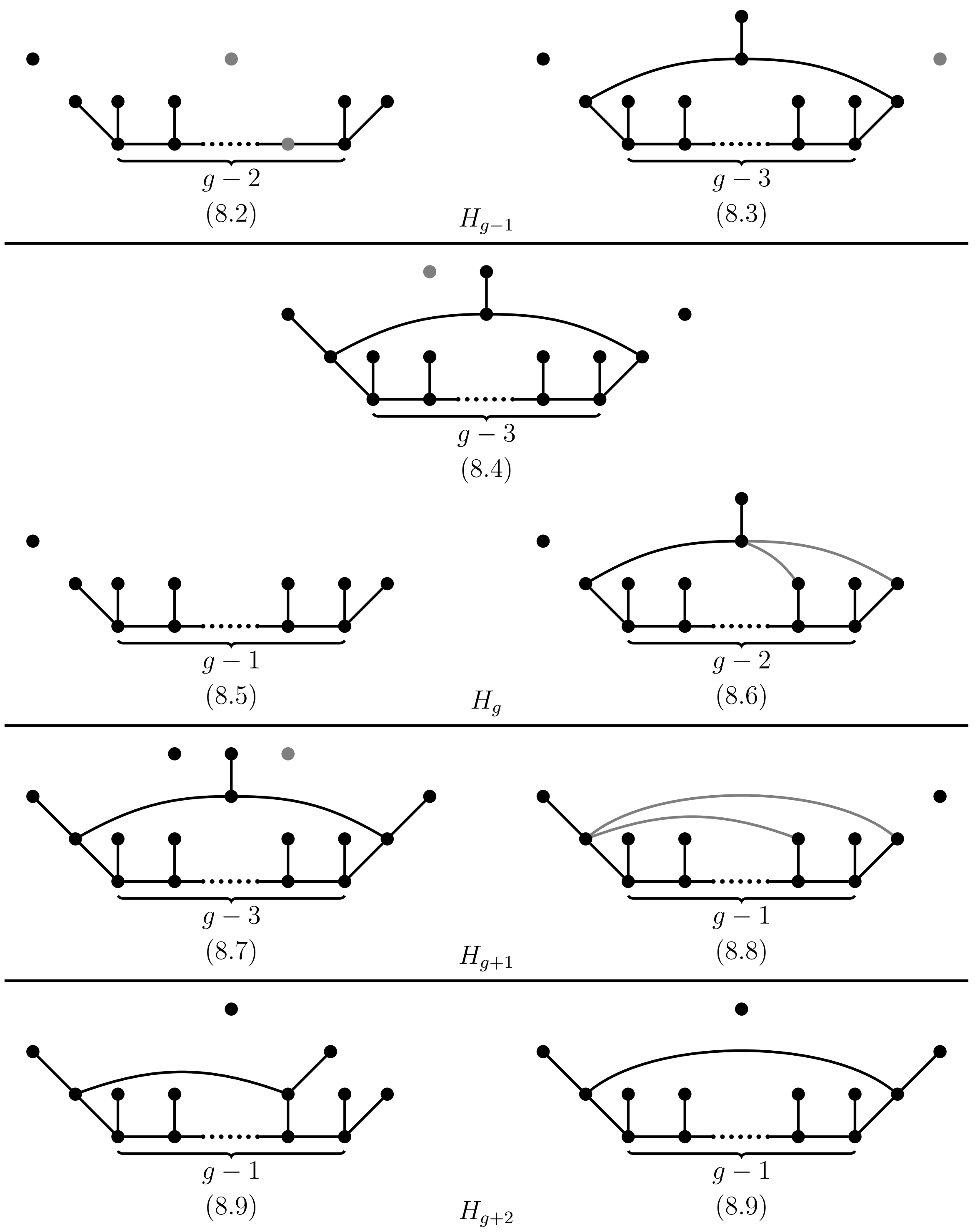


Figure 8.2.: The forms the graphs H_{g-1} to H_{g+2} can take in the proof on Theorem 8.11. Note that the grey degree 0 vertices may or may not be present in the graph. In the graph labelled (8.2), both grey vertices cannot occur simultaneously and exactly one of the grey edges in the graphs labelled (8.6) and (8.8) is part of the graph.

This completes the inequality $g \leq k$ for $k \geq 4$: if $g = k + 1$, then in both Option (8.2) and Option (8.3) we get that $d_0^{g-1} = 0$, contradicting the fact that this value is always positive.

From now on we may assume that

$$k, g \geq 7$$

and need to verify the last two inequalities. Our next goal is to describe the graph $H_g := H_{g-1} + E\,w + v_g$, where we had Options (8.2) and (8.3) for H_{g-1}. Note that, by $g \geq k - 1$, $d_0^{g-1} \leq 2$.

Let us start with the Option (8.3). We know here that $d_{H_{g-1}}\,w = 2$ and we denote its unique new neighbour by u. If u has degree 0, then

$$\begin{aligned}&H_g \text{ has a unique cycle,}\\ &t_g = 1,\ d_3^g = d_1^g = g - 1,\ d_2^g = 1, \text{ and } d_0^g = k - g + 1.\end{aligned} \tag{8.4}$$

Otherwise, u lies in the same component as w and we obtain a path of length at most 2 between two vertices of C since C contains all vertices of degree at least 2. This yields a cycle of length at most $\frac{|C|}{2} + 2$ and we get $g \leq \frac{g}{2} + 2$ or $g \leq 4$, a contradiction.

If H_{g-1} is acyclic, there are more options. For ease of notation, let T be the non-trivial component of H_{g-1}. First assume that $d_{H_{g-1}}\,w = 2$ and let u be its unique new neighbour. If u is in T, then $|wTu| \leq g - 1$ by Observation 8.8 since $d_2^{g-1} \leq 1$, a contradiction. Thus, u has degree 0 and

$$\begin{aligned}&H_g \text{ is acyclic,}\\ &t_g = 1,\ d_3^g = g - 1,\ d_2^g = 0,\ d_1^g = g + 1, \text{ and } d_0^g = 1\end{aligned} \tag{8.5}$$

by Lemma 8.3. Here we have used that $g \geq k - 1$ to obtain that $d_2^g = 0$, in particular, this case does not occur if $g = k$. Again, our illustration in Figure 8.2 puts the degree 3 vertices on a path despite this not being mandatory at this point.

This just leaves the case that $d_{H_{g-1}}\,w \leq 1$, giving us $d_2^{g-1} = 0$. Assume first that $d_{H_{g-1}}\,w = 1$. Should both new neighbours of w have degree 0, then H_g is acyclic, and we are in Option (8.5). Otherwise, let $u \in T$ be a new neighbour of w. Again we use Observation 8.8 to see that $|uTw| \leq g$ and obtain a cycle C of length exactly g in $H_{g-1} + uw$. Should the second new neighbour of w also be in T, we obtain a path of length at most 2 between two vertices of C, yielding girth at most 4, just as above. This does not occur, so H_g contains a unique cycle of length g and checking the values d_j^g shows that we are in Option (8.4).

We are left with the situation where $d_{H_{g-2}}\,w = 0$. Because $d_0^{g-1} \leq 2$ at least two of its new neighbours, say u_1, u_2, are in T. We first argue that the third neighbour, u_3, is not. To see this, note that $C := u_1Tu_2wu_1$ is a cycle in G of length g or $g + 1$, by the girth requirement and the number of vertices of degree at least 2. If $u_3 \in T$, then we obtain a path between two vertices of C. If $|C| = g + 1$, then C contains all vertices of degree 3 and 2, giving this path length at most 2. Otherwise, if $|C| = g$, this path potentially has length 3. Hence, we obtain a cycle of length at most $\frac{g}{2} + 3$ or $\frac{g+1}{2} + 2$. From this we can deduce that $g \leq \frac{g}{2} + 3$, yielding a contradiction to $g \geq 7$.

Consequently, u_3 has degree 0 and H_g has a unique cycle $u_1 T u_2 w u_1$ of length at most $g + 1$, giving us the final option in which

$$\begin{aligned}&H_g \text{ has a unique cycle,}\\ &t_g = 1,\ d_3^g = g - 1,\ d_2^g = 2,\ d_1^g = g - 1, \text{ and } d_0^g = 1.\end{aligned} \tag{8.6}$$

The last equality follows once more from the fact that H_g has $k+g$ vertices, so $g = k-1$ in this case. Here, Figure 8.2 is almost correct: either all degree 3 vertices lie on the path or we missed one. The only slight inaccuracy is that this potentially missed vertex need not be next to the one of degree 2, it could be anywhere.

We now have several possible options for H_g and want to obtain information about $H_{g+1} := H_g + E\,x + v_{g+1}$. We begin by taking a look at Option (8.4), which is the only one relevant for the case that $k = g$. Here, $d_{H_g}\,x = 2$ and the identical argumentation to before shows that the new neighbour of x has degree 0 and H_{g+1} satisfies

$$\begin{aligned}&H_{g+1} \text{ has a unique cycle,}\\ &t_{g+1} = 1,\ d_3^{g+1} = d_1^{g+1} = g,\ d_2^{g+1} = 0, \text{ and } d_0^{g+1} = k - g + 1.\end{aligned} \tag{8.7}$$

We claim that this suffices to prove the third inequality. Assume $g = k$, then we already know we are in Option (8.4) as Options (8.5) and (8.6) only occur for $g = k-1$. Hence, H_{g+1} is of the form described above and $d_0^{g+1} = 1$. We consider $H_{g+2} := H_{g+1} + E\,y + v_{g+2}$. If the degree 0 vertex leaves, all its neighbours, u_1, u_2, u_3 say, are in the same component as the cycle C. Suppose u_1 and u_2 have minimal distance in C, then there exists a path of length at most $\frac{g}{3}$ between them and we obtain a cycle of length $\frac{g}{3} + 4$ by extending it to use $u_1 y u_2$. Note that all degree 1 vertices of H_{g+1} are adjacent to a degree 3 one by Observation 8.7 and all degree 3 vertices are on C. This yields $g \leq 6$, a contradiction. Hence, $d_{H_{g+1}}\,y = 1$ and it has at least one neighbour u in the same component as the cycle. But now we get a cycle of length at most $\frac{g}{2} + 3$ which again yields $g \leq 6$, proving the $g \leq k - 1$ for $k \geq 7$.

For the final inequality ($g \leq k - 2$ if $k \geq 10$), we can update our assumptions to

$$g = k - 1, \text{ and } k \geq 10, \text{ so } g \geq 9.$$

With this we finish the description of the options for H_{g+1} and note that, in Option (8.7), we can now deduce that $d_0^{g+1} = 2$.

In the case that Option (8.5) applies to H_g, we denote the non-trivial tree of H_g by T. If the vertex of degree 0 leaves, then it has all three neighbours, u_1, u_2, u_3 say, in T. As before, we obtain a cycle $C = u_1 T u_2 x u_1$, this time of length g, $g + 1$, or $g + 2$, and a path of length at most 4, 3, or 2 between two vertices of C. Consequently, there is a cycle of length at most $\frac{g}{2}+4$, $\frac{g+1}{2}+3$, or $\frac{g+2}{2}+2$. Again, we arrive at a contradiction to $g \geq 9$ since we have obtained that $g \leq \frac{g}{2}+4$.

As a result, $d_{H_g} x = 1$. If x has two neighbours in T, then we repeat the previous argument for the degree 0 vertex to get a short cycle, contradicting our girth assumption. So x has one new neighbour of degree 0 and the other one in T. Let u be this neighbour, then $|uTx| \leq g+1$ by Observation 8.8 and we obtain

$$\begin{aligned}&H_{g+1} \text{ has a unique cycle containing the vertex of degree 2,}\\ &t_{g+1}=1,\ d_3^{g+1}=d_1^{g+1}=g, \text{ and } d_2^{g+1}=d_0^{g+1}=1.\end{aligned} \tag{8.8}$$

Finally, we consider the Option (8.6), which again contains a unique cycle C. Here, $d_{H_g} x = 2$ and x has a unique new neighbour u. If $u \notin T$, then this coincides with Option (8.8). Otherwise, if $u \in T$, we obtain a path of length at most 3 or 2 between two vertices of the cycle C of length g or $g+1$. In either case, we obtain a cycle of length at most $\frac{g+1}{2}+2 \leq \frac{g}{2}+3$ and this yields $g \leq 8$, a contradiction.

The next graph is $H_{g+2} \coloneqq H_{g+1} + E\,y + v_{g+2}$. For H_{g+1} only the two Options (8.7) and (8.8) remain, and we start with the latter. Here, we have $d_{H_{g+1}} y = 2$ by hdf and y is on the unique cycle C. Again, if the new neighbour of y is not a vertex of degree 0, the same argument as above yields a contradiction. Thus, we may assume that a unique cycle remains and we get

$$\begin{aligned}&H_{g+2} \text{ has a unique cycle,}\\ &t_{g+2}=1,\ d_3^{g+2}=d_1^{g+2}=g+1,\ d_2^{g+2}=0, \text{ and } d_0^{g+2}=1.\end{aligned} \tag{8.9}$$

If H_{g+1} is as specified in (8.7), then $d_{H_{g+1}} y = 0$ results in a path of length 4 between two vertices of C, which gives us a cycle of length at most $\frac{g}{2}+4$. This implies that $g \leq 8$, a contradiction. So y has degree 1 and its new neighbours have degree 0 as otherwise we obtain a path of length 3 between two vertices of C. This leaves the unique cycle of length g intact and results in $g+1$, 0, $g+1$, 1 vertices of degree 3, 2, 1, 0, letting us include it in Option (8.9).

Now we can wrap up the proof by considering H_{g+3}. Since only Option (8.9) remains for H_{g+2}, it has a unique cycle C. If a vertex of degree 1 leaves, then it is in the same component as C in H_{g-2} and at least one of its new neighbours is in this component as well. This yields a cycle of length at most $\frac{g}{2}+4$ or $\frac{g+1}{2}+3$ in H_{g+3}, which are contradictions to $g \geq 9$. Should the degree 0 vertex leave, we get that all three of its neighbours are in the component of H_{g-2} containing C and we find a new cycle of length at most $\frac{g}{3}+5$ or $\frac{g+1}{3}+4$. Again this contradicts $g \geq 9$ and the proof is complete. $\square$

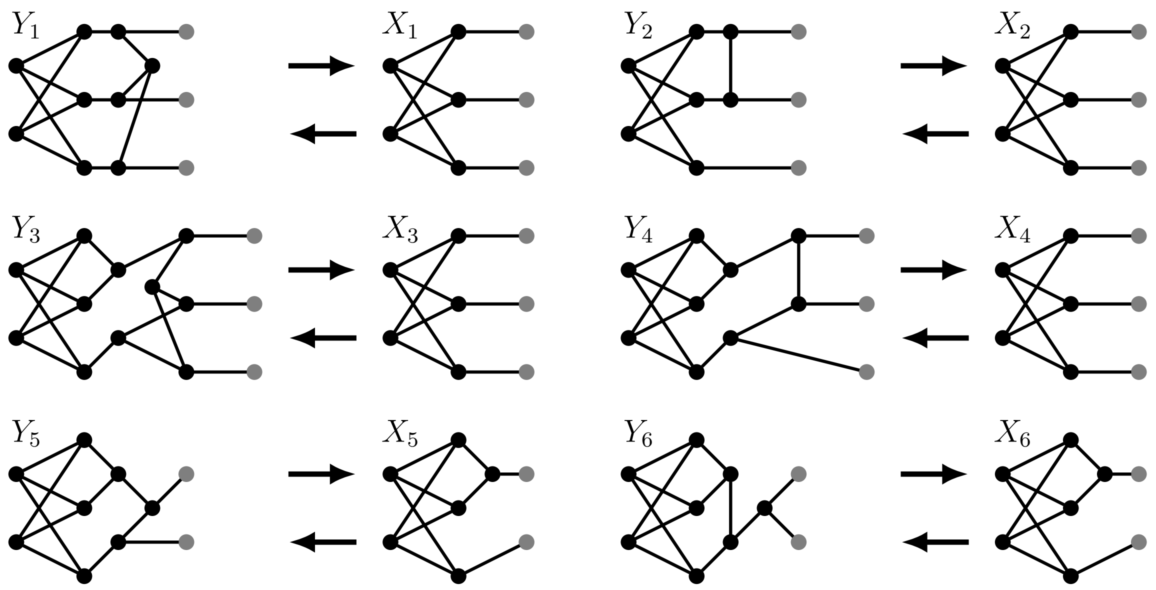


Figure 8.3.: Reductions and extensions for graphs of path-width 3 and girth 4.

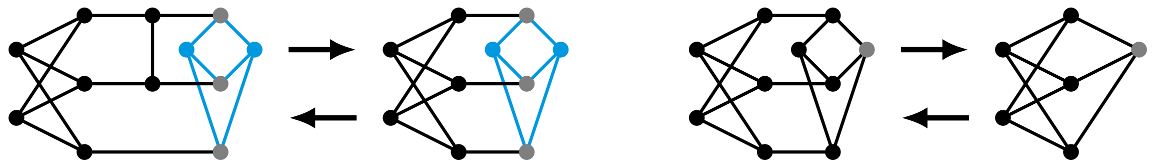


Figure 8.4.: Obtaining the $K_{3,3}$ using the reductions in Figure 8.3. Left: a path-width 3 graph of girth 4. It contains $c\,Y_2$, consisting of the black vertices, as an induced subgraph. Applying the (Y_2, X_2)-reduction yields a smaller graph. Right: this smaller graph contains $c\,Y_1$ as an induced subgraph and applying the (Y_1, X_1)-reduction results in a $K_{3,3}$. Reversing this process shows how to construct the start graph from a $K_{3,3}$.

8.3. Classifying cubic graphs of path-width 3 and girth 4

As a further example of an application of our algorithm, we prove the following theorem, which gives a constructive characterisation of the class of all cubic graphs of path-width 3 which are extremal with respect to ξ.

8.12 Theorem. [2.10] [2.11] [8.14] [8.15] (8.16)

(a) Every cubic graph of path-width 3 and girth 4 can be constructed from a $K_{3,3}$ by a finite number of (X_i, Y_i)-extensions from Figure 8.3, for $i \in \{1, \dots, 6\}$.

(b) Every 3-connected cubic graph of path-width 3 and girth 4 can be obtained from a $K_{3,3}$ by a finite number of (X_1, Y_1)- and (X_2, Y_2)-extensions from Figure 8.3. $\triangleleft$

We recall Definition 5.2 for the notion of transformations and refer to Figure 8.4 for an example of this constructive characterisation.

For the proof of Theorem 8.12 we establish the following lemmas.

[2.12] (8.14) (8.15) **8.13 Lemma.** If G is a subcubic graph of girth 4 with at most two vertices of degree 2 and none of degree less than 2, then G has a minor $M \cong K_4$. ◁

Proof. Note that all cubic graphs have a K_4-minor by Wormald's characterisation of cubic graphs. Since G has girth 4, the neighbours of every vertex are non-adjacent. Thus, if G has exactly one degree 2 vertex, we can remove it and join its two neighbours by an edge. The result is a cubic graph and, hence, has the desired minor.

This also works if G has two degree 2 vertices u and u' unless $Nu = Nu' = \{v, v'\}$. In the latter case, we proceed inductively: either $G - \{u, u', v, v'\}$ is a smaller subcubic graph with two degree 2 vertices, or it has a single degree 1 vertex w and only degree 3 vertices otherwise. In the first case, the claim follows by induction and, in the remaining case, we delete w to end up with one vertex of degree 2 and obtain the desired minor by the first part of the proof. □

[7.21] [7.22] [8.13] (8.12) **8.14 Lemma.** Let uv be a cut-edge in a connected cubic graph G of path-width 3 and girth 4. Furthermore, let K_u, K_v be the components of $G - uv$ containing u, v, respectively. If $Nu = \{v, u_1, u_2\}$, then there exists a path-decomposition of $K_u - u$ of width 3 whose first bag contains $\{u_1, u_2\}$. ◁

Proof. Let $Nv \coloneqq \{u, v_1, v_2\}$. Notice that K_v is a subcubic graph with exactly one vertex of degree 2. By Lemma 8.13 it contains a minor M with $M \cong K_4$. Hence, $H \coloneqq K_u \cup M + uw$, where $w \in VM$, is a minor of G and has path-width 3. Let $(P, \mathcal{V})$ be a smooth path-decomposition of H, then there exists a bag $V_i = VM$.

Since $H - VM$ is connected, i is an end of P, say $i = 1$. By Lemma 7.21 we may assume that the vertices in $VM \setminus \{w\}$ are the first to leave the path-decomposition. Using Lemma 7.22, we may further assume that the vertex w leaves next. Consequently, $V_5 = \{u, x_1, x_2, x_3\}$ for some x_1, x_2, $x_3 \in VH \setminus VM$.

If u leaves V_5, then $\{u_1, u_2\} \subseteq V_5$. Otherwise the vertex x_1 leaving V_5 has degree 3, so $Nx_1 = \{u, x_2, x_3\}$. In particular, x_1 is a neighbour of u, say $x_1 = u_1$. Again, by Lemma 7.22, we may assume that u leaves V_6. Replacing V_5 by $\{u, u_1, u_2, x_2\}$ and V_6 by $\{u_1, u_2, x_2, x_3\}$ yields another path-decomposition of H. Since $\{u_1, u_2\} \subseteq V_5$ in either case now, deleting the vertices of the M and u together with the first four (now empty) bags yields a path-decomposition of $K_u - u$ with the desired property. □

[7.21] [7.22] [8.13] (8.12) **8.15 Lemma.** Let $\{uv, u'v'\}$ be a 2-edge-separator of a connected cubic graph G of path-width 3 and girth 4 such that $uu' \notin E\,G$ and $u \neq u'$. Assume that the vertices u and u' are contained in the same component $K_{u,u'}$ of $G - \{uv, u'v'\}$. If $Nu = \{v, u_1, u_2\}$ and $Nu' = \{v', u'_1, u'_2\}$, then there exists a path-decomposition of $K_{u,u'} - u$ of width 3 whose first bag contains $\{u', u_1, u_2\}$ or there exists a path-decomposition of $K_{u,u'} - u'$ of width 3 whose first bag contains $\{u, u'_1, u'_2\}$. ◁

Proof. Let $K_{v,v'}$ be the component of $G - \{uv, u'v'\}$ containing v and v'. Similarly to the previous proof, we apply Lemma 8.13 to $K_{v,v'}$ to obtain a minor $M \cong K_4$. Hence,

$H = K_{u,u'} \cup M + \{uw, u'w'\}$ (where w and w' are vertices of M and $w = w'$ is possible) is a minor of G and has path-width 3. Let $(P, \mathcal{V})$ be a smooth path-decomposition of H, then there exists a bag $V_i = VM$.

Since $H - VM$ is connected, i is an end of P and we may assume that $V_1 = VM$. By Lemma 7.21 we may assume that the two or three vertices of M that have degree 3 in H are the first to leave. In the case that $w \neq w'$, we thus have $V_3 = \{w, w', x_1, x_2\}$. The vertices x_1 and x_2 cannot leave next: if x_1 left, $Nx_1 = \{w, w', x_2\}$, but w and w' have no common neighbours outside of M. (Here, we used that $u \neq u'$ by assumption.) Hence, we may assume that w leaves V_3 and, by Lemma 7.22, w' leaves V_4. As a result, the vertices of M are the first vertices to leave and $V_5 = \{u, u', y_1, y_2\}$.

In the case that $w = w'$, we can assume this is the case as well. As stated already, we may assume that the vertices of $M - w$ leave the first three bags and $V_4 = \{w, x_1, x_2, x_3\}$. If w leaves this bag, we are in the situation we want and, otherwise, we may assume that x_1 leaves. Consequently, $Nx_1 = \{w, x_2, x_3\}$ and we may assume that $x_1 = u$. Moreover, wx_2, $wx_3 \notin EG$ since G has girth 4. By Lemma 7.22, we may assume that w leaves V_5 and, thus, $V_5 = \{w, u', x_2, x_3\}$. By setting V_4 to $\{w, u, u', x_2\}$ and V_5 to $\{u, u', x_2, x_3\}$ we get another smooth path-decomposition of H in which the vertices of M are the ones to leave the first four bags.

Using that $V_5 = \{u, u', y_1, y_2\}$, we now obtain the decomposition asked for in the lemma. If u or u' leave V_5, then the remaining two vertices in the bag are its neighbours as $uu' \notin EG$ by assumption. Therefore, deleting the vertices of M together with u or u' and the first four bags yields the desired decomposition.

Otherwise, assume that y_1 leaves. Since $Ny_1 = \{u, u', y_2\}$, y_1 is a neighbour of u, say u_1. By Lemma 7.22 we may assume that u leaves $V_6 = \{u, u', y_2, y_3\}$. Since G has girth 4, $y_2u \notin EG$ and $y_3 = u_2$. By setting V_5 to $\{u, u', u_1, u_2\}$ and V_6 to $\{u', u_1, u_2, y_2\}$, we get another path-decomposition of H in which u leaves V_5. As above, this yields a desired decomposition. □

We are now ready to prove Theorem 8.12.

Proof (of Theorem 8.12). Let G be a cubic graph of path-width 3 and girth 4. Running Algorithm 7.2 for $\mathcal{G}$ being the class of cubic graphs of girth 4, $\mathcal{U} \coloneqq \{K_{3,3}, cY_1, \dots, cY_6\}$, and $k \coloneqq 3$ confirms that $\mathcal{U}$ is unavoidable for $\mathcal{G}^3$. If $G \cong K_{3,3}$, then we are done without needing to apply any extensions. Therefore, we may assume that G contains one of the graphs $cY_1, \dots, cY_6$ as a subgraph.

To be able to apply a (Y_i, X_i)-reduction, we need to ensure that such a subgraph is induced. We immediately notice that this is the case if G has cY_i as a subgraph for $i \neq 2$, since G has girth 4 and is not the $K_{3,3}$. For $i = 2$, two of the vertices of degree less than 3 may be adjacent, however, if this occurs, then G contains cY_6. Thus, we may, in fact, assume that G contains one of the graphs $cY_1, \dots, cY_6$ as an induced subgraph.

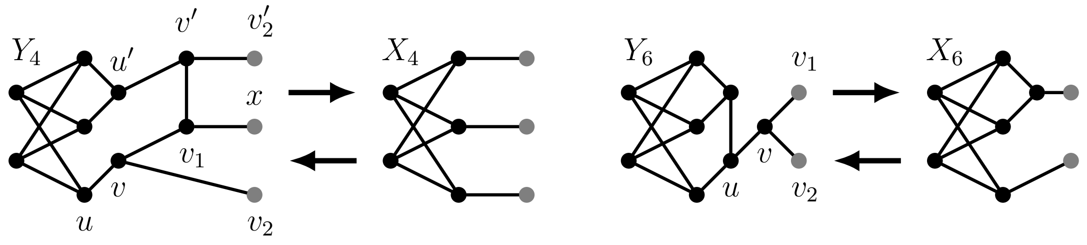


Figure 8.5.: Reductions ρ_4 and ρ_6 for graphs of path-width 3 and girth 4.

We fix $i \in \{1, \ldots, 6\}$ and assume that G' is obtained from G by a (Y_i, X_i)-reduction. Note that G' is connected and simple since the minimum degree in all $c\,X_i$ is 2. Its girth remains 4 since no triangles are introduced. If $i \in \{1, 2, 3, 5\}$, then G' is a minor of G and, hence, G' is of path-width at most, and therefore exactly, 3, see Observation 2.11.

For $c\,Y_6$ we denote the cut-edge by uv, as seen in the right of Figure 8.5, and the components of $G - uv$ by K_u and K_v. By applying Lemma 8.14, we get a path-decomposition of width 3 for $K_v - v$ that has $Nv \setminus \{u\} =: \{v_1, v_2\}$ in its first bags. This can now be turned into a path-decomposition of width 3 for G' by combining it with a path-decomposition of $c\,X_6$ whose last bag contains the neighbours of OX_6.

We use Lemma 8.15 to obtain that G' has path-width 3 if it was obtained from G by a (Y_4, X_4)-reduction. Let $\{uv, u'v'\}$ be the two edge separator in $c\,Y_4$ as illustrated on the left in Figure 8.5. We also refer to this figure for all upcoming vertex names.

Since uu', $vv' \notin E\,G$ and all four ends are distinct, the lemma is applicable to $K_{v,v'}$. Hence, we obtain a width 3 path-decomposition of $K_{v,v'} - v$ whose first bag contains $\{v', v_1, v_2\}$ or of $K_{v,v'} - v'$ whose first bag contains $\{v, v_1, v_2'\}$. By replacing v' by v_2', v_1 by x, and v by v_2 in all bags, we obtain a path-decomposition of $G - I\,Y_4$ whose last bag is $\{v_2, v_2', x\}$, in both cases. As before, we obtain a path-decomposition of width 3 for G' by combining it with a path-decomposition of $c\,X_4$ whose last bag contains the neighbours of OX_4.

With this, we have now determined that all six reductions in Figure 8.3 yield a new connected cubic graph G' which has path-width 3 and girth 4. Since $|G'| \leq |G| - 2$, we obtain inductively that G can be obtained from $K_{3,3}$ by a finite number of (X_i, Y_i)-extensions, proving Part (a). For Part (b), we note that the graphs $c\,Y_3, \ldots, c\,Y_6$ all exhibit a 2-edge-separator and cannot be a subgraph of G if G is 3-connected. Thus, if the remaining two reductions preserve 3-connectivity, we are done.

To see that they do, let G' be obtained from a 3-connected graph G by a (Y_i, X_i)-reduction for $i \in \{1, 2\}$. Let uv, $u'v'$ be two edges of G'. By Observation 2.10 it suffices to show that $G' - \{uv, u'v'\}$ is connected. This is, in particular, the case if neither edge is in $H \coloneqq G' - I\,X_i = G - I\,Y_i$.

We note that H is 2-edge-connected: let e be a bridge of H and K_1, K_2 the components of $H - e$. We obtain a 2-edge-separator of G since $E_{H-e}\,K_j \subseteq E\,H$ contains only a single edge for one of the two components K_j, contradicting the 3-connectivity of G.

Thus, if H contains just one of the two edges uv and $u'v'$, then these do not separate G'. If H contains both, then $H - \{uv, u'v'\}$ is either connected, and we are done, or it consists of two components. Since $G - \{uv, u'v'\}$ is not disconnected, each of these components has a neighbour in $c\,Y_i$ in G and, therefore, one in $c\,X_i$ in G', ensuring connectivity and completing the proof. □

We now know that with the six operations from Figure 8.3 we can construct all cubic graphs of path-width 3 and girth 4 *if we do it right.* We complete this section by showing that this is easy to do.

8.16 Theorem. Let G be the graph obtained from G' by an (X_i, Y_i)-extension from Figure 8.3. Then G is a cubic multigraph of path-width 3. It is simple and has girth 4 unless two outer vertices u, v of Y_i exist which satisfy [2.13] [8.12]

$$d_{Y_i}(u,v) + d_{G'}(u,v) \leq 3.$$

Moreover, if G' is 3-connected and $i \in \{1,2\}$, then G is 3-connected as well. ◁

Proof. To see that the girth remains 4, we note that $G - I\,Y_i$ and Y_i contain no triangles. By the requirement that $d_{Y_i}(u,v) + d_{G'}(u,v) \geq 4$, none exist between them either. This also implies that G is simple.

Let F be the set of the (one or three) possible edges between the outer vertices of X_i in G' and let $H \coloneqq G' - I\,X_i$. The graph $H + F$ is a minor of G', so G' has a path-decomposition with a bag containing OX_i. This bag is at an end, since OX_i is not a separator of G'. Similarly to the proof of Theorem 8.12, we can extend this decomposition to G.

For the 3-connectivity claim, we just note that the transition from X_2 to Y_2 performs Operation (i) from Tutte's characterisation once, and the one from X_1 to Y_1 does it twice. Thus, by said characterisation, both resulting graphs are 3-connected. □

Theorem 8.16 shows that we only need to obey minor restrictions when applying the extensions: the first and third are always applicable, the second and fourth require the top two outer vertices to be distinct in G', the fifth also needs this for its outer vertices, and the sixth even wants them at distance at least 2. Additionally, only applying the first two extensions ensures that we do not leave the class of 3-connected graphs.

Part III.

Two Graph Connectivity Problems

In this part we consider two graph connectivity problems. More precisely, we are concerned with a digraph D that has two designated vertices s and t. We want to know something about the existence or non-existence of (certain) st-paths in D: in Chapter 9 we study the complexity of locally certifying the existence of an st-path and see that a constant amount of bits does not suffice for this task (in contrast to the undirected case, where a single bit is enough). In Chapter 10 we look at the complexity of the separating by forbidden pairs problem, which asks for a smallest set of (unordered) arc pairs in D such that every st-path contains both arcs from some pair. We prove that this problem is $\Sigma_2\mathsf{P}$-complete.

Chapter 9.

Local Certification of Reachability

Motivated by self-stabilising algorithms, the objective of local certification is to verify whether a (global) property holds while being restricted to a local view of the graph. For example, despite being a global property, bipartiteness is easy to verify locally. By specifying a bipartition, a vertex can simply check that all its neighbours are assigned to the other part. If no vertex encounters a problem, then the graph is bipartite.
As the example suggests, local certification, roughly speaking, works as follows: a prover assigns a certificate to every vertex of a graph. Subsequently a verifier checks, for every vertex, whether its local view of the graph is consistent with the property we wish to verify. If this is the case at all vertices, then it accepts and it rejects otherwise. For a prover-verifier pair to locally certify a graph property, the verifier must accept all graphs with this property given the certificates from the prover and it must reject any proof on a graph that does not have the desired property. The quality of such a pair is measured by the prover's size which is the length of the longest certificate it uses.
One local certification concept is that of proof labelling schemes and, in this setting, determining whether an undirected graph has an st-path (for specified vertices s and t) can be done with a prover of size 1. In the directed case it is known that $\mathcal{O}(\log \Delta)$ bits suffice. We prove a matching lower bound, showing, in particular, that a constant number of bits are insufficient.

The objective of local certification is to locally verify (global) properties in a distributed system. It is motivated by self-stabilising algorithms [Dol00]. These algorithms are used in distributed systems which are subject to faults and have the property that they converge to a solution for a given problem. A possible way of designing such an algorithm is to first move to a solution and then to maintain it for as long as it remains correct. This so-called local detection paradigm was introduced by Afek, Kutten, and Yung [AKY97]. Local certification describes this last step, where the algorithm needs to detect whether the solution is correct or not.

In local certification, a global prover has to convince a verifier that a graph has a specific property. To do so, the prover first presents a proof by assigning certificates to the vertices. Afterwards, the verifier decides at every vertex whether to accept or reject the proof provided. The decision at a vertex v is solely based on the local view of the graph around v, including certificates. If a graph has the designated property, the prover must be able to choose certificates in a way that makes the verifier accept at every vertex. Otherwise, the verifier must reject at some vertex of the graph, regardless of which certificates are given. What exactly the term local view means differs amongst the various local certification concepts.

As an illustration, we sketch how local certification of bipartiteness works [GS16]. As local view, we assume that every vertex has access to its own certificate as well as those of its neighbours. In the case of a bipartite graph, the prover could specify a bipartition using the certificates 0 and 1. The verifier then only needs to check that the certificates of its neighbours differ from its own. If this is the case everywhere, then the graph is bipartite, so the verifier never accepts a non-bipartite graph. Furthermore, the bipartition specified by the prover makes the verifier accept. Hence, this prover-verifier pair locally certifies bipartiteness using a single bit. As a slightly more interesting example, we also present the proof labelling scheme for acyclicity from [KKP10] at the start of Section 9.1.

Local certification does not restrict the computational power of the prover or verifier. Instead the quality is measured by the certificate lengths needed. The prover's size is the length of the longest certificate it assigns to a vertex. Given some property, a natural question to consider is what size a prover needs to have and what size suffices to locally certify this property.

The answer to this question may depend on the precise concept used since a 'larger' local view of the verifier may result in smaller certificate lengths being sufficient. Here, we are interested mainly in two such concepts: proof labelling schemes and locally checkable proofs, which we defined in Section 2.3. These were introduced by Korman, Kutten, and Peleg [KKP10] and Göös and Suomela [GS16], respectively. Recall that the verifiers in locally checkable proofs are more powerful than the ones in proof labelling schemes. Thus, ideally, one determines lower bounds on the prover's size using locally checkable proofs, and upper bounds using proof labelling schemes (since then they hold for both concepts).

Many lower and upper bound results are known, for example, certifying acyclicity [KKP10] and planarity [Feu+20] both require $\Theta(\log n)$ bits, whilst minimum spanning trees need $\Theta(\log n \log W)$ bits [KK06], where n is the order of the graph and W is the largest weight of an edge. These bounds are for proof labelling schemes, though the planarity result also works for locally checkable proofs. That $\Theta(\log n)$ bits are required for acyclicity is also true in the locally checkable proof setting was open until recently and is, in fact, also shown in [Feu+20]. For more results we refer to [Feu21], a recent survey of local certification.

A very basic problem is that of certifying whether an st-path exists, for two specified vertices s and t, which we refer to as the st-reachability problem. In [GS16] it is shown how to solve this for undirected graphs using a single bit. In the directed case, a prover using $\mathcal{O}(\log \Delta)$ bits is described. Whether a constant size proof exists is posed as an open question. We recapitulate how these locally checkable proofs work in Section 9.1 and why the directed case cannot be solved analogously.

Foerster et al. [Foe+18] look at locally certifying *st*-reachability as well. The authors use a significantly more restrictive model in which the verifier sees less information, making it weaker. To obtain a logarithmic lower bound for *st*-reachability, they additionally restrict to a one-way communication model in which vertices only see their predecessors. The proof heavily relies on this restriction and a lower bound for the standard two-way communication is left open.

This suggests that the directed version is harder than the undirected one, which is a common occurrence: in the k disjoint paths problem the task to find vertex-disjoint paths between k pairs of specified vertices. It can be solved in polynomial time on undirected graphs for any fixed k [RS95] but it is NP-complete in the directed case, even for $k = 2$ [FHW80]. Similarly, the feedback arc set problem is NP-complete [GJ90] while its undirected counterpart is solved by computing a spanning tree. Ajtai and Fagin [AF90] demonstrate that undirected *st*-reachability can be expressed by existential monadic second order logic whereas directed reachability cannot be.

We show that locally certifying *st*-reachability (in the proof labelling scheme setting) is another such example. More precisely, we prove that the upper bound of $\mathcal{O}(\log \Delta)$ is tight. In particular, this covers the missing lower bound in [Foe+18] and shows that no constant number of bits suffice. Hence, this yields a new very basic problem where we now have tight bounds for proof labelling schemes, but whose lower bounds do not extend to locally checkable proofs.

This chapter is joint work with Tim Bergner and Sven O. Krumke and is published in [BBK22b].

9.1. Two examples of proof labelling schemes

We start this chapter by presenting two examples for proof labelling schemes, starting with acyclicity and then moving on to the reachability problem we are interested in.

Verifying acyclicity. We have already seen how to certify bipartiteness and have mentioned in the introduction that $\Theta(\log n)$ bits are needed for verifying acyclicity on undirected graphs. Let us now sketch the upper and lower bound proofs presented in [KKP10], to which we refer for the details. We note that the lower bound proof does not extend to locally checkable proofs (but [Feu+20] provides one).

The upper bound is obtained by rooting a tree: we choose an arbitrary vertex as the root and the certificate at each vertex is its distance to this root. To locally verify this, a vertex with certificate d need only check that it has exactly one neighbour with distance $d-1$ and all other neighbours have distance $d+1$ (unless $d=0$ in which case all neighbours must have distance 1). This is a proof labelling scheme since the graph is a tree if the verifier accepts: in any cycle in the graph, the vertex with largest certificate has at least two neighbours whose certificates are not larger, which the verifier can detect.

To obtain the lower bound, assume that a proof labelling scheme exists that uses $o(\log n)$ bits. Then, for large enough n, we can assume it has fewer than $c \log n$ bits for some appropriate constant c. By choosing n large enough, we can ensure that any path of length n contains two disjoint pairs of adjacent vertices (u_1, v_1) and (u_2, v_2) with the property that u_1, u_2 and v_1, v_2 are assigned the same certificate, respectively. By connecting the second vertex of the first pair to the first vertex of the second pair, a cycle is obtained in which all local behaviour is identical. Since the path must be accepted, the verifier also accepts the cycle, contradicting soundness.

The reason this no longer works for locally checkable proofs is because the stronger verifier there can see the identities of its neighbours. As a result, the verifier would notice that its neighbour has changed in the transition to the cycle, since the ends of the new edge now have a neighbour whose identity differs from the path.

Verifying reachability. In order to discuss the reachability problem, let us first formally define it: the *(directed) st-reachability problem* starts with the graph class $\mathcal{G}$ of (directed) graphs in which there is a unique vertex labelled s and a vertex labelled t (and all other vertices have empty labels). The subclass $\mathcal{F}$ that we wish to verify contains those graphs in which t is reachable from s, that is, those graphs in $\mathcal{G}$ that have an st-path. For the remainder of this chapter, $\mathcal{G}$ and $\mathcal{F}$ will be used to denote these graph classes.

In the undirected case, a single bit is sufficient to verify reachability (as described in [GS16]): by fixing some shortest st-path P and setting $\mathbf{P}\,v = 1$ if $v \in P$ and $\mathbf{P}\,v = 0$ otherwise, a vertex v can check that either $\mathbf{P}\,v = 0$ or exactly two of its neighbours are also assigned a 1 as their certificate. The vertices s and t are exceptions, they must receive certificate 1 and require exactly one neighbour with this property.

This breaks down in the directed case, since the analogous requirement of asking for exactly one predecessor and one successor does not yield a proof labelling scheme. The reason this works in the undirected case is because, on shortest paths, vertices have a unique predecessor and successor on this path: multiple ones would create a shortcut. In the directed case, however, the existence of back-edges may lead to larger quantities of both, even on shortest paths. This can be fixed by additionally specifying the distance from s (as in the verification of acyclicity), letting us detect back-edges. Alternatively, one can specify which of the edges incident to a vertex leads to the successor on the path. These yield upper bounds of $\mathcal{O}(\log(\text{diam}))$ and $\mathcal{O}(\log \Delta)$, both of which are potentially $\mathcal{O}(\log n)$ many.

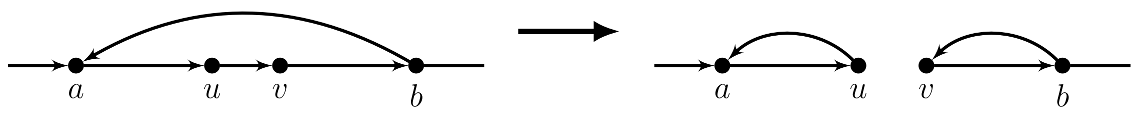


Figure 9.1.: The split-path operation.

9.2. A lower bound for the reachability problem

In this section, we show that the second approach, where the prover specified which outgoing edge the path uses, is best possible in the sense that $o(\log \Delta)$ bits are insufficient to obtain a proof labelling scheme in the directed case. To achieve this, we assume that a proof labelling scheme $(\mathcal{P}, \mathcal{V})$ exists that only uses x bits and therefore uses at most $c := 2^x$ distinct certificates. We construct a digraph $D \in \mathcal{G}$ that the verifier would falsely accept and check that its maximum degree is polynomial in c. This yields that $\log \Delta$ is some multiple of x, and $x \in \Omega(\log \Delta)$.

Our construction works even if we can detect parallels and anti-parallels and can see the labels of neighbouring vertices. It also does not use the relation between c and x. Therefore, we suppose that $(\mathcal{P}, \mathcal{V})$ is a prover-verifier pair with neighbourhood-local and identity-restricted verifier that locally certifies directed st-reachability using c distinct certificates. To facilitate our argumentation, we now provide some notation.

Split paths and their properties. Let $\overleftarrow{P}$ be an st-path with back-edges, that is, $P := sv_1 \dots v_k t$, $\overleftarrow{A} := \{v_i v_j \colon i > j\}$, and $P \subseteq \overleftarrow{P} \subseteq P + \overleftarrow{A}$ (for some k). Note that $\overleftarrow{P} \in \mathcal{F}$. Let uv be an edge of P and ba be a back-edge in $\overleftarrow{P}$ with $uv \in \mathring{a}P\mathring{b}$. The *split-path* (of $\overleftarrow{P}$) at uv using ba is the digraph $\overleftarrow{P} - \{uv, ba\} + \{bv, ua\}$, which is in $\mathcal{G} \setminus \mathcal{F}$. This operation is illustrated in Figure 9.1.

We now show that assigning certain certificates to the vertices of a path with back-edges would make $\mathcal{V}$ accept a split-path, and hence these assignments may not occur. For simplicity, we write $\mathbf{P}\,xy$ for $(\mathbf{P}\,x, \mathbf{P}\,y)$, where $\mathbf{P}$ is some proof and xy is an edge.

9.1 Lemma. Let $\overleftarrow{P}$ be an st-path with back-edges, $\mathbf{P} = \mathcal{P}\,\overleftarrow{P}$, uv be an edge of P, and ba be a back-edge in $\overleftarrow{P}$ with $uv \in \mathring{a}P\mathring{b}$. If $v \notin Nb$, $u \notin Na$, and $vu \notin \overleftarrow{P}$ (in the underlying graph), then $\mathbf{P}\,uv \neq \mathbf{P}\,ba$. $\triangleleft$ (9.2)

Proof. Suppose $\mathbf{P}\,uv = \mathbf{P}\,ba$. Let S be the split path of $\overleftarrow{P}$ at uv using ba. We show that $\mathcal{V}$ accepts $\mathbf{P}$ for S, which contradicts $S \in \mathcal{G} \setminus \mathcal{F}$.

Recall that the graphs $\overleftarrow{P}_x^N$ and S_x^N are the local views the verifier has access to as defined in Section 2.3. Notice that for all vertices $x \notin \{a, b, u, v\}$ the digraphs $\overleftarrow{P}_x^N$ remains unchanged in the transition to S_x^N and thus $\mathcal{V}$ accepts at all these vertices. In the remaining four balls we only exchange the vertices a with v and b with u. Since these are assigned the same certificate by assumption, the verifier is faced with the same certificates. Moreover, none of these vertices can be s or t, so they are all unlabelled. Finally

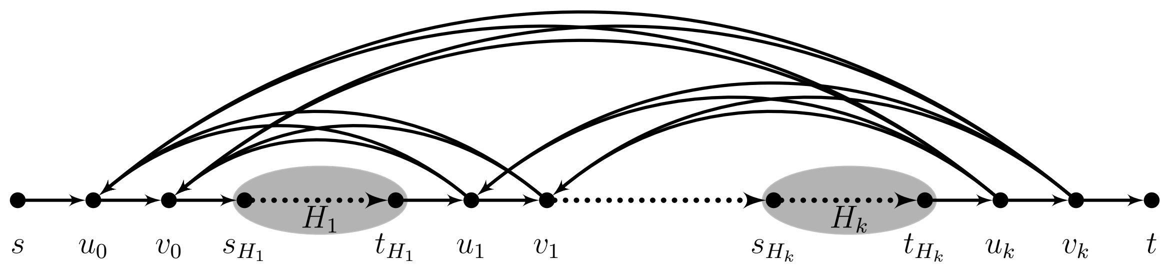


Figure 9.2.: The digraph D_k constructed for the proof of Theorem 9.4. It consists of k copies $H_1, \dots, H_k$ of D_{k-1} which are paths with back-edges, together with the vertices and edges shown, making it a path with back-edges as well.

note that the remaining assumptions of the lemma ensure that none of the edges uv, ba, bv, or ua have a parallel or an anti-parallel in $\overleftarrow{P}$ or S, hence the digraphs $\overleftarrow{P}_x^N$ and S_x^N also coincide for $x \in \{a, b, u, v\}$. □

Constructing a counterexample. We are now ready to construct a path with back-edges $\overleftarrow{P}$ in which any proof that the verifier accepts leads to a split path which is accepted as well. Since the verifier accepts $\mathcal{P}\,\overleftarrow{P}$, this is a contradiction.

We let $r \coloneqq \binom{c}{2} + c$ and define a digraph D_k for $0 \leq k \leq r$, each of which is an st-path with back-edges. For a copy H of a digraph D_k, we write P_H for the copy of the st-path of D_k in H and write s_H, t_H for its start and end, respectively. The digraph D_0 is the path su_0v_0t. For $k \geq 1$, the digraph D_k is the disjoint union of k copies $H_1, \dots, H_k$ of D_{k-1} which are combined to an st-path with back-edges as follows: copies H_i and H_{i+1} are connected by a path $t_{H_i}u_iv_is_{H_{i+1}}$ introducing new vertices u_i and v_i. Additionally, we prepend the path $su_0v_0s_{H_1}$ and append the path $t_{H_k}u_kv_kt$. Finally, all possible back-edges between vertices $\{u_i, v_i\}$ and $\{u_j, v_j\}$ for $i > j$ are added.

This construction is visualised in Figure 9.2 and a formal definition of the edge and vertex set of the digraph D_k is given below:

$$\begin{aligned}
V D_k &\coloneqq \bigcup_{i=1}^{k} V H_i \cup \{u_i, v_i \colon\ 0 \leq i \leq k\} \cup \{s, t\}\,, \\
A\, D_k &\coloneqq \bigcup_{i=1}^{k} A\, H_i \cup \{u_iv_i \colon\ 0 \leq i \leq k\} \\
&\qquad\qquad \cup \{v_{i-1}s_{H_i}, t_{H_i}u_i \colon\ 1 \leq i \leq k\} \cup \{su_0, v_kt\} \\
&\qquad\qquad \cup \{x_ix_j \colon\ x_i \in \{u_i, v_i\}\,, x_j \in \{u_j, v_j\}\,, i < j\}\,.
\end{aligned}$$

We say that a pair of certificates (c_1, c_2) is *missing* on a path with back-edges if no edge on the path has certificate (c_1, c_2). We extend this definition to sets $\{c_1, c_2\}$, where c_1 and c_2 need not be distinct, and call such a set missing if the tuples (c_1, c_2) and (c_2, c_1)

are. Note that r is exactly the number of such sets. With these definitions at hand we can now prove the following lemma.

9.2 Lemma. For every $k \leq r$, the digraph D_r contains a copy H of D_k in which at least $r - k$ sets are missing on the path $\mathring{P}_H$. $\triangleleft$ [9.1] (9.4)

Proof. We prove this by induction on k, where in the case $k = r$ there is nothing to show. For $k < r$ let H' be the copy of D_{k+1} given by the induction hypothesis and let $P' := \mathring{P}_{H'}$. Then $r - (k+1)$ sets are missing on P' and every vertex on this path is assigned a certificate whose corresponding set is amongst the remaining $k+1$ many. Since D_{k+1} has the $k+2$ edges $u_0v_0, \ldots, u_{k+1}v_{k+1}$, at least two of the corresponding edges in P' are assigned certificates that give rise to the same set $\{c_1, c_2\}$. For simplicity, we assume these edges are u_0v_0 and u_1v_1. The set $\{u_0v_0, u_1v_1\}$ is not missing in P', but we show that it is missing on the path $P := \mathring{P}_H$ where H is the copy of D_k between u_0v_0 and u_1v_1 in H'.

To see that this is indeed the case, we note that for any edge uv on the path P we can apply Lemma 9.1 to the edges uv and $ba := u_1u_0$ in D_r: since the only edges with an end in $VH' \setminus VH$ and the other in VH are v_0s_H and t_Hu_1, we get that $v \notin Nu_1$ and $u \notin Nu_0$. Moreover, $vu \notin D_r$ since D_r has no anti-parallels by construction. Therefore, the assumptions of Lemma 9.1 are satisfied, and $\mathbf{P}\,uv \neq \mathbf{P}(u_1u_0)$. The same holds for the back-edges u_1v_0, v_1u_0, and v_1u_1.

Since we assumed that both u_0v_0 and u_1v_1 are assigned a certificate corresponding to the set $\{c_1, c_2\}$, one of the four back-edges has certificate (c_1, c_2) and another has (c_2, c_1). Thus, we have ensured that both of these are missing, giving us the new missing set $\{c_1, c_2\}$ (in addition to the $r - (k+1)$ many provided by the induction hypothesis), which completes the proof. $\square$

By Lemma 9.2 for $k = 0$, D_r has a copy H of D_0 in which r pairs are missing on the path $P := \mathring{P}_H$. But since these are all possible pairs, the single edge in P has no valid assignment, which is a contradiction. To complete this section, we only need to see how c relates to $\Delta\, D_r$.

9.3 Observation. The maximum degree of D_r is $2r + 2 = c^2 + c + 2$. $\triangleleft$ (9.4)

Proof. We prove by induction that the maximum degree of D_k is $2k + 2$. Since D_0 is a path, this holds initially. For $k > 0$ let v be a vertex in D_k. If v is in some copy H of D_{k-1}, then its degree is at most $2k$: in H only the vertices s_H and t_H have incident edges to vertices outside of this copy of H, and these have degree 2 in D_k.

The vertices of D_k that are not in a copy of D_{k-1} are s, t, and those in $\{u_0, v_0, \ldots, u_k, v_k\}$. The first two have degree 1 and every u_i and v_i has two neighbours on the path and $2k$ further neighbours in D_k, namely all u_j and v_j for $j \in \{0, \ldots, k\} \setminus \{i\}$. Hence, the maximum degree of D_k is $2k + 2$.

The missing equality follows from the definition of r. $\square$

By Observation 9.3, $\Delta\, D_r$ is, indeed, polynomial in c: at least $\log_2(c+1)$ bits are required to obtain $c+1$ distinct certificates. From simple computations we get that $\log_2(\Delta\, D_r) \leq 2\log_2 c + 2$. Consequently, since we need at least $c+1$ certificates to correctly verify D_r, at least $\frac{1}{2}\log_2(\Delta\, D_r) - 1$ bits are necessary. We have thus arrived at the desired result.

[9.2] [9.3] (9.5) **9.4 Theorem.** The *st*-reachability problem cannot be locally certified by a prover-verifier pair $(\mathcal{P}, \mathcal{V})$ of size $o(\log(\Delta\, D))$ if the verifier is neighbourhood-local and identity-restricted. $\triangleleft$

As a corollary, we get the same result for proof labelling schemes.

[9.4] **9.5 Corollary.** There exists no proof labelling scheme of size $o(\log(\Delta\, D))$ for directed *st*-reachability. $\triangleleft$

Chapter 10.

Separating by Forbidden Pairs

By Menger's theorem, the maximum number of arc-disjoint paths from a vertex s to a vertex t in a directed graph is equal to the minimum number of arcs needed to disconnect s and t, that is, the minimum size of an st-cut. The max-flow problem in a network with unit capacities is equivalent to the arc-disjoint paths problem. Moreover, the max-flow and min-cut problems form a strongly dual pair. We relax the disjointedness requirement on the paths, allowing them to be almost disjoint, meaning they may share up to one arc. The resulting almost disjoint paths problem (ADP) asks for k st-paths such that they share at most one arc pairwise. The separating by forbidden pairs problem (SFP) is the corresponding dual problem and asks for a set of k arc pairs such that every st-path contains both arcs of at least one such pair.
In this chapter, we analyse the complexity of these two problems by giving an overview of results for ADP and proving that SFP is $\Sigma_2\mathsf{P}$-complete, even for acyclic graphs.

In many applications, customers receive various offers from which they can select one or between which they can switch. Usually, these offers should be as diverse as possible to provide the customer with many different options. A common use case is the construction of alternative routes in transportation or road networks. These make sense in this context, for example to avoid route closures, heavy traffic, or tolls. Another example where alternative routes are of use is to distribute risk. For example, if dangerous goods need to be transported regularly, alternative routes that affect different people allow for an equal risk distribution amongst the people exposed. Several practical algorithms for computing alternative routes have been developed, see, for example, [Abr+10, AEB00, Bad+11, DGS05, Jeo+09].

On the graph-theoretic side, the (arc- or vertex-) disjoint paths problem is well-studied. Determining a maximum number of disjoint st-paths can easily be done using maximum flow techniques [AMO93]. By Menger's theorem, this number is equal to the minimum number of arcs needed to separate s from t. This result is implied by the max-flow min-cut theorem, which shows that these two problems form a strongly dual pair.

The following extension of the disjoint paths problem is also well-understood: given k pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$, the objective is to find disjoint $s_i t_i$-paths. For undirected graphs it is solvable in polynomial time if k is constant (see [RS95] for a cubic and [KKR12] for a quadratic algorithm) and NP-complete in general [EIS76]. In the case of directed graphs, a single path is easy to find and two paths are already NP-complete [FHW80]. Vygen [Vyg95] showed that the problem remains NP-complete for few paths even on very restricted graph classes like acyclic, Eulerian, or planar graphs.

Another possible extension is to ask for k disjoint st-paths that are short, which again makes sense for routing purposes. Suurballe [Suu74] describes an algorithm for this problem that is based on shortest path labellings. It is possible to combine both extensions and ask for shortest paths between different terminals. Eilam-Tzoreff [Eil98] shows that these problems in all configurations (for directed and undirected graphs with vertex- or arc-disjoint paths) are NP-complete and provides a polynomial algorithm for two paths in an undirected graph with positive edge-weights. Berczi and Kobayashi [BK17] present a polynomial algorithm for the directed version, also for the case of two paths and positive arc-lengths.

In contrast, the same problem where the paths need not be completely disjoint has not garnered as much attention in the literature. There are several natural relaxations of the disjointedness condition: a first option allows arcs to be part of more than one path, say each arc may be used by two. This problem can be solved in the same way as the disjoint paths problem, by a maximum flow computation in the graph with arcs of capacity 2. Another alternative is to allow some arcs, say one, to be part of an arbitrary number of paths. This, too, can be solved by maximum flow techniques, but requires one flow computation for each arc. These flows are computed in the graphs where one arc's capacity is set to infinity (and the rest remain at capacity 1).

The choice we consider here uses the following relaxation:

10.1 Definition. A set of paths is *almost disjoint* if every two paths in the set share at most one edge. $\triangleleft$

With this definition, we arrive at the problem below.

10.2 Problem. The *almost disjoint paths problem* (ADP) is given by a digraph D together with two designated vertices s, $t \in VD$ and a natural number $k \in \mathbb{N}$. The question is whether a set of k almost disjoint st-paths exist. $\triangleleft$

Despite being a natural choice, this problem has not been previously studied. We give an overview of complexity results regarding ADP in Section 10.1.

Most of the literature on nearly disjoint paths is of a very practical nature as is evidenced by the initial examples we presented. We now discuss some of the (rarer) theoretical results that exist for problems similar to ADP. Liu et al. [Liu+18] introduce the k shortest paths with diversity problem, in which the goal is to find a set of sufficiently dissimilar paths of maximum size (bounded by k). Of such sets, the one that minimises the total path length is optimal. For this problem, an incorrect NP-hardness proof as well as a greedy framework is presented. Chondrogiannis et al. [Cho+18] fix said NP-hardness proof, showing that the problem is indeed strongly NP-hard, and develop an exact algorithm for it as well as heuristics. Moreover, Chondrogiannis et al. [Cho+20] consider the problem of finding k shortest paths with limited overlap. They prove that this variant is weakly NP-hard and develop two exact algorithms for it. The problems here are similar to ADP in the sense that they look for paths that are sufficiently dissimilar, though the measures used always result in similarity values between 0 and 1 because they compare the number of arcs in common with some function based on the lengths of the two paths. Additionally, they want to minimise the total length of the paths found.

Inspired by the strong duality of max-flow and min-cut, we make analogous considerations for our almost disjoint paths problem. By dualising the linear relaxation of an integer programming formulation, we obtain a dual problem whose integer version we call separating by forbidden pairs (SFP). Its goal is to select as few arc pairs as possible such that every st-path in G contains both arcs of at least one chosen pair.

10.3 Definition. Let D be a digraph with two designated vertices s, $t \in VD$. A set of (unordered) arc pairs $\mathcal{A} \subseteq \binom{AD}{2}$ separates s and t in D if every st-path contains both arcs of at least one pair in $\mathcal{A}$. ◁

Again, we formalise SFP in the following problem statement.

10.4 Problem. The *separating by forbidden pairs problem* (SFP) is given by a digraph D, two designated vertices s, $t \in VD$, and a natural number $k \in \mathbb{N}$. The question is whether a set $\mathcal{A}$ of k arc pairs exists which separates s and t in D. ◁

While the linear programming relaxations of these problems form a dual pair and thus have the same objective value [GKT51], the corresponding integer versions are only weakly dual. Note that in the min-cut problem we select an arc on every st-path whereas in SFP we select a pair of arcs on every st-path.

Apart from being dual to ADP, SFP adds another level on top of the well-known path avoiding forbidden pairs problem (PAFP). In the latter problem one is given a set of arc pairs $\mathcal{A}$ and has to identify whether some st-path avoids all pairs in $\mathcal{A}$ (meaning SFP asks for a set that makes the corresponding PAFP instance unsolvable). Originating from the field of automated software testing [KSG73], PAFP also has applications in aircraft routing [Bla+15] and biology, for example in peptide sequencing [Che+01] or predicting gene structures [KVB09]. PAFP is NP-complete [GMO76] and various restrictions on the set of forbidden pairs have been considered. The problem becomes solvable if the pairs satisfy certain symmetry properties [Yin97] or if they have a hierarchical structure [KP09] while it remains NP-hard even if the pairs have a halving structure [KP09] or no two pairs are nested [Kov13]. The structure of the PAFP polytope has been analysed [Bla+15] and Hajiaghayi et al. [Haj+10] show that determining a path that uses a minimal number of forbidden pairs cannot have a sublinear approximation algorithm.

Note that some of the referenced papers consider forbidden pairs of vertices instead of pairs of arcs. However, these two variants can be converted into one another by standard constructions. In Section 10.2 we show that SFP is $\Sigma_2\mathsf{P}$-complete, even on acyclic digraphs (which is a natural restriction for the PAFP that most authors assume there).

This chapter is joint work with Tim Bergner and Sven O. Krumke and a preprint is available on arXiv [BBK22a].

10.1. An overview of ADP

Chapter setup. As the title of this paragraph indicates, we provide some global conventions for the entire chapter here.

10.5 Convention. In this chapter, D is a directed graph with designated vertices s and t. ◁

We also make the following two assumptions on D, that are without loss of generality for both problems.

10.6 Assumption. Let D be a digraph with designated vertices s and t.

(i) The direct arc st is not contained in D.
(ii) Every arc and every vertex of D is contained in an st-path. ◁

If the direct arc st is contained in the graph, this arc itself forms an st-path. With respect to ADP this path is disjoint from every other st-path and can always be added to a set of almost disjoint paths. (Note that it is even almost disjoint from itself, so if we did not require the solution to be a set of paths, it could be included arbitrarily often.) Regarding SFP this is a path that never contains a forbidden pair as it only has length 1. Thus, in every instance containing the direct arc st, s and t cannot be separated. This justifies Assumption 10.6 (i).

For both problems, ADP as well as SFP, every arc and every vertex that is not contained in any st-path is irrelevant and can be removed. Hence, we assume such arcs and vertices do not exist as stated in Assumption 10.6 (ii).

Duality. We noted that the problems ADP and SFP form a pair of weakly dual problems, meaning they have integer programming formulations whose LP-relaxations are dual to one another. In contrast to Menger's theorem, these problems have an unbounded duality gap, as the following lemma shows.

10.7 Lemma ([BBK22a, Lemma 2.4]). *The duality gap between* ADP *and* SFP *is unbounded.* ◁

However, the duality gap is not always unbounded. For example, if we restrict ourselves to graphs that have an *st*-cut with a single outgoing arc, the duality gap disappears [BBK22a, Lemma 2.5].

Complexity. The almost disjoint paths problem is in NP, since verifying that paths are almost disjoint is easy. If we restrict k to being constant, the problem becomes solvable in polynomial time, by making use of a dynamic program.

10.8 Theorem ([BBK22a, Theorem 1.1]). *For constant k,* ADP *is polynomial-time solvable.* ◁

For $k = 2$, a quadratic algorithm is obtained by using maximum flow techniques [BBK22a, Lemma 3.1]. If k is not a constant however, the problem becomes NP-hard.

10.9 Theorem ([BBK22a, Theorem 1.2]). ADP *is NP-complete, even on acyclic digraphs.* ◁

10.2. The complexity of SFP

The objective of this section is to prove the following theorem.

10.10 Theorem. SFP is Σ_2P-complete, even on acyclic digraphs. ◁ [10.20] [10.21] [10.22] [10.23] [10.24]

SFP is contained in $\Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$ since it can be solved by a non-deterministic Turing machine that has access to an oracle for the NP-complete [GMO76] path avoiding forbidden pairs problem (see [AB09, Remark 5.16]). In the remainder of this section we reduce the Σ_2P-complete problem Σ_2SAT, see Problem 2.16, to SFP proving its Σ_2P-hardness.

10.2.1. Notation for the problem Σ_2SAT

We already saw the Σ_2SAT problem in the preliminaries. In order to work with it here we first have to introduce some notation.

10.11 Notation. The Boolean formula $\varphi \coloneqq \varphi(x, y)$ depends on n_x many x-variables $X \coloneqq \{x_1, \ldots, x_{n_x}\}$ and on n_y many y-variables $Y \coloneqq \left\{y_1, \ldots, y_{n_y}\right\}$ whose union we denote by $Z \coloneqq X \cup Y$. A *truth assignment* $T\colon Z \to \{\texttt{True}, \texttt{False}\}$ assigns a Boolean value to every variable. If we are only interested in the assignments of x- or y-variables, we write $T_X\colon X \to \{\texttt{True}, \texttt{False}\}$ as well as $T_Y\colon Y \to \{\texttt{True}, \texttt{False}\}$ and identify T with (T_X, T_Y), where $T_X \coloneqq T_{|X}$ and $T_Y \coloneqq T_{|Y}$. $\lhd$

We say that the instance φ is *satisfiable* if an x-variable assignment T_X exists such that φ evaluates to True for every y-variable assignment T_Y.

10.2.2. Outline of the $\Sigma_2\mathsf{P}$-hardness proof

To prove the hardness of SFP, we construct a directed acyclic graph D for such a Boolean formula φ. For carefully chosen $k \in \mathbb{N}$ we show that a source s and a target t in D can be separated by a set $\mathcal{A}$ of k forbidden pairs if and only if the Σ_2SAT instance φ is satisfiable.

In this digraph D, most separating pairs are predetermined. Those that are not have essentially two options, which are used to encode assignments of the x-variables. This means that an assignment T_X of the x-variables corresponds to a selection of forbidden pairs $\mathcal{A}$ and vice versa. An assignment T_Y of the y-variables will correspond to st-paths in the digraph that contain a pair from $\mathcal{A}$ if and only if the assignment $T = (T_X, T_Y)$ satisfies a clause. From this we conclude that an assignment T_X exists such that φ evaluates to True for all assignments T_Y if and only if there exists a small set $\mathcal{A}$ such that every st-path contains a pair from $\mathcal{A}$. However, the construction of the digraph also generates st-paths that do not correspond to any y-variable assignment T_Y. To make the argumentation work, we have to enforce that all these paths contain forbidden pairs.

In the following, we start with non-restrictive assumptions about the Boolean formula φ. Thereafter, we introduce the different gadgets and concepts required for the final $\Sigma_2\mathsf{P}$-hardness proof.

10.2.3. Assumptions and assignments

Without loss of generality we may assume that the Boolean formula φ is given in 3-DNF, that is, in disjunctive normal form where each clause contains exactly three literals, see [Haa19, Section 2.2.1]. Hence, we can write $\varphi = C_1 \vee \ldots \vee C_m$ as a disjunction of m clauses where each clause is the conjunction of three literals.

10.12 Assumption. The Boolean formula φ is given in 3-DNF. ◁

Let us consider a clause consisting entirely of x-variables. If it contains a variable x_i and its negation $\overline{x}_i$, the clause can never be fulfilled and we can remove it. Otherwise, we can satisfy this clause (and with it the entire formula φ) solely by an appropriate x-variable assignment. Hence, we might also assume that every clause contains at least one y-variable.

10.13 Assumption. No clause of φ consists entirely of x-variables. ◁

Our last assumption is that no variable is contained in a single clause only. This can be guaranteed, for example, by duplicating all clauses.

10.14 Assumption. Every variable is contained in at least two clauses of φ. ◁

Assumptions 10.12 and 10.13 directly imply the following lemma.

10.15 Observation. Every clause contains either one, two, or three y-variables. ◁

Before we continue, let us give an example of a formula satisfying these assumption:

$$\varphi(x,y) = (x_1 \wedge y_1 \wedge y_2) \vee (\overline{x}_1 \wedge y_1 \wedge \overline{y}_2) \vee (x_1 \wedge \overline{x}_2 \wedge \overline{y}_2) \vee (\overline{x}_2 \wedge \overline{y}_1 \wedge y_2) \vee (\overline{x}_1 \wedge \overline{y}_1 \wedge \overline{y}_2)$$

is in 3-DNF, every variable occurs at least twice, and no clause consists entirely of x-variables. It is also a yes-instance to the Σ_2SAT problem since setting x_1 to `True` and x_2 to `False` satisfies all clauses, regardless of what truth values are assigned to the y-variables.

Before we describe the graph construction in detail, we introduce local as well as global y-variable assignments and define inconsistencies.

10.16 Notation. The Boolean formula $\varphi := C_1 \vee \ldots \vee C_m$ is given as a disjunction of m clauses, where every clause $C_i := l_j^1 \wedge l_j^2 \wedge l_j^3$ is a conjunction of exactly three literals $l_j^i \in \{z, \overline{z} \colon\ z \in Z\}$. By $Y\,C$ we denote the set of y-variables that occur (negated or otherwise) in a clause C. We call an assignment of these variables a *local (y-variable) assignment* and denote it by $T_{Y\,C} \colon\ Y\,C \to \{\texttt{True}, \texttt{False}\}$. In the same spirit, we call T_Y a *global assignment.* ◁

10.17 Definition. Two local y-variable assignments $L := T_{Y\,C}$ and $L' := T_{Y\,C'}$ for distinct clauses C and C' are *consistent* if they coincide on $Y\,C \cap Y\,C'$. Otherwise, they are *inconsistent* and the (unordered) pair $I := \{L, L'\}$ is an *inconsistency.* ◁

We are now ready to start with our graph construction, which requires the introduction of several gadgets.

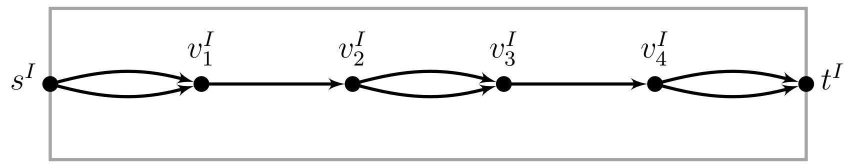


Figure 10.1.: An inconsistency gadget corresponding to an inconsistency I.

10.2.4. Graph components

Inconsistency gadgets. We start with the simplest gadget, the *inconsistency gadget.* It corresponds to inconsistencies and its only purpose is to enforce that a minimal separating set $\mathcal{A}$ contains a specific pair of arcs. We use inconsistency gadgets to ensure that paths not corresponding to a global y-variable assignment contain a forbidden pair.

Every inconsistency gadget is a directed acyclic graph as depicted in Figure 10.1. It consists of an $s^I t^I$-path with five arcs where the first, third, and last arc is replaced by two parallel arcs.

(10.22) (10.23) (10.24) **10.18 Lemma.** The unique optimal solution to separate s^I and t^I in an inconsistency gadget by forbidden pairs is $\mathcal{A}_I = \left\{\left\{v_1^I v_2^I, v_3^I v_4^I\right\}\right\}$. ◁

Proof. The set $\mathcal{A}_I$ separates s^I and t^I and every separating set needs at least one pair. Thus, every optimal solution consists of a single forbidden pair. To prove the uniqueness, suppose there is an optimal solution whose pair contains one of two parallel arcs. In this case, a path using the other arc does not completely contain this pair, which yields a contradiction. □

Variable Gadgets. The *variable gadgets* correspond to the x-variables in φ. Their purpose is to reflect a truth assignment T_X of these variables. That is, there should be exactly two optimal sets of forbidden pairs in such a gadget: one corresponding to setting the variable to `True` and one for making it `False`. An illustration of such a gadget is given in Figure 10.2. We now describe its construction in more detail. Thereafter, we explain what the two separating sets look like and prove that these are indeed the only two optimal solutions.

Essentially, the variable gadget corresponding to a variable x_i consists of two vertices s^i and t^i that are connected by several paths. Similar to the inconsistency gadgets we double some arcs on these paths and we link them in a certain way.

The gadget contains an $s^i t^i$-path for every occurrence of x_i in the formula φ. More precisely, the jth occurrence corresponds to a path $s^i v_{j,1}^i \dots v_{j,7}^i t^i$ on which we replace the first, fourth, fifth, and last arc by two parallels. Additionally, we add a path $s^i v_{0,1}^i v_{0,2}^i v_{0,3}^i v_{0,5}^i v_{0,6}^i v_{0,7}^i t^i$, which is not associated with any occurrence. On this path we replace the first, fourth, and last arc by two parallel arcs. Furthermore, we introduce the arcs $v_{0,3}^i v_{j,4}^i$ and $v_{j,4}^i v_{0,5}^i$ between these paths.

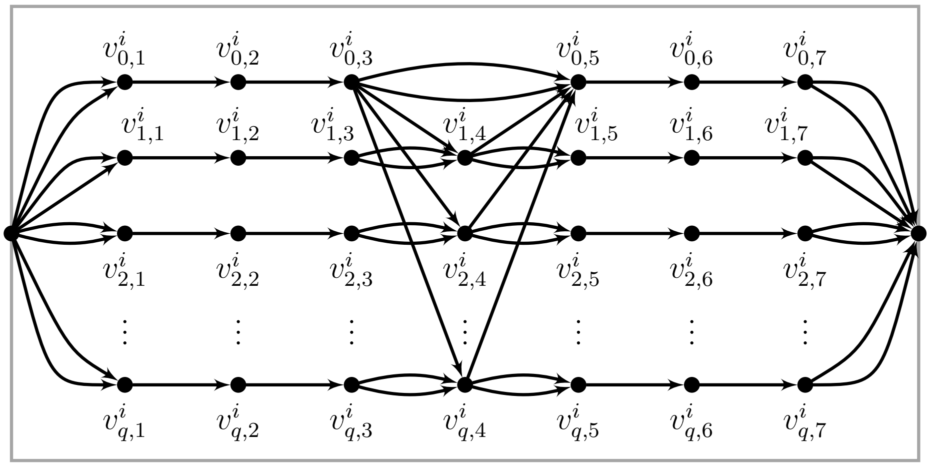

$v^i_{0,1}$ $v^i_{0,2}$ $v^i_{0,3}$ $v^i_{0,5}$ $v^i_{0,6}$ $v^i_{0,7}$
$v^i_{1,1}$ $v^i_{1,2}$ $v^i_{1,3}$ $v^i_{1,4}$ $v^i_{1,5}$ $v^i_{1,6}$ $v^i_{1,7}$
s^i
t^i
$v^i_{2,1}$ $v^i_{2,2}$ $v^i_{2,3}$ $v^i_{2,4}$ $v^i_{2,5}$ $v^i_{2,6}$ $v^i_{2,7}$
$v^i_{q,1}$ $v^i_{q,2}$ $v^i_{q,3}$ $v^i_{q,4}$ $v^i_{q,5}$ $v^i_{q,6}$ $v^i_{q,7}$

Figure 10.2.: A variable gadget corresponding to variable x_i. We use $q \coloneqq q_i$ for the number of occurrences of x_i (including negated literals) in formula φ.

Let q_i denote the number of occurrences of variable x_i in the formula φ. As there are, by construction, $q_i + 1$ arc-disjoint $s^i t^i$-paths in this gadget, an optimal set of forbidden pairs separating s^i and t^i must contain at least $q_i + 1$ pairs. Thus, the two separating sets $\mathcal{A}_i \coloneqq \left\{\left\{v^i_{j,1}v^i_{j,2}, v^i_{j,2}v^i_{j,3}\right\} : j = 0, \dots, q_i\right\}$ and $\overline{\mathcal{A}_i} \coloneqq \left\{\left\{v^i_{j,5}v^i_{j,6}, v^i_{j,6}v^i_{j,7}\right\} : j = 0, \dots, q_i\right\}$ are optimal. The following lemma shows that these are, in fact, the only two optimal sets of forbidden pairs. We identify choosing the separating set $\mathcal{A}_i$ with setting x_i to `True` and choosing $\overline{\mathcal{A}_i}$ with setting x_i to `False`.

10.19 Lemma. The sets $\mathcal{A}_i$ and $\overline{\mathcal{A}_i}$ are the only optimal sets of forbidden pairs separating s^i and t^i in the variable gadget corresponding to variable x_i. ◁ (10.22) (10.23) (10.24)

Proof. As argued above, an optimal separating set contains exactly $q_i + 1$ pairs, thus $\mathcal{A}_i$ and $\overline{\mathcal{A}_i}$ are optimal separating sets. It remains to prove their uniqueness.

Similarly to the proof of Lemma 10.18 we can show that no forbidden pair of an optimal solution uses one of two parallel arcs: otherwise, there are still $q_i + 1$ disjoint $s^i t^i$-paths, none of which completely contains this pair. With the same argumentation it follows that none of the arcs $v^i_{0,3}v^i_{j,4}$ and $v^i_{j,4}v^i_{0,5}$ between these paths is contained in a forbidden pair of an optimal solution.

Thus, all forbidden pairs are composed of arcs of the form $v^i_{j,1}v^i_{j,2}$, $v^i_{j,2}v^i_{j,3}$, $v^i_{j,5}v^i_{j,6}$, and $v^i_{j,6}v^i_{j,7}$. For $j \in \{1, \dots, q_i\}$ we consider the four different paths

$$s^i v^i_{0,1} \dots v^i_{0,7} t^i,\ \ s^i v^i_{j,1} \dots v^i_{j,7} t^i,\ \ s^i v^i_{0,1} v^i_{0,2} v^i_{0,3} v^i_{j,4} \dots v^i_{j,7} t^i, \text{ and } s^i v^i_{j,1} \dots v^i_{j,4} v^i_{0,5} v^i_{0,6} v^i_{0,7} t^i.$$

An optimal solution has to separate these four paths with only two forbidden pairs as there are $q_i - 1$ disjoint paths in the remaining gadget. This, however, is only possible if either the pairs

$$\left\{v^i_{0,1}v^i_{0,2}, v^i_{0,2}v^i_{0,3}\right\} \text{ and } \left\{v^i_{j,1}v^i_{j,2}, v^i_{j,2}v^i_{j,3}\right\}$$

or the pairs

$$\left\{v^i_{0,5}v^i_{0,6}, v^i_{0,6}v^i_{0,7}\right\} \text{ and } \left\{v^i_{j,5}v^i_{j,6}, v^i_{j,6}v^i_{j,7}\right\}$$

are chosen. Since this holds for all $j \in \{1, \dots, q_i\}$, the claim follows. □

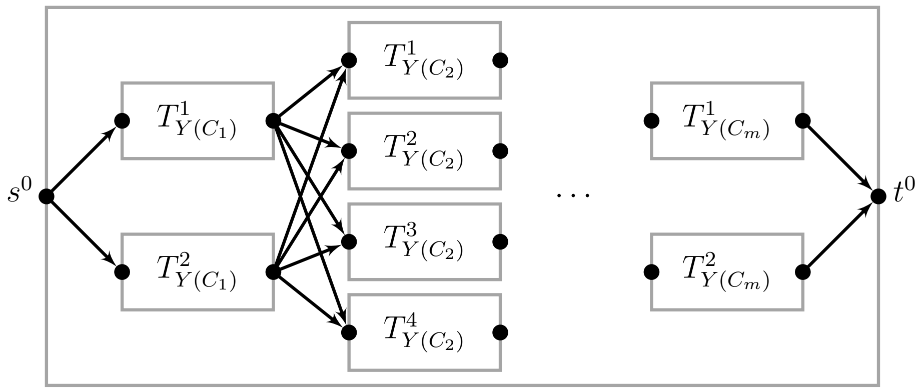


Figure 10.3.: The formula gadget for the formula $\varphi \coloneqq C_1 \vee C_2 \vee \ldots \vee C_m$ in 3-DNF. To distinguish the different possible local assignments of a clause C we enumerate them $T^1_{Y\,C}$, $T^2_{Y\,C}$, and so on.

Formula Gadget. The *formula gadget* consists of clause assignment units, which we describe later, that are ordered in a layered structure. For now it suffices to know that they have one input vertex and one output vertex, which we use to connect them. By Observation 10.15, every clause C of φ, contains $l \in \{1, 2, 3\}$ many y-variables. For every of the 2^l possible local y-variable assignments $L \coloneqq T_{Y\,C}$ for C we introduce one such clause assignment unit. We denote its input vertex by s^L and its output vertex by t^L. This yields either two, four, or eight clause assignment units for each clause.

The jth layer of the formula gadget consists of all clause assignment units corresponding to the jth clause of φ. A source s^0 is connected to the input s^L of every clause assignment unit corresponding to a local assignment $L \coloneqq T_{Y\,C_1}$ of the first clause. In addition, we connect the clause assignment units of consecutive clauses in the formula gadget by complete bipartite graphs. Finally, we connect every output t^L of a unit corresponding to the last clause C_m with the target t^0. This construction is visualised in Figure 10.3.

Most clause assignment units provide paths from their input to their output vertex. Therefore, s^0t^0-paths through the formula gadget pass through exactly one clause assignment unit of every layer. This way, every such path selects a local y-variable assignment for every clause. If these are consistent, that is, if every y-variable is assigned the same truth value in each local assignment of a clause that contains it, they can be combined to a global y-variable assignment. Conversely, we can also associate an assignment T_Y with an s^0t^0-path which uses, in every layer, the clause assignment unit corresponding to $T_{Y\,C} = T_{Y|_{Y\,C}}$ for the respective clause C.

Thus, the paths through the formula gadget are linked with the global y-variable assignments. Our goal is to ensure that any such path contains a forbidden pair if and only if the associated assignment satisfies the formula φ (in conjunction with the x-variable assignment). For this, the variable gadgets will play an important role. However, we also have to take those paths into consideration that do not correspond to consistent y-variable assignments. In order to ensure that these paths contain forbidden pairs as well, we will make use of the inconsistency gadgets.

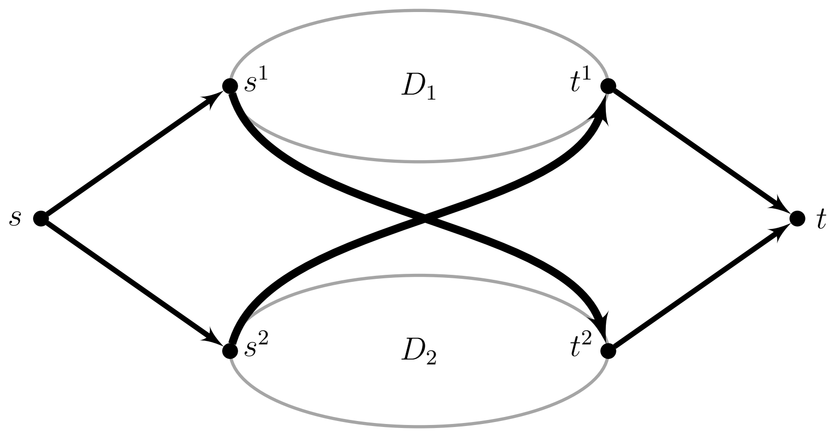


Figure 10.4.: An exemplary typification construction. The bold arcs ss^1, ss^2, t^1t, and t^2t represent bunches of p parallel arcs. The even thicker arcs s^1t^2 and s^2t^1 represent bunches of $p+1$ parallel arcs.

Typification. All the gadgets introduced until now need to be part of st-paths in the final graph. Our construction so far has only looked at paths living entirely in one of these gadgets. Since it will be possible to travel between gadgets later, new paths arise. In particular, we obtain 'mixed' paths that start at the source of one gadget and end at the terminal of another. In order to keep these mixed paths in check when we finally put these pieces together we need the concept of *typification*.

To explain the idea of typification, we start with a small example. Let two disjoint digraphs D_1 and D_2 be given. In each digraph D_i we want to separate a source s^i and a target t^i by forbidden pairs. However, we want to combine these two digraphs to a single digraph D without affecting the optimal choice of forbidden pairs, that is, we still only want to select pairs in D_1 and D_2. Simply adding a source s, a target t, and connecting these with arcs ss^1, ss^2, t^1t, and t^2t does not suffice as the combined instance can always be separated by the two forbidden pairs $\{ss^1, t^1t\}$ and $\{ss^2, t^2t\}$. But if we know that $p-1$ pairs are sufficient to separate D_i for $i \in \{1,2\}$, we can replace each of the four additional arcs by a bunch of p parallel arcs. In other words: if k_i pairs are sufficient to separate D_i, we can choose any $p > \max\{k_1, k_2\}$. Therefore, an optimal solution in the combined instance only uses arcs that are contained within the subgraphs D_1 and D_2.

If we also add $p+1$ parallel arcs from s^1 to t^2 as well as from s^2 to t^1, we have to choose forbidden pairs separating all paths ss^1t^2t and all paths ss^2t^1t. These paths only consist of additional arcs not contained in the original digraphs D_1 and D_2. The unique optimal solution to separate these paths is to choose the $2p^2$ forbidden pairs that combine an arc ss^1 with an arc t^2t and an arc ss^2 with an arc t^1t. Thus, we can separate D_1 by k_1 forbidden pairs and D_2 by k_2 forbidden pairs if and only if we can separate D by $2p^2 + k_1 + k_2$ forbidden pairs. This situation is visualised in Figure 10.4.

The reason we introduce these additional arcs is because they help us weed out mixed paths: if we allow arcs between D_1 and D_2 in D, then it becomes possible to obtain st-paths containing s^i and t^j for $i \neq j$. By adding the additional 'diagonal' arcs s^1t^2

and s^2t^1 we enforce the choice of all $2p^2$ pairs $\{ss^i, t^jt\}$ for $i \neq j$ and, thus, ensure that these mixed paths are already saturated with at least one pair. This just leaves paths that start with ss^i and end with t^it. We only have to examine whether all paths of these two *types* contain a forbidden pair or not. Note that this does include paths that are not solely part of a subgraph D_i, as they can leave and return, but it does reduce the potential paths without a forbidden pair immensely.

This construction can be generalised to encompass more than only two types. For q subgraphs $D_1, \dots, D_q$ with sources s^i and targets t^i, $i \in \{1, \dots, q\}$, we can add p parallel arcs from s to every source s^i and from every target t^i to t. Additionally, we add $p+1$ parallel arcs s^it^j for all i, $j \in \{1, \dots, q\}$ with $i \neq j$. Every optimal solution has to use the p^2 forbidden pairs of arcs $\{ss^i, t^jt\}$ of different types $i \neq j$. Thus, every optimal solution has $p^2q(q-1)$ forbidden pairs and, additionally, the pairs required to separate all paths of the q different types (all paths using ss^i and t^it for some i). We intend to use this to give all inconsistency gadgets, all variable gadgets, as well as the formula gadget their own type.

Clause Assignment Units and Graph Construction. To construct the digraph corresponding to formula φ we use the formula gadget, a variable gadget for every x-variable, and several inconsistency gadgets. More precisely, for every pair of clause assignment units (within the formula gadget) that corresponds to incompatible assignments we introduce one such inconsistency gadget. All these gadgets are combined into the digraph D as explained in the typification section.

Let us describe the graph construction in detail. That is, we finally have to specify what the clause assignment units look like and how these are connected to the other gadgets. Recall that the formula gadget contains a clause assignment unit for every clause C and every possible assignment $T_{Y\,C}$ of Boolean values to the y-variables contained in C. As already stated in the formula gadget section, these are 2^l clause assignment units for a clause with l many y-variables.

A *clause assignment unit* corresponding to a y-variable assignment L of a clause C contains exactly three vertices: s^L, v^L, and t^L. Note that we use s^L and t^L in order to connect the clause assignment units in the formula gadget as described above. The vertices v^L and t^L are connected by an arc v^Lt^L if and only if C contains at least one y-literal that evaluates to `False` with the y-variable assignment L. This arc is missing in exactly one clause assignment unit corresponding to a clause C as there is only exactly one assignment $T_{Y\,C}$ that satisfies all y-literals in C.

These are all components within a clause assignment unit. In particular, the clause assignment units are not connected and, thus, neither is the formula gadget. The following modifications only add some arcs between different gadgets. These are illustrated by dashed arcs in Figures 10.5 to 10.7.

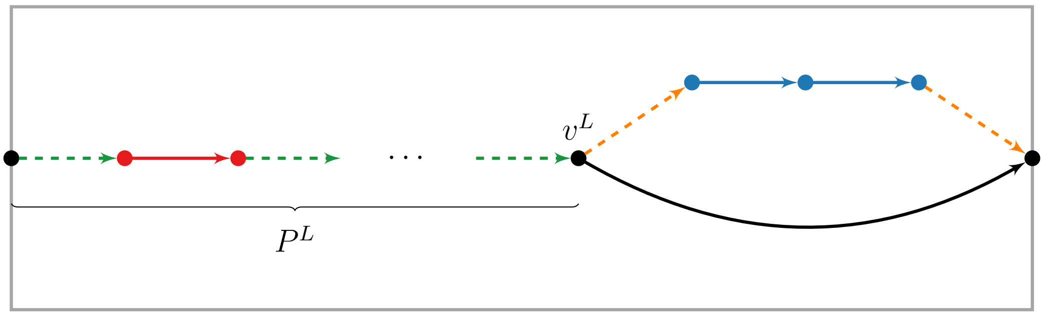


Figure 10.5.: A clause assignment unit. It corresponds to a local y-variable assignment L for clause C containing a single x-variable. The assignment L does not fulfil all y-literals of C as the arc $v^L t^L$ is present. The coloured, solid arcs are contained in other gadgets and the dashed arcs connect these, see also Figures 10.6 and 10.7.

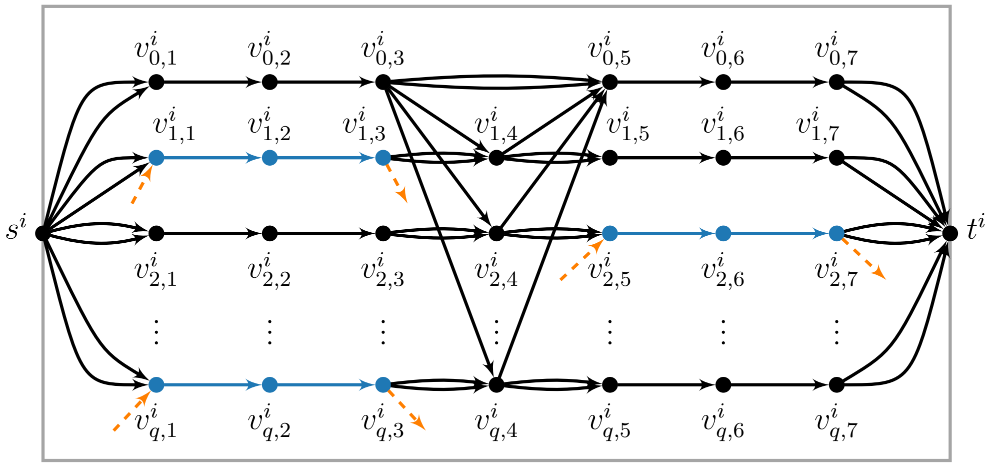


Figure 10.6.: A variable gadget with the connections to clause assignment units. New, in comparison to Figure 10.2, are the blue and the dashed orange arcs which correspond to those from Figure 10.5. In this example, the first and last occurrence of the corresponding x-variable occurs non-negated and the second occurrence is negated.

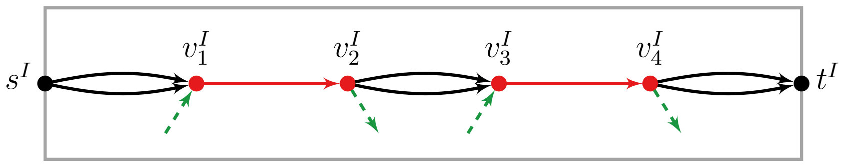


Figure 10.7.: An inconsistency gadget with its connections to clause assignment units or other inconsistency gadgets. The red and the dashed green arcs are newly introduced in comparison to Figure 10.1 and correspond to those from Figure 10.5.

In addition to the (potentially non-existing) arc $v^L t^L$, we add another path from v^L to t^L for every x-literal contained in C. The path of a literal corresponding to variable x_i passes through the variable gadget of x_i. If the occurrence of x_i in C is the jth occurrence in φ in total, this path uses either the arcs $v^i_{j,1}v^i_{j,2}$ and $v^i_{j,2}v^i_{j,3}$ (if C contains the literal x_i) or the arcs $v^i_{j,5}v^i_{j,6}$ and $v^i_{j,6}v^i_{j,7}$ (if C contains the literal $\overline{x}_i$). In the former case we add the inter-gadget arcs $v^L v^i_{j,1}$ and $v^i_{j,3}t^L$ and in the latter case we add $v^L v^i_{j,5}$ as well as $v^i_{j,7}t^L$. These connecting arcs are indicated by dashed orange arcs in Figures 10.5 and 10.6.

Additionally, we introduce arcs between different gadgets that provide paths from s^L to v^L. For this, we introduced one inconsistency gadget for every inconsistency which connect them as follows. A clause assignment unit corresponding to an assignment $L \coloneqq T_{Y\,C_i}$ of clause C_i gets an $s^L v^L$-path P^L that passes through every inconsistency gadget for an inconsistency $I = \left\{L, T_{Y\,C_j}\right\}$ containing L. In such a gadget, this path uses either the arc $v^I_1 v^I_2$ (if $j > i$) or the arc $v^I_3 v^I_4$ (if $j < i$). The path P^L collects all these arcs in arbitrary order by introducing further arcs between them. The connecting arcs are indicated by dashed green arcs in Figures 10.5 and 10.7.

Magnitudes and Parameters. Recall that we denote the number of clauses of the formula $\varphi \coloneqq C_1 \vee \ldots \vee C_m$ by m and the number of x- and y-variables by n_x and n_y, respectively (compare Notations 10.11 and 10.16). Also as before, let q_i denote the number of occurrences of the ith x-variable x_i in the formula φ. Additionally, we denote the number of inconsistencies by n_I.

The digraph of the corresponding SFP instance consists of one formula gadget, n_x variable gadgets, and n_I inconsistency gadgets. The formula gadget consists of at most eight clause assignment units per clause. Hence, we have $\mathcal{O}(m)$ clause assignment units. Moreover, since we have at most one inconsistency gadget for every pair of clause assignment units, we have $n_I \in \mathcal{O}(m^2)$.

As shown in Lemmas 10.18 and 10.19 we need one forbidden pair to separate s^I and t^I in the inconsistency gadget for I and $q_i + 1$ forbidden pairs to separate s^i and t^i in the variable gadget for x_i. Since the formula gadget itself is not connected, we do not need additional forbidden pairs for it. Thus, for the typification framework, we choose

$$p = \max_{i=1,\ldots,n_x} q_i + 2 \tag{10.1}$$

and add p parallel arcs from a source s to all input vertices of variable and inconsistency gadgets as well as to the formula gadget. That is, we add all parallels of the form ss^i, ss^I, and ss^0. Analogously, we add p parallel arcs from every such output vertex to a target t resulting in parallels $t^i t$, $t^I t$, and $t^0 t$. Furthermore, we add $p+1$ parallel arcs from every input vertex of such a gadget to the output vertices of all other gadgets. In total, we introduce

$$2p(n_x + n_I + 1) + (p+1)(n_x + n_I + 1)(n_x + n_I) \in \mathcal{O}\left(m^5\right)$$

arcs for the typification, where the claimed asymptotic complexity $\mathcal{O}(m^5)$ follows since $n_I \in \mathcal{O}(m^2)$ and since both n_x and p are bounded by the number $3m$ of literals in φ. As explained in the typification section we need

$$k_0 = p^2(n_x + n_I + 1)(n_x + n_I) \tag{10.2}$$

pairs to separate s and t in the graph that only consists of arcs introduced for typification. We show in the analysis subsection below that we can separate the digraph D by

$$k = k_0 + n_I + \sum_{i=1}^{n_x}(q_i + 1) \tag{10.3}$$

forbidden pairs if and only if the Boolean formula φ has an x-variable assignment such that φ evaluates to `True` for every y-variable assignment.

10.2.5. Analysis

So far, given a Boolean formula φ, we have constructed an SFP instance D with source s and target t and specified the number k of forbidden pairs. In the following, we use this to give a proof for Theorem 10.10, divided into Lemmas 10.20 to 10.24.

10.20 Lemma. The digraph D that is constructed as described above is acyclic. $\triangleleft$ (10.10)

Proof. As a digraph is acyclic if and only if it exists a topological ordering, we prove the claim by specifying such a topological ordering for D. However, we do not explicitly map every vertex to a natural number. Instead, we describe a procedure how to obtain the order of the vertices. The reason is that we have to insert some vertices in between others multiple times. This would make a formal definition of this mapping quite technical.

In a first step, we enumerate all vertices in the formula gadget together with the interior vertices from inconsistency gadgets. Here, the 'interior vertices' of an inconsistency gadget for an inconsistency I are the vertices $v_1^I, \dots, v_4^I$. Note that each arc of the form $v_1^I v_2^I$ and $v_3^I v_4^I$ is contained in an $s^L v^L$-path P^L of some clause assignment unit. We start to enumerate the vertices in clause assignment units corresponding to the first clause C_1. There, we first enumerate the paths P^L of assignments L for C_1 followed by the output vertices t^L of the corresponding gadgets. Afterwards, we proceed in the same way with the subsequent clauses. This procedure is visualised in Figure 10.8.

By enumerating the formula gadget this way, for $i < j$, every vertex corresponding to a clause C_i gets a lower number than every vertex corresponding to clause C_j. This holds, in particular, for the vertices on the $s^L v^L$-paths P^L in the clause assignment units. For every inconsistency $I \coloneqq \{L, L'\}$ with $L \coloneqq T_{Y\,C_i}$, $L' \coloneqq T_{Y\,C_j}$ and $i < j$, the path P^L uses the arc $v_1^I v_2^I$ and the path $P^{L'}$ uses the arc $v_3^I v_4^I$ within the corresponding inconsistency gadget. Thus, the partial topological ordering defined up to this point is not only consistent with all arcs of the formula gadget and the arcs in between formula and inconsistency gadgets, but also within all these inconsistency gadgets.

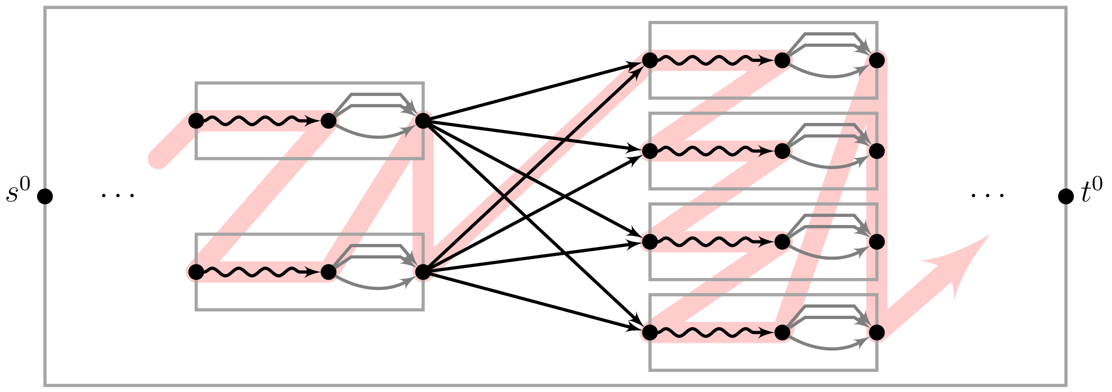


Figure 10.8.: A topological ordering of the formula gadget. Here, a schematic representation of how to enumerate the vertices in a formula gadget for a topological ordering is shown. The $s^L v^L$-paths within the clause assignment units are drawn as wavy lines. The paths and lines that might connect v^L and t^L are only indicated. This visualises the first step in the proof of Lemma 10.20.

It remains to be proved that we can extend this partial ordering to the variable gadgets and the missing in- and output vertices. The latter are, however, no problem as we can put all input vertices at the beginning and all output vertices at the end, directly after s or before t (except for the in- and outputs of clause assignment units that already are assigned a number in the first step).

Thus, in a second step, we have to assign numbers to the vertices of the variable gadgets. Such a variable gadget corresponding to a variable x_i consists of $q_i + 1$ many $s^i t^i$-paths. With the exception of the additional path, every path corresponds to one occurrence of this variable. Let us consider the jth occurrence and let C be the corresponding clause. Depending on whether x_i occurs negated or not, we have restrictions either for the values of $v^i_{j,5}$ and $v^i_{j,7}$ or for the values of $v^i_{j,1}$ and $v^i_{j,3}$, respectively (as those have arcs to vertices in clause assignment units that are already assigned a number). In particular, we only have restrictions on the 'left half' or on the 'right half' of the path but not on both. In the case the jth occurrence is not negated, we assign the vertices $v^i_{j,1}, \ldots, v^i_{j,3}$ increasing values that we insert in between the highest number of a vertex v^L and the lowest number of a vertex t^L in the topological ordering for every assignment L of clause C. Note that we have enumerated these vertices in the first phase, such that the highest number of a vertex v^L is in fact smaller than the lowest number of a vertex t^L for all assignments L of clause C.

As all paths in the variable gadget are only connected to the additional path

$$s^i v^i_{0,1} v^i_{0,2} v^i_{0,3} v^i_{0,5} v^i_{0,6} v^i_{0,7} t^i,$$

we can extend the partial topological ordering within every variable gadget. Therefore, we can assign $v^i_{0,1}$, $v^i_{0,2}$, and $v^i_{0,3}$ values that are smaller and $v^i_{0,5}$, $v^i_{0,6}$, and $v^i_{0,7}$ values that are larger than any values of vertices within the variable gadget. Thereafter, we can insert the 'missing half' of paths accordingly. □

10.21 Lemma. The digraph D corresponding to the formula φ is of polynomial size and it can be constructed in polynomial time, both with respect to the size of φ. $\triangleleft$ (10.10)

Proof. For a given instance $\varphi = C_1 \vee \ldots \vee C_m$ with n_x many x-variables, let D be the digraph as described in this section. Its size is polynomial in the size of φ as it contains $\mathcal{O}(m)$ clause assignment units, n_x variable gadgets, and $\mathcal{O}(m^2)$ inconsistency gadgets. The size of the clause assignment and inconsistency gadgets is constant and the size of a variable gadget is linear in the number of occurrences of the corresponding x-variable. We add $\mathcal{O}(m^5)$ arcs for the typification and to see that also only polynomially many arcs connect different gadgets we can associate these to at least one of the two corresponding gadgets. Every inconsistency gadget is connected by exactly four inter-gadget arcs and every clause assignment unit is connected by either two, four, or six arcs. All the arcs connecting a variable gadget to other gadgets have their other endpoint in a clause assignment unit and are thus already considered. Hence, the number of inter-gadget arcs is polynomially bounded. Moreover, we can also construct the digraph D from the formula φ in polynomial time. $\square$

10.22 Lemma. At least k forbidden pairs are required to separate s and t in D, where k is defined as in Equation (10.3) on Page 165. $\triangleleft$ [10.18] [10.19] (10.10) (10.24)

Proof. This follows from the typification construction. Every set of forbidden pairs $\mathcal{A}$ has to contain

- the k_0 pairs to separate the graph that consist only of typification arcs,
- the n_I pairs to separate all inconsistency gadgets (see Lemma 10.18), and
- for every x_i either $\mathcal{A}_i$ or $\overline{\mathcal{A}_i}$ (see Lemma 10.19). $\square$

10.23 Lemma. If the Σ_2SAT instance φ is satisfiable, then we can separate s and t in D by k forbidden pairs, where k is defined as in Equation (10.3) on Page 165. $\triangleleft$ [10.18] [10.19] (10.10)

Proof. If φ is satisfiable, then there is an x-variable assignment T_X such that φ evaluates to `True` no matter which values are assigned to the y-variables. We define a set $\mathcal{A}$ of forbidden pairs depending on T_X as follows.

First, $\mathcal{A}$ contains the k_0 forbidden pairs that separate the graph only consisting of typification arcs. Secondly, it contains the n_I pairs that separate all inconsistency gadgets, compare Lemma 10.18. And finally, we choose a separating set for every x-variable x_i. If $T_X(x_i) =$ `True`, we use the separating set $\mathcal{A}_i$. Otherwise, we use $\overline{\mathcal{A}_i}$. See Lemma 10.19 for more information on these two sets.

By Equation (10.3) and Lemmas 10.18 and 10.19 we have chosen k forbidden pairs. Moreover, by the typification construction, all paths that do not use any gadget and those whose first and last gadgets are not the same contain a forbidden pair.

It remains to prove that every path that enters a gadget via a direct arc from s and leaves this gadget via a direct arc to t completely contains at least one forbidden pair. We consider the different gadgets.

First, consider an inconsistency I and the corresponding inconsistency gadget D_I. Every path entering D_I via ss^I must also use the arc $v_1^I v_2^I$. Analogously, every path leaving D_I via $t^I t$ must also use the arc $v_3^I v_4^I$. Thus, every path entering and leaving this gadget via input s^I and output t^I contains the forbidden pair $\left\{v_1^I v_2^I, v_3^I v_4^I\right\}$ that we have chosen.

Next, consider the variable gadget D_i for a variable x_i. Similarly to the inconsistency gadget, a path entering D_i via s^i can leave D_i at some vertex $v_{j,3}^i$ at the earliest and, thus, such a path has to contain the pair $\left\{v_{j,1}^i v_{j,2}^i, v_{j,2}^i v_{j,3}^i\right\}$. Analogously, a path leaving D_i via t^i must enter D_i at some vertex $v_{j',5}^i$ at the latest and, thus, this path has to contain the pair $\left\{v_{j',5}^i v_{j',6}^i, v_{j',6}^i v_{j',7}^i\right\}$. At least one of these two pairs is contained in the set of forbidden pairs we have chosen.

Finally, let us consider the formula gadget and let P be a path that enters the gadget via s^0 and leaves it finally via t^0. The path P passes through multiple inconsistency gadgets. If it uses more than one arc from one of them, it directly contains the forbidden pair chosen in this inconsistency gadget. Thus, we can assume that P uses at most one arc from every inconsistency gadget.

If the path P leaves some clause assignment unit for a clause C via an arc to a variable gadget, it has to leave this variable gadget via an arc to the vertex t^L of a clause assignment unit that also corresponds to clause C. Thus, for every clause, this path enters exactly one clause assignment unit via its input s^L and uses the $s^L v^L$-path P^L therein. Consequently, if P passes through clause assignment units corresponding to inconsistent assignments L and L', it contains the forbidden pair contained in the inconsistency gadget for $I = \{L, L'\}$.

Therefore, we can assume that P enters only clause assignment units corresponding to consistent assignments. This allows us to define a global y-variable assignment T_Y by combining the local clause assignments related to the clause assignment units that P enters via s^L. As φ is satisfiable and T_X is chosen appropriately, we have that φ evaluates to `True` with $T = (T_X, T_Y)$. In particular, there is at least one clause C that is fulfilled. Let us consider the clause assignment unit associated with C that P enters via s^L. Since this clause is fulfilled, all y-literals are `True` and, thus, the arc $v^L t^L$ is not present. Hence, the path P has to pass through an x-variable gadget. However, as this x-literal in C is also `True`, by the construction of the digraph and the choice of the forbidden pairs, P has to use a forbidden pair in this variable gadget.

In all possible cases, the path P contains a forbidden pair. Thus, s and t can be separated in D by k forbidden pairs. □

[10.18] [10.19] [10.22] **10.24 Lemma.** If the Σ_2SAT instance φ is not satisfiable, we cannot separate s and t in D by k forbidden pairs, where k is defined as in Equation (10.3) on Page 165. ◁

(10.10) *Proof.* Suppose for the sake of a contradiction that we can separate s and t in D by k forbidden pairs.

By Lemma 10.22 we need at least k forbidden pairs. By the typification construction and by Lemmas 10.18 and 10.19 we have to choose the forbidden pairs from $\mathcal{A}_i$ or $\overline{\mathcal{A}_i}$ in a variable gadget corresponding to variable x_i.

We define an x-variable assignment T_X based on this set of forbidden pairs. A variable x_i is set to `True` if we have chosen $\mathcal{A}_i$ to separate its variable gadget. Otherwise, if we have chosen $\overline{\mathcal{A}_i}$, we set x_i to `False`.

As φ is not satisfiable, there exists a y-variable assignment T_Y such that φ evaluates to `False` with $T = (T_X, T_Y)$. This y-variable assignment T_Y corresponds to exactly one clause assignment unit $T_{Y\,C} = T_{Y|_{Y\,C}}$ for every clause C. We now construct an st-path in D that does not contain a forbidden pair: this path starts with the arc ss^0 and ends with t^0t. For every clause C it passes through the clause assignment unit corresponding to $L = T_{Y|_{Y\,C}}$ where it first uses the s^Lv^L-path P^L. If the clause contains a y-literal that is `False`, the path continues along the arc v^Lt^L that is present in this case. Otherwise, there is an x-literal that is not fulfilled. In this case, there exist a v^Lt^L-path through the corresponding variable gadget using two arcs that are not chosen as a forbidden pair (as this literal is `False`).

The path constructed this way does not contain a forbidden pair from a variable gadget. It does not contain a forbidden pair from an inconsistency gadget either as it uses at most one arc from every inconsistency gadget. This is the case because it only uses consistent assignments for the clauses. And since the path does not contain a forbidden pair used to separate the graph consisting of only typification arcs, the path does not contain a forbidden pair at all. This contradicts our initial assumption and the proof is complete. □

10.2.6. A note on parallels

Since we are usually primarily concerned with simple graphs, we take a look at the necessity of parallels in the reduction we just presented. Parallels appeared in the inconsistency and variable gadgets as well as in the typification construction. For those occurring in the gadgets, this was purely a cosmetic choice. Subdividing these arcs removes the parallels and still ensures that neither of the ‘parallel replacement arcs’ are used in a forbidden pair.

However, the typification construction breaks if we simply subdivide the parallels. To see this, recall the example from Figure 10.4 where we needed $2p^2$ pairs, together with those required to separate D_1 and D_2. If we now subdivide all parallels, we suddenly have access to the pairs consisting of both arcs on these newly created length 2 paths. Using these, we can separate s and t using just $2p$ pairs, none of which touch the graphs D_i at all.

This issue can be remedied by the following construction: the $p + 1$ parallels from s^i to t^j for $i \neq j$ are replaced by $M := q^2p^2$ many and subdivided, where q is the number

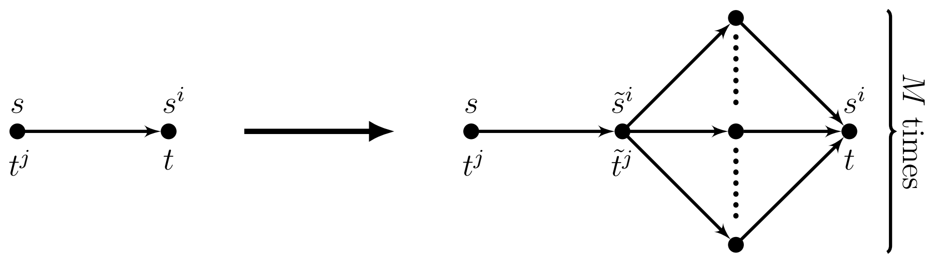


Figure 10.9.: Replacing the parallels in the typification construction. Each arc in a bundle of parallels out of s or into t is replaced by an arc that is followed by M parallels, each of which is then subdivided.

of graph types in use, as in the typification paragraph. The p parallels from s to s^i or from t^i to t are replaced as illustrated in Figure 10.9.

We assume that $p \geq 2$ (we set it to 2 otherwise) and claim that, with this construction, it is again optimal to pick $q(q-1)p^2$ pairs, together with those needed to separate the D_i. That this many suffice can be seen as before, except that this time we choose the pairs $\left\{s\tilde{s}^i, t^j\tilde{t}^j\right\}$ instead of $\{ss^i, t^jt\}$.

We now just need to verify that fewer pairs are not possible. To this end, let $\mathcal{A}$ be a minimal set of pairs. We show that $\mathcal{A}$ contains no pair with an arc in a length 2 path from s^i to t^j: to see that this holds, suppose $\mathcal{A}$ has a pair $\{a, a'\}$ where a is on a length 2 path. Then there exists an st-path P that only contains this pair by the minimality of $\mathcal{A}$. In P, we can replace the length 2 path containing a by any of the $M-1$ alternatives, at least one of which does not contain an arc in a pair of $\mathcal{A}$ since $|\mathcal{A}| \leq q(q-1)p^2 < M$. Hence, this replacement avoids all pairs in $\mathcal{A}$, a contradiction.

In the same fashion, we conclude that no pair contains an arc on the paths from $\tilde{s}^i$ to s^i or from $\tilde{t}^j$ to t. Hence, we are only left with the pairs presented above that can possibly separate the st-paths of length 8 that avoid the graphs D_i, making this choice optimal.

Chapter 11.

Conclusion and Future Research

Many combinatorial proofs contain a significant number of case distinctions, a statement which is also true for several proofs in this thesis. While these yield results, they are often hard to read, especially if little work is put into structuring and illustrating them. As a result, such proofs generally do not scale well if they are done for a special case, even if a majority of the arguments carry over to a more general setting. In our study of the 3-decomposition conjecture, we encountered two such proofs and put emphasis on making them more scalable.

The first one is our proof that star-like cubic graphs satisfy the 3-decomposition conjecture. It is true that the proof we presented is not the shortest way to obtain the result, but shortening the proof comes at the cost of readability and several insights are lost in the process. Instead, we structured it in a way that first describes decompositions that can be obtained in the restriction to a cycle, where the star-like structure plays no role yet. After investing the time to properly define these decompositions and prove that they exist, we easily derive the result for star-like graphs. However, we now also know decompositions we can use in a more general setting, where the cycles are not nicely arranged in a star. We believe that tree-like graphs are within reach by employing similar methods.

The second example of a proof requiring many case distinctions and which does not scale well is our proof that the 3-decomposition conjecture holds for path-width 4. First, we proved this in the usual way, by looking at all the cases, and then determined the steps responsible for most of the manual labour required. These are: the finding of local structures that behave nicely with respect to 3-decompositions and the checking that small path-width makes one of these appear. We show how both steps can be largely automated, making them feasible for larger graphs and path-width values. With the algorithmic support provided, it should be possible to prove the result for path-width 5, though it seems to require either a non-negligible number of larger graphs that can easily be seen to behave well or a clever argument to show that some smaller subgraph is compatible with 3-decompositions.

We note that the procedure we use to find compatible local structures is specific to the 3-decomposition conjecture. However, our algorithm for determining whether small path-width implies the existence of certain subgraphs is more general and can theoretically be applied to many conjectures. It does benefit from somewhat restrictive graph classes though, such as cubic or, more generally, regular graphs.

Aside from this main narrative, we also looked at two connectivity problems. We proved a tight lower bound on the number of bits needed to locally certify the standard *st*-reachability problem on digraphs for (a concept slightly stronger than) proof labelling schemes. While the undirected case is easy and known, the directed case was missing a lower bound, even in a very restricted setting. Ideally, one would want a lower bound for the stronger concept of locally checkable proofs, though a different construction appears to be needed for this.

We also studied the separating by forbidden pairs problem, which is the dual to a very natural generalisation of the disjoint paths problem in which paths are allowed to share up to one arc, pairwise. This almost disjoint paths problem has garnered little to no attention in the literature, despite the fact that disjoint paths are well-studied. We obtained first complexity results here, but a further investigation of the almost disjoint paths problem and its duality is an interesting new research direction.

Appendix A.

Straightforward Extensions of 3-Decompositions

Here we go over the straightforward local behaviours of the extensions to the $K_{2,3}$, the claw-square, the domino, the Pet$^-$, and the twin-house that we omitted in Chapter 5. We specify these in the same way we did for the triangle in Figure 5.5, by determining the possible local behaviours at the smaller graph and showing how to extend them. We also reduce the number of local behaviours required for the square by employing switching arguments.

The $\mathbf{K_{2,3}}$. Recall the extension shown in Figure 5.6a. We determined the local behaviours of the single vertex in Figure 5.4 and, because the $K_{2,3}$ is symmetric, we only need to check these three cases. For each, Figure A.1 shows how to extend these to the $K_{2,3}$.

The Pet$^-$. For the extension from Figure 5.7a we need the same three behaviours as above, since the Pet$^-$ is also symmetric. The second one is covered in Lemma 5.14, and the remaining two are shown in Figure A.2.

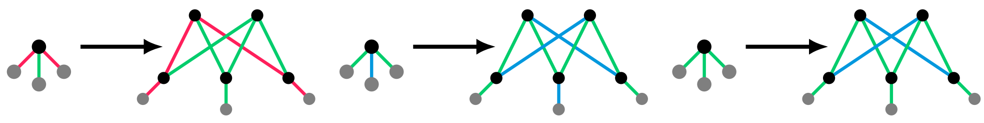


Figure A.1.: Extending a 3-decomposition from a vertex to a $K_{2,3}$. (5.12)

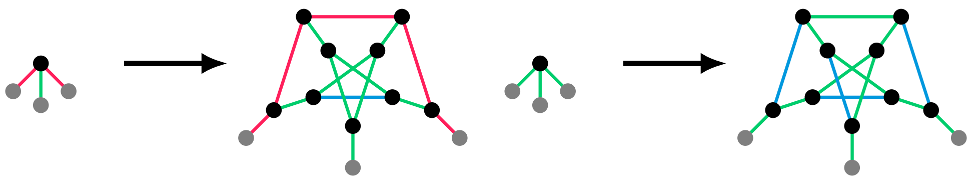


Figure A.2.: Extending a 3-decomposition from a vertex to a Pet$^-$. (5.14)

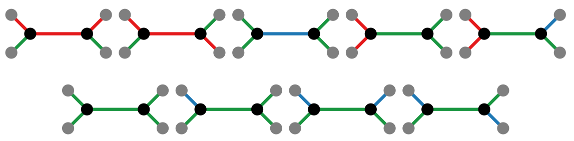
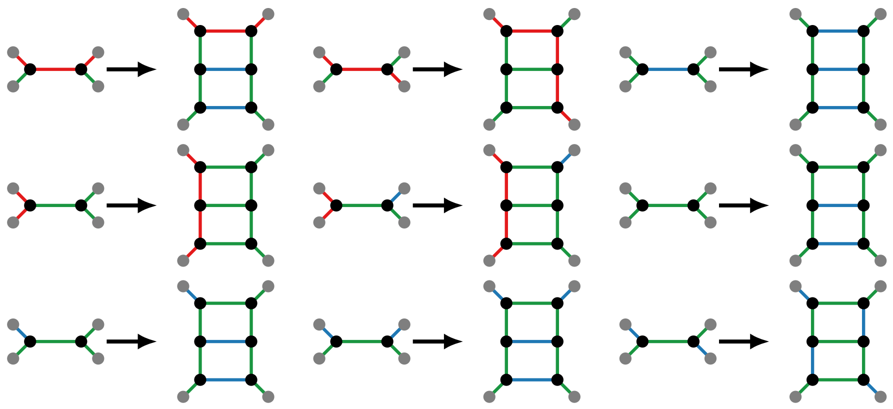


Figure A.3.: Behaviour of a 3-decomposition at an edge (without reflections).


(5.12) Figure A.4.: Extending a 3-decomposition from an edge to a domino.

The domino, part I. Next, we look at the extension in Figure 5.6c. For this, we first need to look at the possible behaviours of a 3-decomposition of G at the edge u_1u_2 we are extending. If the edge is in C, then one further edge at each end must also be in C with the remaining two being part of T. This yields, up to reflectional symmetry at the horizontal and vertical axis, the first two cases in Figure A.3. Note that both the edge and the domino have these two symmetries, such that it suffices to consider one of the symmetrical cases.

Next, assume that u_1u_2 is in M, then all remaining edges are in T, giving us the third case. For all other cases, we have $u_1u_2 \in T$ and we need to distinguish the behaviour of the remaining edges. Should one of them be in C, say one at u_1, then both at this vertex are and at least one at u_2 is in T. This gives us the fourth and fifth case.

We may now assume no edges are in C and only need to consider the number of M-edges left. If there are none, all edges must be in T. A single M-edge just yields one case up to symmetries and two give us two more. More than two are not possible.

In total, there are nine types of behaviours to check, all of which give rise to a straightforward extension to the domino, as shown in Figure A.4.

Local behaviours of the square. All remaining extensions start from the square, which is why we first check which local behaviours 3-decompositions can exhibit. Afterwards, we use switching arguments to eliminate behaviours that need not be considered. Let $u_1u_2u_4u_3u_1$ be a square (as seen in Figure 5.10) in a graph G with 3-decomposition (T, C, M). The possibilities obtained initially are shown in Figure A.5, up to rotational symmetry.

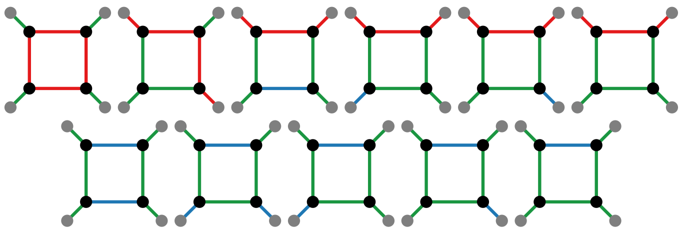


Figure A.5.: Behaviours of a 3-decomposition at a square (without rotations).

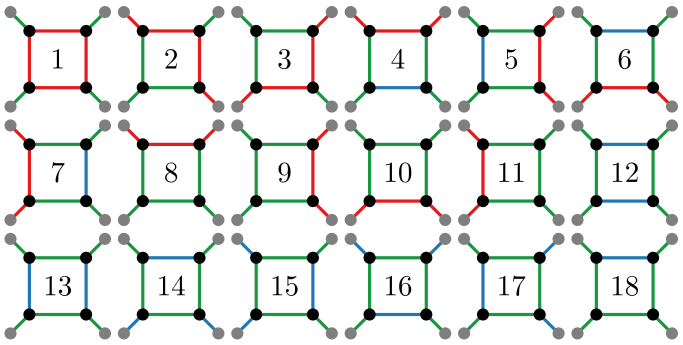


Figure A.6.: The local behaviours of the square that suffice to be checked. These are all behaviours shown in Figure A.5, including the rotations, but excluding ones removed by the transformations shown in Figure A.7.

We first distinguish cases based on the number of edges of C in the square. If there are four such edges, then the remaining edges with an end in the square must be part of T. It is not possible for exactly three edges to be in C as this isolates an edge. In the case of two edges, they form a path between two non-adjacent vertices and all other edges are in T, yielding just one case with four rotations. For a single edge, we need each of its ends to be incident to another edge in C that leaves the square. Thus, the last remaining edge at the ends is part of T. The final edge of the square can then be in M, resulting in two more T-edges, or in T, resulting in at most one M-edge. This gives the next four cases, each of which has four rotations.

This just leaves the case without any C-edges in the square. We now differentiate by the number of M-edges, where zero, three, and four are impossible. Two M-edges must be opposite and all other edges are in T, resulting in one case with two rotations. If there is just a single M-edge, it comes with five T-edges (three in the square and two more at its ends). The remaining two can be any combination of M- and T-edges, yielding a total of four more cases with four rotations each.

The total number of cases is now 39, so significantly more than we want to check for each extension, which is why we show how these can be reduced to the 18 shown in Figure A.6. This is done by locally changing the decomposition of the eliminated cases into one that is still present.

The switches used to achieve this are shown in Figure A.7 and we describe each of them here in turn. For the first one we note that the length 2 paths through the square can

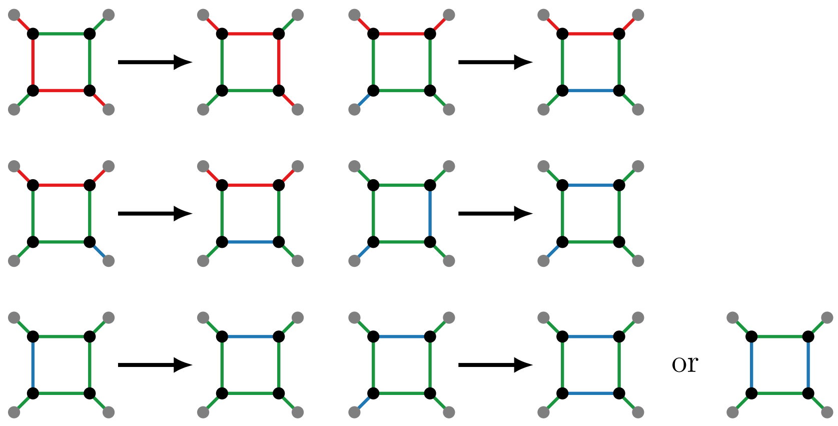


Figure A.7.: The transformations used to eliminate cases for the square. By replacing the left decomposition with the right one (or, in the last case, with one of the ones on the right) a new 3-decomposition is obtained.

be used interchangeably, removing the two edges of the tree creates two isolated vertices that are reconnected by using those previously in a cycle, saving us two cases.

Next, consider the case with a single C-edge u_1u_2 where the edge at u_3 that is not in the square is in the matching. By removing the T-edge u_3u_4, we get two components, one of which is the edge u_1u_3. By moving the M-edge at u_3 to the tree, these components are unified, giving us a new tree. The analogous transformation works if the M-edge is at u_4 and for all rotations, saving us eight cases in total.

This completes the reduction of cases with C-edges. Let us now consider the case in which the square contains a single M-edge, say u_2u_4 and there is a second one at u_3. Adding the edge u_2u_4 to the tree yields the square as the unique cycle and we can remove u_1u_2 to create a new spanning tree and 3-decomposition. As this works for the four rotations, we save another four cases. We can also remove three of the four rotations of the square with one M-edge and only T-edges elsewhere with the same argument.

Finally, the graph with M-edges u_1u_2 and u_3v_3 can be reduced to one of the two cases with two M-edges, reducing the total number of cases by four yet again. To see this, note that adding the M-edge at u_3 to the tree yields a cycle. If it uses the edge u_3u_4, we put it into the matching instead and get a new 3-decomposition. Otherwise, the edge u_3u_1 is used and the cycle consists of this edge and a path from u_1 to u_3 that does not meet the square. But we have already seen that we may exchange the edges u_1u_2 and u_2u_4 and in the resulting graph the cycle obtained when adding the M-edge at u_3 to the tree remains unchanged. This lets us swap it with u_1u_3 to obtain a new 3-decomposition.

We have now completed the switches. Note that there are symmetric cases amongst the remaining 18, but the symmetries we may use also depend on which are present in the extension. We cannot use ninety degree rotations, for example, as neither the domino, the twin-house, nor the claw-square have this symmetry.

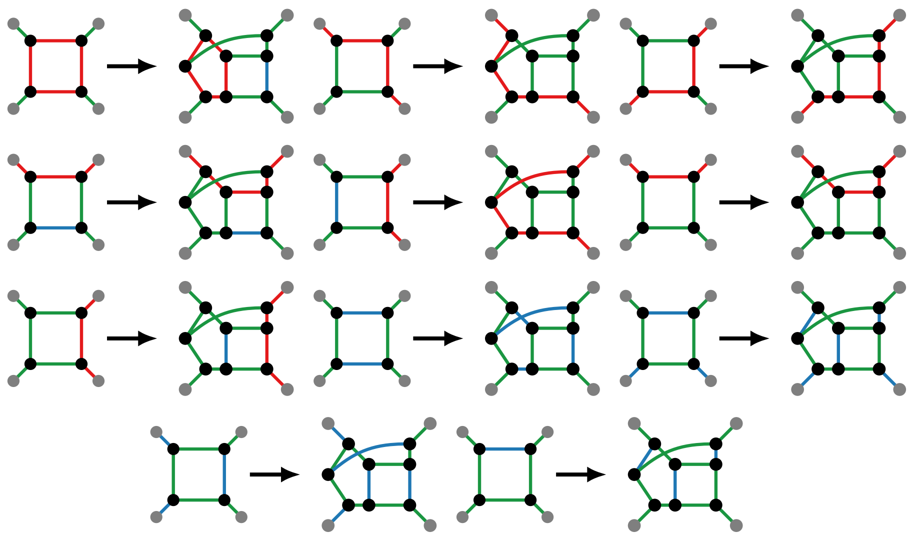
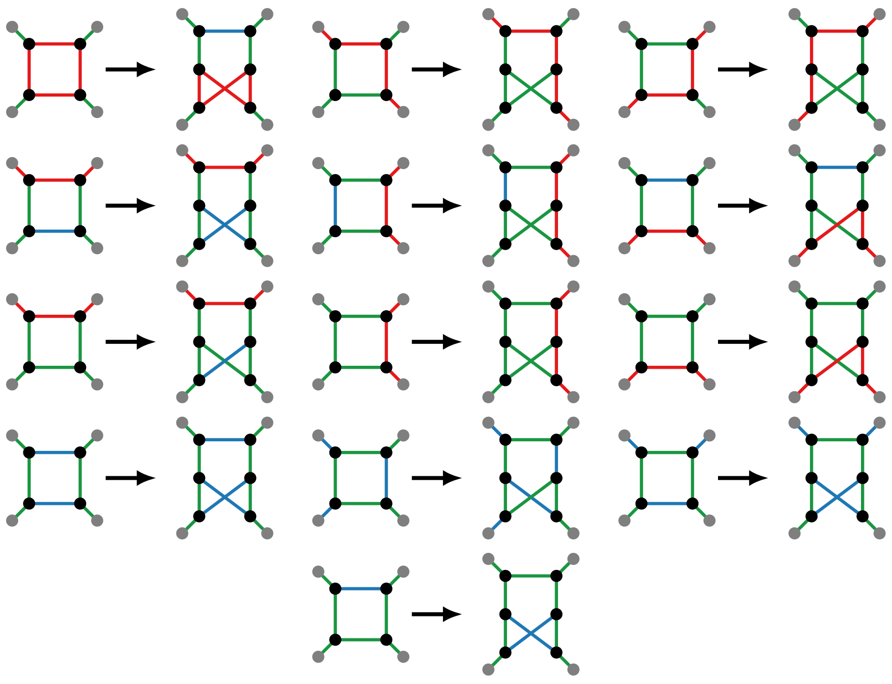


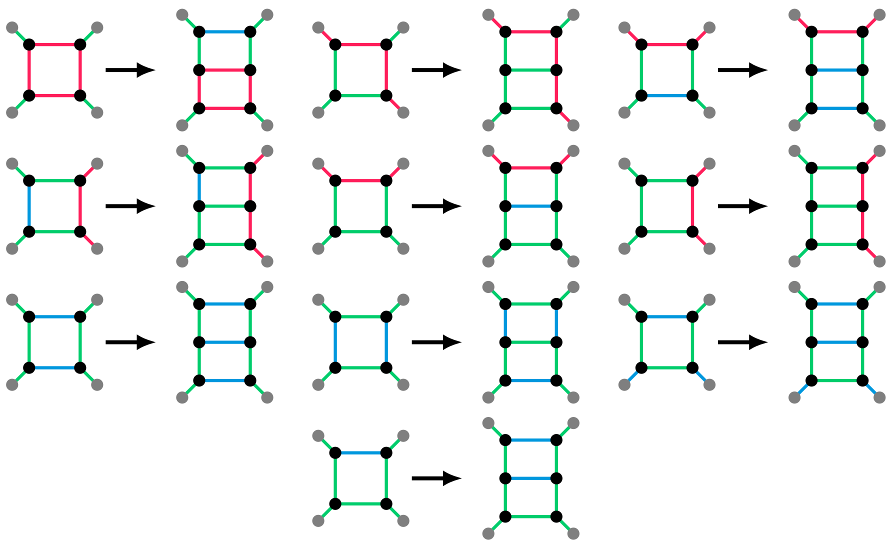
Figure A.8.: Extending a 3-decomposition from a square to a claw-square. (5.12)

The claw-square. We can now look at extension of the square to the claw-square, seen in Figure 5.6b. Here we can eliminate the behaviours at position 6, 7, 10, 11, 13, 16, and 17 in Figure A.6 by symmetry. We are left with the eleven behaviours shown in Figure A.8.

The twin-house. The next extension of the square that we consider is the one to the twin-house, seen in Figure 5.8a. Of the behaviours from Figure A.6, we can omit the ones at position 7, 11, and 17, by symmetry. We have covered behaviours 13 and 14 in Lemma 5.17, leaving the thirteen shown in Figure A.9.

The domino, part II. We finish by completing the missing extension for the domino, seen on the right of Figure 5.10. Of the behaviours from Figure A.6, we can omit the following due to the domino's symmetries: 3, 6, 7, 10, 11, 16, and 17, where the number indicates their position. We have covered behaviour 15 in Lemma 5.17, leaving the ten shown in Figure A.10.


(5.15) Figure A.9.: Extending a 3-decomposition from a square to a twin-house.


(5.17) Figure A.10.: Extending a 3-decomposition from a square to a domino.

Bibliography

[AAA18] E. Aboomahigir, M. Ahanjideh, and S. Akbari. *Decomposing Claw-Free Subcubic Graphs and 4-Chordal Subcubic Graphs.* Version 2. 2018. arXiv: `1806.11009 [math.CO]`.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* First edition. Cambridge University Press, 2009. ISBN: 978-0-511-80409-0.

[Abd+16] F. Abdolhosseini et al. *Hoffmann-Ostenhof's Conjecture for Traceable Cubic Graphs.* Version 1. 2016. arXiv: `1607.04768 [math.CO]`.

[Abr+10] I. Abraham et al. "Alternative Routes in Road Networks". *Experimental Algorithms.* 9th International Symposium, SEA 2010. Lecture Notes in Computer Science 6049. Springer Berlin Heidelberg, 2010, pages 23–34. DOI: `10.1007/978-3-642-13193-6_3`.

[AEB00] V. Akgün, E. Erkut, and R. Batta. "On Finding Dissimilar Paths". *European Journal of Operational Research* 121.2 (2000), pages 232–246. DOI: `10.1016/S0377-2217(99)00214-3`.

[AF90] M. Ajtai and R. Fagin. "Reachability Is Harder for Directed than for Undirected Finite Graphs". *Journal of Symbolic Logic* 55.1 (1990), pages 113–150. DOI: `10.2307/2274958`.

[AH77] K. Appel and W. Haken. "Every Planar Map Is Four Colorable. Part I: Discharging". *Illinois Journal of Mathematics* 21.3 (1977), pages 429–490. DOI: `10.1215/ijm/1256049011`.

[AHK77] K. Appel, W. Haken, and J. Koch. "Every Planar Map Is Four Colorable. Part II: Reducibility". *Illinois Journal of Mathematics* 21.3 (1977), pages 491–567. DOI: `10.1215/ijm/1256049012`.

[AJS15] S. Akbari, T. R. Jensen, and M. Siggers. "Decompositions of Graphs into Trees, Forests, and Regular Subgraphs". *Discrete Mathematics* 338.8 (2015), pages 1322–1327. DOI: `10.1016/j.disc.2015.02.021`.

[AKY97] Y. Afek, S. Kutten, and M. Yung. "The Local Detection Paradigm and Its Applications to Self-Stabilization". *Theoretical Computer Science* 186.1 (1997), pages 199–229. DOI: `10.1016/S0304-3975(96)00286-1`.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993. ISBN: 978-0-13-617549-0.

[AT85] N. Alon and M. Tarsi. “Covering Multigraphs by Simple Circuits”. *SIAM Journal on Algebraic Discrete Methods* 6.3 (1985), pages 345–350. DOI: 10.1137/0606035.

[Bac15] A. C. Bachstein. “Decomposition of Cubic Graphs on the Torus and Klein Bottle”. Master’s thesis. Middle Tennessee State University, 2015. URL: http://jewlscholar.mtsu.edu/handle/mtsu/4753.

[Bad+11] R. Bader et al. “Alternative Route Graphs in Road Networks”. *Theory and Practice of Algorithms in (Computer) Systems.* 1st International ICST Conference, TAPAS 2011. Lecture Notes in Computer Science 6595. Springer, 2011, pages 21–32. DOI: 10.1007/978-3-642-19754-3_5.

[Bal+05] J. Balogh et al. “Covering Planar Graphs with Forests”. *Journal of Combinatorial Theory, Series B* 94.1 (2005), pages 147–158. DOI: 10.1016/j.jctb.2004.12.002.

[Bat+62] J. Battle et al. “Additivity of the Genus of a Graph”. *Bulletin of the American Mathematical Society* 68.6 (1962), pages 565–568. DOI: 10.1090/S0002-9904-1962-10847-7.

[BBK22a] O. Bachtler, T. Bergner, and S. O. Krumke. *Almost Disjoint Paths and Separating by Forbidden Pairs.* Version 1. 2022. arXiv: 2202.10090 [math.CO].

[BBK22b] O. Bachtler, T. Bergner, and S. O. Krumke. “Local Certification of Reachability”. *Proceedings of the 10th International Network Optimization Conference.* INOC 2022. OpenProceedings.org, 2022, pages 40–44. DOI: 10.48786/inoc.2022.08.

[BH20] O. Bachtler and I. Heinrich. *Automated Testing and Interactive Construction of Unavoidable Sets for Graph Classes of Small Path-Width.* Version 1. 2020. arXiv: 2010.08373 [math.CO].

[BH21] O. Bachtler and I. Heinrich. *Reductions for the 3-Decomposition Conjecture.* Version 2. 2021. arXiv: 2104.15113 [math.CO].

[BK11] H. L. Bodlaender and A. M. C. A. Koster. “Treewidth Computations II. Lower Bounds”. *Information and Computation* 209.7 (2011), pages 1103–1119. DOI: 10.1016/j.ic.2011.04.003.

[BK17] K. Berczi and Y. Kobayashi. “The Directed Disjoint Shortest Paths Problem”. *25th Annual European Symposium on Algorithms.* ESA 2017. Leibniz International Proceedings in Informatics (LIPIcs) 87. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. DOI: 10.4230/LIPIcs.ESA.2017.13.

[BK22] O. Bachtler and S. O. Krumke. “Towards Obtaining a 3-Decomposition from a Perfect Matching”. *The Electronic Journal of Combinatorics* 29.4 (2022). DOI: 10.37236/11128.

[Bla+15] M. Blanco et al. “On the Path Avoiding Forbidden Pairs Polytope”. *Electronic Notes in Discrete Mathematics* 50 (2015), pages 343–348. DOI: 10.1016/j.endm.2015.07.057.

[BM08] J. A. Bondy and U. S. R. Murty. *Graph Theory*. First edition. Springer, 2008. ISBN: 978-1-84996-690-0.

[Bod+19] H. L. Bodlaender et al. *Knot Diagrams of Treewidth Two*. Version 2. 2019. arXiv: 1904.03117 [cs.DS].

[Bod98] H. L. Bodlaender. "A Partial k-Arboretum of Graphs with Bounded Treewidth". *Theoretical Computer Science* 209.1 (1998), pages 1–45. DOI: 10.1016/S0304-3975(97)00228-4.

[Bot+21] F. Botler et al. "The 2-Decomposition Conjecture for a New Class of Graphs". *Procedia Computer Science* 195 (2021), pages 359–367. DOI: 10.1016/j.procs.2021.11.044.

[Bri+13] G. Brinkmann et al. "House of Graphs: A Database of Interesting Graphs". *Discrete Applied Mathematics* 161.1 (2013), pages 311–314. DOI: 10.1016/j.dam.2012.07.018.

[Bri96] G. Brinkmann. "Fast Generation of Cubic Graphs". *Journal of Graph Theory* 23.2 (1996), pages 139–149. DOI: 10.1002/(sici)1097-0118(199610)23:2<139::aid-jgt5>3.0.co;2-u.

[Cam11] P. J. Cameron. "Research Problems from the BCC22". *Discrete Mathematics* 311.13 (2011), pages 1074–1083. DOI: 10.1016/j.disc.2011.02.024.

[CE83] L. Clark and R. Entringer. "Smallest Maximally Nonhamiltonian Graphs". *Periodica Mathematica Hungarica* 14.1 (1983), pages 57–68. DOI: 10.1007/BF02023582.

[CFS15] D. Conlon, J. Fox, and B. Sudakov. "Recent Developments in Graph Ramsey Theory". *Surveys in Combinatorics 2015*. London Mathematical Society Lecture Note Series 424. Cambridge University Press, 2015. ISBN: 978-1-316-10685-3.

[Che+01] T. Chen et al. "A Dynamic Programming Approach to De Novo Peptide Sequencing via Tandem Mass Spectrometry". *Journal of Computational Biology* 8.3 (2001), pages 325–337. DOI: 10.1089/10665270152530872.

[Cho+18] T. Chondrogiannis et al. "Finding k-Dissimilar Paths with Minimum Collective Length". *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '18. Association for Computing Machinery, 2018, pages 404–407. DOI: 10.1145/3274895.3274903.

[Cho+20] T. Chondrogiannis et al. "Finding k-Shortest Paths with Limited Overlap". *The VLDB Journal* 29.5 (2020), pages 1023–1047. DOI: 10.1007/s00778-020-00604-x.

[Chu+16] M. Chudnovsky et al. "Unavoidable Induced Subgraphs in Large Graphs with No Homogeneous Sets". *Journal of Combinatorial Theory, Series B* 118 (2016), pages 1–12. DOI: 10.1016/j.jctb.2016.01.008.

[CM93] B. Courcelle and M. Mosbah. “Monadic Second-Order Evaluations on Tree-Decomposable Graphs”. *Theoretical Computer Science* 109.1 (1993), pages 49–82. DOI: 10.1016/0304-3975(93)90064-Z.

[Cor+22] T. H. Cormen et al. *Introduction to Algorithms*. Fourth edition. MIT press, 2022. ISBN: 978-0-262-04630-5.

[Cou90] B. Courcelle. “The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs”. *Information and Computation* 85.1 (1990), pages 12–75. DOI: 10.1016/0890-5401(90)90043-H.

[CS05] L. S. Chandran and C. R. Subramanian. “Girth and Treewidth”. *Journal of Combinatorial Theory, Series B* 93.1 (2005), pages 23–32. DOI: 10.1016/j.jctb.2004.05.004.

[CS13] G. Chen and S. Shan. “Homeomorphically Irreducible Spanning Trees”. *Journal of Combinatorial Theory, Series B* 103.4 (2013), pages 409–414. DOI: 10.1016/j.jctb.2013.04.001.

[CW17] D. W. Cranston and D. B. West. “An Introduction to the Discharging Method via Graph Coloring”. *Discrete Mathematics* 340.4 (2017), pages 766–793. DOI: 10.1016/j.disc.2016.11.022.

[DF55] G. B. Dantzig and D. R. Fulkerson. *On the Max Flow Min Cut Theorem of Networks*. RAND Corporation, 1955. URL: https://www.rand.org/pubs/papers/P826.html.

[DGS05] P. Dell’Olmo, M. Gentili, and A. Scozzari. “On Finding Dissimilar Pareto-Optimal Paths”. *European Journal of Operational Research* 162.1 (2005), pages 70–82. DOI: 10.1016/j.ejor.2003.10.033.

[Die16] R. Diestel. *Graph Theory*. Fifth edition. Graduate Texts in Mathematics 173. Springer, 2016. ISBN: 978-3-662-53622-3.

[Dir52] G. A. Dirac. “Some Theorems on Abstract Graphs”. *Proceedings of the London Mathematical Society* s3-2.1 (1952), pages 69–81. DOI: 10.1112/plms/s3-2.1.69.

[Dol00] S. Dolev. *Self-Stabilization*. MIT press, 2000. ISBN: 978-0-262-52921-1.

[Eil98] T. Eilam-Tzoreff. “The Disjoint Shortest Paths Problem”. *Discrete Applied Mathematics* 85.2 (1998), pages 113–138. DOI: 10.1016/S0166-218X(97)00121-2.

[EIS76] S. Even, A. Itai, and A. Shamir. “On the Complexity of Timetable and Multicommodity Flow Problems”. *SIAM Journal on Computing* 5.4 (1976), pages 691–703. DOI: 10.1137/0205048.

[EJ12] G. Exoo and R. Jajcay. “Dynamic Cage Survey”. *The electronic journal of combinatorics* (2012). DOI: 10.37236/37.

[Feu+20] L. Feuilloley et al. *Compact Distributed Certification of Planar Graphs*. Version 1. 2020. arXiv: 2005.05863 [cs.DC].

[Feu21] L. Feuilloley. “Introduction to Local Certification”. *Discrete Mathematics & Theoretical Computer Science* 23.3 (2021). DOI: 10.46298/dmtcs.6280.

[FG04] M. Frick and M. Grohe. “The Complexity of First-Order and Monadic Second-Order Logic Revisited”. *Annals of Pure and Applied Logic* 130.1 (2004), pages 3–31. DOI: 10.1016/j.apal.2004.01.007.

[FGH20] E. Fuchs, L. Gellert, and I. Heinrich. “Cycle Decompositions of Pathwidth-6 Graphs”. *Journal of Graph Theory* 94.2 (2020), pages 224–251. DOI: 10.1002/jgt.22516.

[FHW80] S. Fortune, J. Hopcroft, and J. Wyllie. “The Directed Subgraph Homeomorphism Problem”. *Theoretical Computer Science* 10.2 (1980), pages 111–121. DOI: 10.1016/0304-3975(80)90009-2.

[Foe+18] K.-T. Foerster et al. “Local Checkability, No Strings Attached: (A)Cyclicity, Reachability, Loop Free Updates in SDNs”. *Theoretical Computer Science* 709 (2018), pages 48–63. DOI: 10.1016/j.tcs.2016.11.018.

[FR94] G. Fan and A. Raspaud. “Fulkerson’s Conjecture and Circuits Covers”. *Journal of Combinatorial Theory, Series B* 61.1 (1994), pages 133–138. DOI: 10.1006/jctb.1994.1039.

[FS07] M. Frick and J. Singleton. “Cubic Maximal Nontraceable Graphs”. *Discrete Mathematics* 307.7 (2007), pages 885–891. DOI: 10.1016/j.disc.2005.11.039.

[Ful71] D. R. Fulkerson. “Blocking and Anti-blocking Pairs of Polyhedra”. *Mathematical programming* 1.1 (1971), pages 168–194. DOI: 10.1007/BF01584085.

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990. ISBN: 978-0-7167-1045-5.

[GKT51] D. Gale, H. W. Kuhn, and A. W. Tucker. “Linear Programming and the Theory of Games”. *Activity analysis of production and allocation* 13 (1951), pages 317–335.

[GMO76] H. N. Gabow, S. N. Maheshwari, and L. J. Osterweil. “On Two Problems in the Generation of Program Test Paths”. *IEEE Transactions on Software Engineering* SE-2.3 (1976), pages 227–231. DOI: 10.1109/TSE.1976.233819.

[Gon09] D. Gonçalves. “Covering Planar Graphs with Forests, One Having Bounded Maximum Degree”. *Journal of Combinatorial Theory, Series B* 99.2 (2009), pages 314–322. DOI: 10.1016/j.jctb.2008.07.004.

[GS16] M. Göös and J. Suomela. “Locally Checkable Proofs in Distributed Computing”. *Theory of Computing* 12.19 (2016), pages 1–33. DOI: 10.4086/toc.2016.v012a019.

[Haa19] R. de Haan. *Parameterized Complexity in the Polynomial Hierarchy*. First edition. Lecture Notes in Computer Science. Springer, 2019. ISBN: 978-3-662-60670-4.

[Haj+10] M. T. Hajiaghayi et al. “The Checkpoint Problem”. *Approximation, Randomization, and Combinatorial Optimization.* 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010. Lecture Notes in Computer Science 6302. Springer, 2010, pages 219–231. DOI: `10.1007/978-3-642-15369-3_17`.

[He+02] W. He et al. “Edge-Partitions of Planar Graphs and Their Game Coloring Numbers”. *Journal of Graph Theory* 41.4 (2002), pages 307–317. DOI: `10.1002/jgt.10069`.

[Hei19] I. Heinrich. “On Graph Decomposition: Hajós’ Conjecture, the Clustering Coefficient and Dominating Sets”. PhD thesis. Technische Universität Kaiserslautern, 2019.

[Hei20] I. Heinrich. *On Graph Decomposition: Hajós’ conjecture, the clustering coefficient and dominating sets*. Dr. Hut, 2020. ISBN: 978-3-8439-4294-2.

[HKO18] A. Hoffmann-Ostenhof, T. Kaiser, and K. Ozeki. “Decomposing Planar Cubic Graphs”. *Journal of Graph Theory* 88.4 (2018), pages 631–640. DOI: `10.1002/jgt.22234`.

[HLY20] Y. Hong, Q. Liu, and N. Yu. “Edge Decomposition of Connected Claw-Free Cubic Graphs”. *Discrete Applied Mathematics* 284 (2020), pages 246–250. DOI: `10.1016/j.dam.2020.03.040`.

[HNO18] A. Hoffmann-Ostenhof, K. Noguchi, and K. Ozeki. “On Homeomorphically Irreducible Spanning Trees in Cubic Graphs”. *Journal of Graph Theory* 89.2 (2018), pages 93–100. DOI: `10.1002/jgt.22242`.

[Hof11] A. Hoffmann-Ostenhof. “Nowhere-Zero Flows and Structures in Cubic Graphs”. PhD thesis. University of Vienna, 2011. DOI: `10.25365/thesis.19501`.

[Hof15] A. Hoffmann-Ostenhof. “A Survery on the 3-Decomposition Conjecture”. Talk. 8th Workshop on the Matthews-Sumner Conjecture and Related Problems. 2015. URL: `http://www.iti.zcu.cz/plzen15/talks/1-2a-Arthur-Survey_decomposition.ppt`.

[HS93] D. A. Holton and J. Sheehan. *The Petersen Graph.* Australian Mathematical Society Lecture Series 7. Cambridge University Press, 1993. ISBN: 978-0-511-66205-8.

[HT73] J. Hopcroft and R. Tarjan. “Algorithm 447: Efficient Algorithms for Graph Manipulation”. *Communucations of the ACM* 16.6 (1973), pages 372–378. DOI: `10.1145/362248.362272`.

[Huc00] A. Huck. “Reducible Configurations for the Cycle Double Cover Conjecture”. *Discrete Applied Mathematics* 99.1 (2000), pages 71–90. DOI: `10.1016/S0166-218X(99)00126-2`.

[Isa75] R. Isaacs. “Infinite Families of Nontrivial Trivalent Graphs Which Are Not Tait Colorable”. *The American Mathematical Monthly* 82.3 (1975), pages 221–239. DOI: 10.1080/00029890.1975.11993805.

[Jae85] F. Jaeger. “A Survey of the Cycle Double Cover Conjecture”. *Annals of Discrete Mathematics (27): Cycles in Graphs*. North-Holland Mathematics Studies 115. Elsevier, 1985. ISBN: 978-0-444-87803-8.

[Jeo+09] Y.-J. Jeong et al. “A Dissimilar Alternative Paths-Search Algorithm for Navigation Services: A Heuristic Approach”. *KSCE Journal of Civil Engineering* 14.1 (2009), pages 41–49. DOI: 10.1007/s12205-010-0041-8.

[KK06] A. Korman and S. Kutten. “Distributed Verification of Minimum Spanning Trees”. *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*. PODC ’06. Association for Computing Machinery, 2006, pages 26–34. DOI: 10.1145/1146381.1146389.

[KKP10] A. Korman, S. Kutten, and D. Peleg. “Proof Labeling Schemes”. *Distributed Computing* 22.4 (2010), pages 215–233. DOI: 10.1007/s00446-010-0095-3.

[KKR12] K. Kawarabayashi, K. Kobayashi, and B. Reed. “The Disjoint Paths Problem in Quadratic Time”. *Journal of Combinatorial Theory, Series B* 102.2 (2012), pages 424–435. DOI: 10.1016/j.jctb.2011.07.004.

[KN12] S. O. Krumke and H. Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. German. Third edition. Teubner, 2012. ISBN: 978-3-8348-1849-2.

[Kov13] J. Kováč. “Complexity of the Path Avoiding Forbidden Pairs Problem Revisited”. *Discrete Applied Mathematics* 161.10 (2013), pages 1506–1512. DOI: 10.1016/j.dam.2012.12.022.

[KP09] P. Kolman and O. Pangrác. “On the Complexity of Paths Avoiding Forbidden Pairs”. *Discrete Applied Mathematics* 157.13 (2009), pages 2871–2876. DOI: 10.1016/j.dam.2009.03.018.

[KSG73] K. W. Krause, R. W. Smith, and M. A. Goodwin. “Optimal Software Test Planning Through Automated Network Analysis”. *Proceedings 1973 IEEE Symposium on Computer Software Reliability*. IEEE Press, 1973, pages 18–22.

[KVB09] J. Kováč, T. Vinař, and B. Brejová. “Predicting Gene Structures from Multiple RT-PCR Tests”. *Algorithms in Bioinformatics*. 9th International Workshop, WABI 2009. Lecture Notes in Computer Science 5724. Springer, 2009, pages 181–193. DOI: 10.1007/978-3-642-04241-6_16.

[LC14] R. Li and Q. Cui. “Spanning Trees in Subcubic Graphs”. *Ars Combinatoria* 117 (2014), pages 411–415. URL: http://www.combinatoire.ca/ArsCombinatoria/AC_article10.pdf.

[Liu+18] H. Liu et al. “Finding Top-k Shortest Paths with Diversity”. *IEEE Transactions on Knowledge and Data Engineering* 30.3 (2018), pages 488–502. DOI: 10.1109/TKDE.2017.2773492.

[LL20] P. Li and W. Liu. “Decompositions of Cubic Traceable Graphs”. *Discussiones Mathematicae Graph Theory* 40.1 (2020), pages 35–49. DOI: `10.7151/dmgt.2132`.

[LM19] K. S. Lyngsie and M. Merker. “Decomposing Graphs into a Spanning Tree, an Even Graph, and a Star Forest”. English. *The Electronic Journal of Combinatorics* 26.1 (2019). DOI: `10.37236/7970`.

[Men27] K. Menger. “Zur Allgemeinen Kurventheorie”. German. *Fundamenta Mathematicae* 10.1 (1927), pages 96–115. DOI: `10.4064/fm-10-1-96-115`.

[Mer99] M. Meringer. “Fast Generation of Regular Graphs and Construction of Cages”. *Journal of Graph Theory* 30.2 (1999), pages 137–146. DOI: `10.1002/(sici)1097-0118(199902)30:2<137::aid-jgt7>3.0.co;2-g`.

[MM20] E. Máčajová and G. Mazzuoccolo. “Reduction of the Berge-Fulkerson Conjecture to Cyclically 5-Edge-Connected Snarks”. *Proceedings of the American Mathematical Society* 148.11 (2020), pages 4643–4652. DOI: `10.1090/proc/15057`.

[OY16] K. Ozeki and D. Ye. “Decomposing Plane Cubic Graphs”. *European Journal of Combinatorics* 52 (2016), pages 40–46. DOI: `10.1016/j.ejc.2015.08.005`.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. ISBN: 978-0-201-53082-7.

[Pet91] J. Petersen. “Die Theorie der Regulären Graphs”. *Acta Mathematica* 15.1 (1891), page 193. DOI: `10.1007/BF02392606`.

[Pos46] E. L. Post. “A Variant of a Recursively Unsolvable Problem”. *Bulletin of the American Mathematical Society* 52.4 (1946), pages 264–268. DOI: `10.1090/S0002-9904-1946-08555-9`.

[Rot95] J. J. Rotman. *An Introduction to the Theory of Groups*. Fourth edition. Graduate Texts in Mathematics 148. Springer, 1995. ISBN: 978-1-4612-4176-8.

[RS04] N. Robertson and P. D. Seymour. “Graph Minors. XX. Wagner’s Conjecture”. *Journal of Combinatorial Theory, Series B* 92.2 (2004), pages 325–357. DOI: `10.1016/j.jctb.2004.08.001`.

[RS90] N. Robertson and P. D. Seymour. “Graph minors. VIII. A Kuratowski Theorem for General Surfaces”. *Journal of Combinatorial Theory, Series B* 48.2 (1990), pages 255–288. DOI: `10.1016/0095-8956(90)90121-F`.

[RS95] N. Robertson and P. D. Seymour. “Graph Minors. XIII. The Disjoint Paths Problem”. *Journal of Combinatorial Theory, Series B* 63.1 (1995), pages 65–110. DOI: `10.1006/jctb.1995.1006`.

[Sage20] The Sage Developers. *SageMath, the Sage Mathematics Software System*. Version 9.0. 2020. URL: `https://www.sagemath.org`.

[SB77] S. Stahl and L. W. Beineke. "Blocks and the Nonorientable Genus of Graphs". *Journal of Graph Theory* 1.1 (1977), pages 75–78. DOI: 10.1002/jgt.3190010114.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. First edition. Springer, 2003. ISBN: 978-3-540-44389-6.

[Sey79] P. D. Seymour. "Sums of Circuits". *Graph Theory and Related Topics* 1 (1979), pages 341–355.

[Suu74] J. W. Suurballe. "Disjoint Paths in a Network". *Networks* 4.2 (1974), pages 125–145. DOI: 10.1002/net.3230040204.

[Sze73] G. Szekeres. "Polyhedral Decompositions of Cubic Graphs". *Bulletin of the Australian Mathematical Society* 8.3 (1973), pages 367–387. DOI: 10.1017/S0004972700042660.

[TT81] C. Thomassen and B. Toft. "Non-Separating Induced Cycles in Graphs". *Journal of Combinatorial Theory, Series B* 31.2 (1981), pages 199–224. DOI: 10.1016/S0095-8956(81)80025-1.

[TUK94] A. Takahashi, S. Ueno, and Y. Kajitani. "Minimal Acyclic Forbidden Minors for the Family of Graphs with Bounded Path-Width". *Discrete Mathematics* 127.1 (1994), pages 293–304. DOI: 10.1016/0012-365X(94)90092-2.

[Tut19] W. T. Tutte. *Connectivity in Graphs*. University of Toronto Press, 2019. ISBN: 978-1-4875-8486-3.

[Tut47] W. T. Tutte. "A Family of Cubical Graphs". *Mathematical Proceedings of the Cambridge Philosophical Society* 43.4 (1947), pages 459–474. DOI: 10.1017/S0305004100023720.

[Tut66] W. T. Tutte. *Connectivity in Graphs*. University of Toronto Press, 1966. ISBN: 978-1-4875-7296-9.

[Vyg95] J. Vygen. "NP-Completeness of Some Edge-Disjoint Paths Problems". *Discrete Applied Mathematics* 61.1 (1995), pages 83–90. DOI: 10.1016/0166-218X(93)E0177-Z.

[Wes01] D. B. West. *Introduction to Graph Theory*. Second edition. Prentice Hall, 2001. ISBN: 978-0-13-014400-3.

[Wor79] N. C. Wormald. "Classifying k-Connected Cubic Graphs". *Combinatorial Mathematics VI*. 6th Australian Conference on Combinatorial Mathematics. Lecture Notes in Mathematics 748. Springer, 1979, pages 199–206. DOI: 10.1007/BFB0102696.

[WZ11] Y. Wang and Q. Zhang. "Decomposing a Planar Graph with Girth at Least 8 into a Forest and a Matching". *Discrete Mathematics* 311.10 (2011), pages 844–849. DOI: 10.1016/j.disc.2011.01.019.

[XZZ20] M. Xie, C. Zhou, and S. Zhou. "Decomposition of Cubic Graphs with a 2-Factor Consisting of Three Cycles". *Discrete Mathematics* 343.6 (2020). DOI: 10.1016/j.disc.2020.111839.

[Yin97] H. Yinnone. “On Paths Avoiding Forbidden Pairs of Vertices in a Graph”. *Discrete Applied Mathematics* 74.1 (1997), pages 85–92. DOI: `10.1016/S0166-218X(96)00017-0`.

Curriculum Vitae

Oliver Bachtler

2019 – today	**Doctoral Studies in Mathematics**, *TU Kaiserslautern*
2019 – today	**Research Assistant**, *TU Kaiserslautern*
2014 – 2019	**Student Assistant**, *TU Kaiserslautern*
2013 – 2019	**Studies in Computer Science**, *TU Kaiserslautern*
	• B.Sc. Computer Science (March 2017)
	• M.Sc. Computer Science (July 2019)
2017 – 2018	**Semester Abroad**, *University of Southern Denmark*, Odense
2013 – 2018	**Studies in Mathematics**, *TU Kaiserslautern*
	• B.Sc. Mathematics (January 2017)
	• M.Sc. Mathematics (November 2018)
2013	**Abitur** (high school graduation), *Otto-Hahn-Gymnasium*, Landau

Wissenschaftlicher Werdegang

Oliver Bachtler

2019 – heute	**Promotion: Mathematik**, *TU Kaiserslautern*
2019 – heute	**Wissenschaftlicher Mitarbeiter**, TU Kaiserslautern
2014 – 2019	**Studentische Hilfskraft**, TU Kaiserslautern
2013 – 2019	**Studium: Informatik**, *TU Kaiserslautern*
	• B.Sc. Informatik (März 2017)
	• M.Sc. Informatik (Juli 2019)
2017 – 2018	**Auslandssemester**, *University of Southern Denmark*, Odense
2013 – 2018	**Studium: Mathematik**, *TU Kaiserslautern*
	• B.Sc. Mathematik (Januar 2017)
	• M.Sc. Mathematik (November 2018)
2013	**Abitur**, *Otto-Hahn-Gymnasium*, Landau