KU LEUVEN

FACULTY OF SCIENCES

Al-partitions of cubic graphs

Jurre Put

Promotor: Stijn Cambie
Affiliation KULeuven

Copromotor: Jorik Jooken
Affiliation KULeuven

Master thesis submitted in fulfillment
of the requirements for the degree in
Master of Mathematics

Academic year 2024-2025

preface

This thesis was written as a part of my master's degree program at the Catholic University of Leuven. In this work I explore the structure of AI-partitions in graphs and Digraphs. There is also a computational component where I wrote my own SageMath code to check graphs for AI-partitions. I found the process of writing this very enriching, as it allowed me to develop new skills and a deeper understanding of the topic. I would like to express my gratitude to my promotor Stijn Cambie for his guidance and support throughout this project and my copromotor Jorik Jooken for his valuable input. Lastly, I also want to thank my family and friends for the emotional support through my study years.

Summary

A coherent decycling partition is a partition into a tree and an (near-) independent set. In 1975 Payan and Sakarovitch proved that any cubic cyclically 4-edge-connected graph has such a partition. We provide constructions to find infinite families of graphs without coherent decycling partitions for cubic graphs with smaller connectivity. Furthermore, R. Nedela, M. Seifrtová, and M. Škoviera proposed the conjecture that any 3-connected upper-embeddable cubic graph has such a partition. This is a weaker condition than cyclically 4-edge-connected, so the conjecture is a stronger claim. They already found counterexamples for the 1- and 2-connected case. We analyse the structure of these to come up with a condition necessary for a 3-connected counterexample. As a final part we look at a recent result by S. Cambie, F. Dross, K. Knauer, H. La, and P. Valicov about AI-partitions in subcubic graphs and look for some minimum counterexamples to stronger variations of their theorem using an algorithm written in SageMath.

Glossary

$\beta(G)$ the first Betti number of G.

$\kappa(G)$ vertex-connectivity of G.

$\lambda(G)$ edge-connectivity of G.

$\Delta(G)$ maximum degree of vertices in G.

$\delta(G)$ minimum degree of vertices in G.

$\delta^+(G)$ minimum out-degree of vertices in G.

$\delta^-(G)$ minimum in-degree of vertices in G.

$\chi(G)$ the Euler characteristic of G.

$\gamma(S)$ the genus of a surface S.

$\gamma_M(G)$ the maximum genus of a graph G.

G_S the subgraph of G induced by S.

K_n the clique of order n.

$L(G)$ the linegraph of G.

$s(G)$ the stability number of G.

List of Figures

Contents

Chapter 1

Introduction

1.1 Basic definitions

Definition 1.1 (graph)**.** A graph is a pair (V, E) of vertices $v \in V$ and edges $e \in E$ where an edge is a pair (v, w) with $v, w \in V$. Then this edge is said to connect the vertices v and w. We use the notation $deg(v)$ for the degree of the vertex v, this is the amount of edges v is included in. $\delta(G)$ is defined as the minimum degree and $\Delta(G)$ as the maximum degree in the graph G.

An edge connecting a vertex to itself is called a loop. We call $G = (V, E)$ a multigraph if repeated edges and loops are allowed. We call it a simple graph otherwise.

Definition 1.2 (walks, paths, circuits and cycles)**.** A walk is a sequence of edges $e_1e_2...e_n$ that connect a sequence of vertices $v_1v_2...v_nv_{n+1}$ such that $e_i = (v_i, v_{i+1})$. If all the vertices in a walk are distinct, we call this a path.

A circuit is a walk where all the edges are distinct and $v_1 = v_{n+1}$. If all vertices in a circuit are distinct, we call this a cycle.

Example 1.3. Consider the set of vertices $V = \{1, 2, 3, 4, 5, 6\}$ and the set of edges $E = \{(1, 3), (1, 4), (2, 2), (3, 5), (3, 5), (4, 6), (5, 6)\}$. When we combine them we can construct the graph $G = (V, E)$ shown in Figure 1.1. This graph contains a loop in vertex 2 and has a double edge between vertices 3 and 5. This makes it a multigraph.

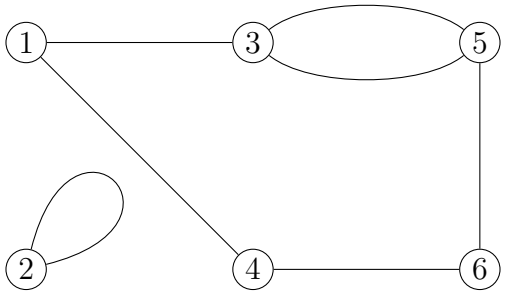


Figure 1.1: Example of a multigraph

In this definition the set E consists of unordered pairs of vertices, meaning that the edge (v, w) and (w, v) are the same. We can also consider the case where we do not want this to be the case.

Definition 1.4 (Digraph)**.** A Digraph is a pair (V, A) of vertices $v \in V$ and arcs $a \in A$. Where an arc is an ordered pair (v, w) with $v, w \in V$. We use the notation $deg^+(v)$ for the out-degree

of a vertex v, this is the amount of arcs where v is the first element. For the amount of arcs where v is the second element, or the in-degree of v, we use $deg^-(v)$.

Definition 1.5 (oriented graph)**.** An oriented graph is a Digraph without loops where there is at most one arc between any two vertices.

Example 1.6. The left Digraph in Figure 1.2 is not an oriented graph, because it has multiple arcs between the same vertices. We can remove some of them to give it an orientation and make it into an oriented Digraph.

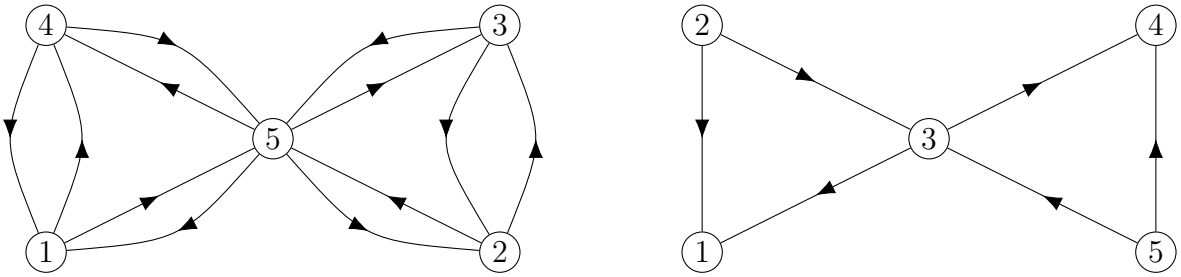
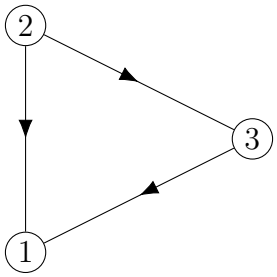


Figure 1.2: Example of an oriented Digraph (right)

An oriented graph is acyclic if it does not have oriented cycles.



While this would be a cycle if we ignore orientation, it is not a cycle in the Digraph. We also call this an unoriented cycle.

In this paper, we will focus mostly on a certain type of graphs, namely the cubic graphs.

Definition 1.7 ((sub)cubic graph)**.** A cubic graph is a graph where every vertex has degree 3. A subcubic graph is a graph G with maximum degree $\Delta(G) \leq 3$.

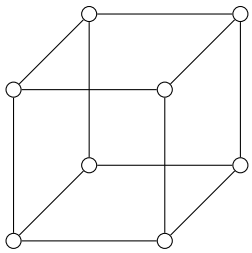


Figure 1.3: Q_3, the cube graph
an example of a cubic graph

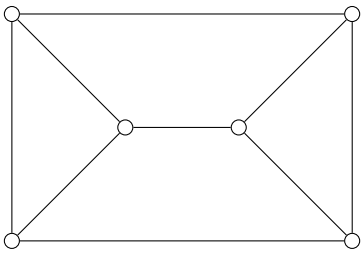


Figure 1.4: Π_3, the triangular prism graph
another example of a cubic graph

Definition 1.8 (K_n)**.** A clique is a graph with n vertices such for every pair of vertices, there is exactly one edge connecting them to each other. We also denote this by K_n.

Figure 1.5: K_n, the n-clique graph

Definition 1.9 (planar graph)**.** A planar graph is a graph that can be drawn on a plane without crossing edges.

Example 1.10. This graph does not appear to be planar at first, but we can rearrange the vertices to make it into a graph which does not have crossing edges. We also call this a planar embedding of the graph and we call a planar graph that is given a planar embedding a plane graph.

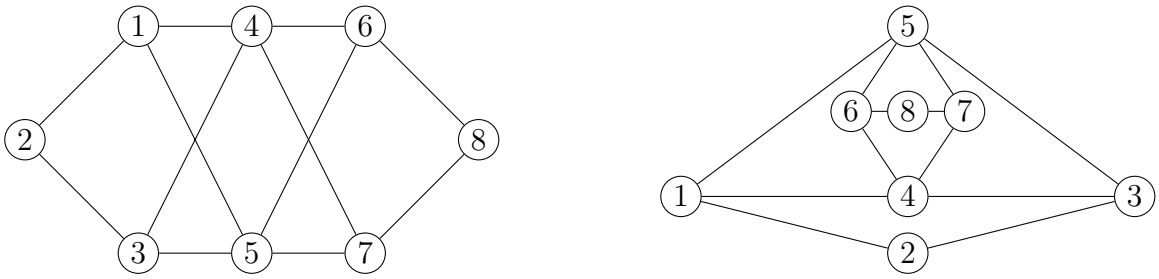


Figure 1.6: Example of a graph with non-planar embedding (left) and planar embedding (right)

Definition 1.11 (triangulation)**.** A triangulation of a plane graph G is a graph with the maximum amount of edges such that it is still a plane graph. This makes every face a triangle.

Example 1.12. The triangulation of the planar graph in Example 1.10 is the following graph.

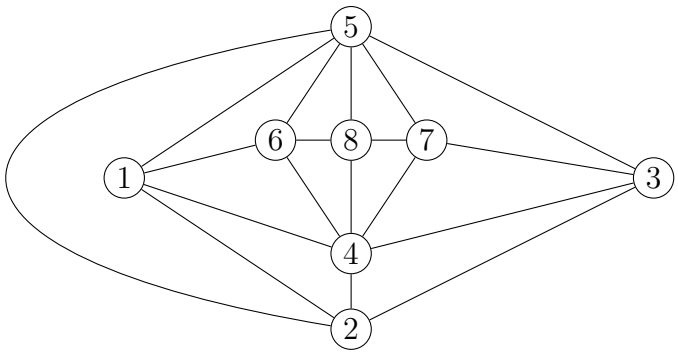


Figure 1.7: A triangulation of our planar graph

Definition 1.13 (dual graph)**.** The dual graph G^* of a planar embedded graph $G = (V, E)$ is defined as the pair (V^*, E^*) where V^* contains a vertex v_f for every face f in G and E^* contains an edge (v_{f_1}, v_{f_2}) if faces f_1 and f_2 share an edge in G.

Taking the dual graph of a dual graph gives back the original graph. The dual graph of a triangulation is always a cubic graph.

Definition 1.14 (bipartite graph)**.** A bipartite graph is a graph where every cycle consists of an even number of vertices.

Definition 1.15 (k-connected)**.** A graph is k-connected (or also k-vertex-connected) if it has more than k vertices and remains connected when fewer than k vertices are removed. We use the notation $\kappa(G) = k$ for the maximum number for which G is k-connected, also called the vertex-connectivity of G.

A related concept to this is edge-connectivity where we have a similar definition using edges instead of vertices.

Definition 1.16 (k-edge-connected)**.** A graph is k-edge-connected if it has more than $k - 1$ edges and remains connected when fewer than k edges are removed. We use the notation $\lambda(G) = k$ for the maximum number for which G is k-edge-connected, also called the edge-connectivity of G.

Example 1.17. This graph is 2-connected. When we remove a single vertex from it, it still remains connected. However, we can remove vertex 1 and vertex 4 to split the graph into 2 connected components.

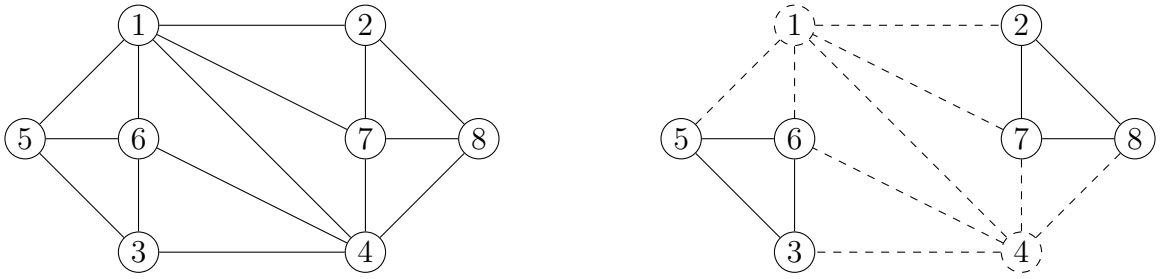


Figure 1.8: An example of a 2-connected graph

In a general simple graph, we have the inequality $\kappa(G) \leq \lambda(G) \leq \delta(G)$. When we have a set of edges (v_i, w_i) of size k that when removed disconnects the graph, we can take the set $(v_1, v_2, ..., v_k)$ which also disconnects the graph when removed. This set of vertices has size $\leq k$. When we only consider cubic graphs, the definitions actually coincide.

Lemma 1.18. For any subcubic graph G, we have $\kappa(G) = \lambda(G)$.

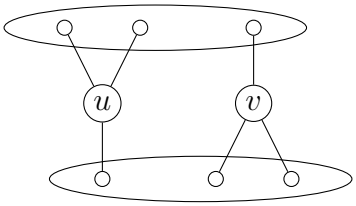
Proof. Take any cubic graph G, we already have the inequalities $\kappa(G) \leq \lambda(G) \leq 3$ so all we have left to proof is $\lambda(G) \leq \kappa(G)$ for $0 < \kappa(G) < 3$.

- $\kappa(G) = 1$

We have a vertex-cut $\{v\}$ of one vertex to disconnect the graph into at least 2 components. This means that there must be a component that is connected by one edge to v. Taking this edge gives us an edge-cut of size one, so $\lambda(G) = 1$.

- $\kappa(G) = 2$

Here we have a vertex-cut $\{v, w\}$ of size 2 that disconnects the graph into at least 2 components. If any of these components is connected by 1 or 2 edges to $\{v, w\}$, we can use those edges as edge-cut set of size ≤ 2. Assume then that every edge is connected to $\{v, w\}$ by at least 3 edges. Because G is cubic, this is only the case when there are two components each connected by 3 edges to the vertex set. It cannot be the case that a component is connected to only one of the vertices in the vertex-cut set, because then it was not a vertex-cut set of G. This means that every component is connected with one edge to one of the vertices in $\{v, w\}$ and with 2 edges to the other one.



Take this one edge for both the components, they define an edge-cut set of size 2. This gives $\lambda(G) \leq 2$ for $\kappa(G) = 2$.

When G is subcubic, the theorem also holds. If we have a vertex of degree 1, then the graph can be at most 1-connected and in that case is also 1-edge-connected. If we have a vertex of degree 2, the graph can be at most 2-connected and also at most 2-edge-connected. We only need to check the case where we have no vertices of degree less than 2 and $\kappa(G) = 1$. For this the same agrument holds as in the cubic case. This proves that $\kappa(G) = \lambda(G)$ for a subcubic graph G. □

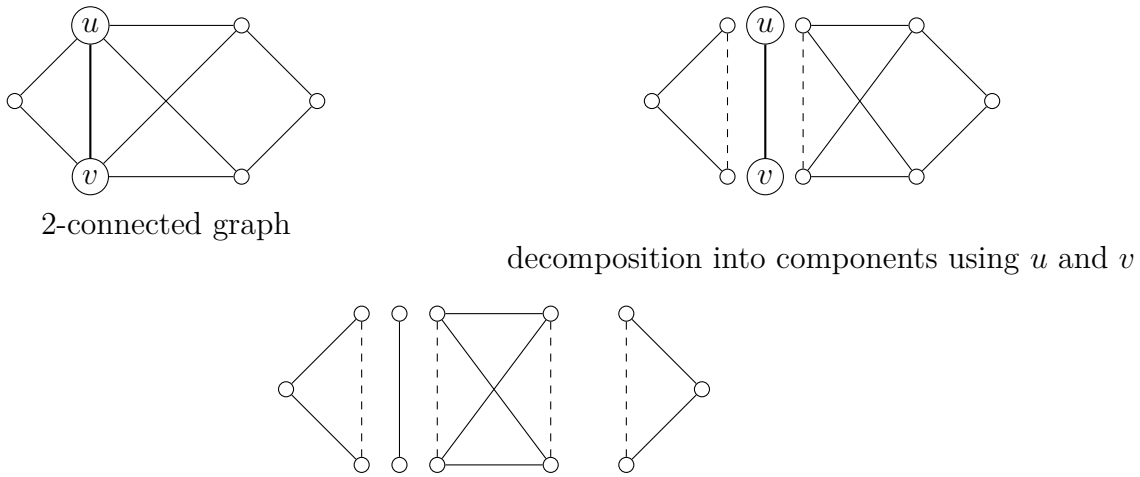
The following is a handy construction to split any 2-connected graph into 3-connected components and cycles. This is useful when we prove something for 3-connected graphs and we want to generalize this to 2-connected graphs.

1.2 The Tutte decomposition

Theorem 1.19. *Any 2-connected graph can be decomposed into bonds and components that are 3-connected graphs or cycles using a Tutte decomposition [14, 11.63].*

The way this decomposition works is defined by the following process [10]:

Take any 2-connected graph G and choose two vertices u and v in $V(G)$ that form a 2-vertex-cut. We define equivalence classes E_1, ..., E_n between vertices in $V(G)$ using the relation that two vertices are in the same class if there exists a path from u to v in G such that both vertices are on this path. Then take $F_1 = \cup_{i=1}^{k} E_i$ and $F_2 = \bigcup_{i=k+1}^{n} E_i$ such that each F_j contains at least one vertex. If there is only one equivalence class E_1 for every pair of vertices, the graph is indecomposable, otherwise we construct subgraphs G_1 and G_2 of G. Here G_i is induced by vertices in $F_i \cup \{u, v\}$, we also draw an edge between u and v which we call a virtual edge. If there was already an edge between u and v in G, then we add another component we call a bond to represent this edge. We keep repeating this process for each component we have constructed until every component is indecomposable. If we combine cycles that are next to each other in the decomposition to bigger cycles then this forms a unique decomposition.



2-connected graph

decomposition into components using u and v

Figure 1.9: Full Tutte decomposition of a 2-connected graph into indecomposable components and bonds

1.3 The 4 color theorem

One of, if not the most famous, theorems in discrete mathematics is the 4 color theorem. This theorem was first proposed in The Athenæum in 1854 as an apparently simple proposition.

Tinting Maps.—In tinting maps, it is desirable for the sake of distinctness to use as few colours as possible, and at the same time no two conterminous divisions ought to be tinted the same. Now, I have found by experience that *four* colours are necessary and sufficient for this purpose,—but I cannot prove that this is the case, unless the whole number of divisions does not exceed five. I should like to see (or know where I can find) a general proof of this apparently simple proposition, which I am surprised never to have met with in any mathematical work. F. G.

Figure 1.10: First statement of the 4-color theorem [7, figure 1]

We can explicitly state this as a theorem as follows.

Theorem 1.20. *Every planar graph has a proper vertex-coloring with 4 or less colors.*

We define a vertex-coloring to be a labeling of the vertices of the graph with colors such that no two vertices connected by an edge have the same color. In this context the graph is assumed to be without loops, otherwise there can never be a proper vertex-coloring for vertices connected to itself.

This theorem has since been proven with the help of a computer program by Appel and Haken [1]. Before this was the case there was an earlier related conjecture by Peter Guthrie Tait that relates the colorability problem to a problem of finding hamiltonian cycles.

Conjecture 1.21 (Tait's conjecture)**.** Every simple cubic planar 3-connected graph has a hamiltonian cycle.

Definition 1.22 (Hamiltonian cycle)**.** A Hamiltonian cycle is a cycle in a graph that contains every vertex other than the beginning vertex exactly once. A graph that contains a Hamiltonian cycle is also called a Hamiltonian graph.

We also define a related concept that we will be using later on.

Definition 1.23 (Eulerian circuit)**.** An Eulerian circuit is a circuit in a graph such that every edge in the graph is used exactly once. A graph that contains an Eulerian circuit is called an Eulerian graph.

At first glance Conjecture 1.21 does not seem related to Theorem 1.20 but there is a nice way to transform from the first problem to the second. We will show this using an example.

Example 1.24. We will show that Tait's conjecture implies that an arbitrary simple planar graph is 4-colorable. To see this start with any simple planar graph G. We can assume this graph is 2-vertex-connected, because if two graphs are 4-vertex-colorable and we overlap them in a single vertex, this graph would also be 4-vertex-colorable.

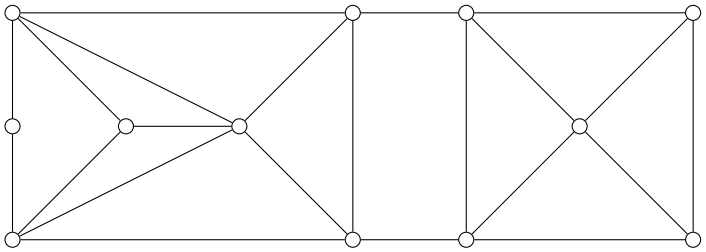


Figure 1.11: Example of a simple planar graph G

Notice that a condition for Conjecture 1.21 is that the graph has to be 3-connected. We can use the Tutte decomposition as seen in Section 1.2 to decompose our 2-connected graph into components. Out of our example G we can construct 4 such components, G_1, G_2, G_3 and G_4.

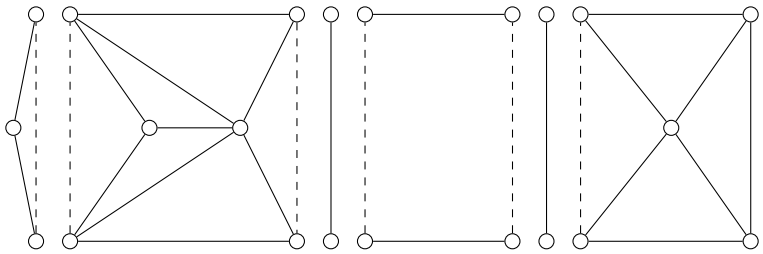


Figure 1.12: Tutte decomposition of our example G

It is clear that if all of these components are 4-colorable, then the whole graph will also be 4-colorable. This is because every component overlaps in two vertices and we can simply switch colors in one of the components such that these overlapping vertices are the same color in both of the components. Because of this, we can look at the components separately to determine if the graph is 4-colorable. An even cycle is always 2-colorable and an odd cycle is 3-colorable, so only the 3-connected components remain to be checked. Instead of the 3-connected components themselves we will actually consider their triangulations.

We can do this because by adding edges to a graph we only make the 4-color problem harder. If the triangulation has a 4-coloring, then the original graph will have the same 4-coloring.

Here we will consider the triangulation of the second component G_2 of G.

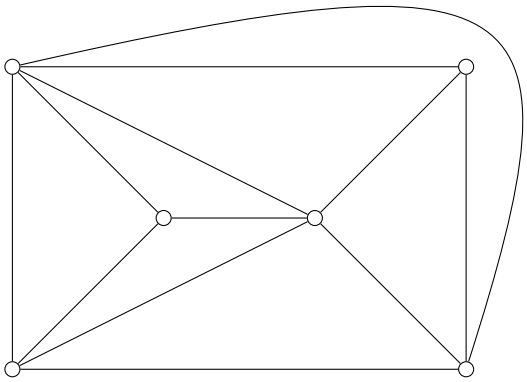


Figure 1.13: The triangulation of the second component G_2 of G

Lastly, we will consider the dual graph of the triangulation of G_2, which is a cubic graph.

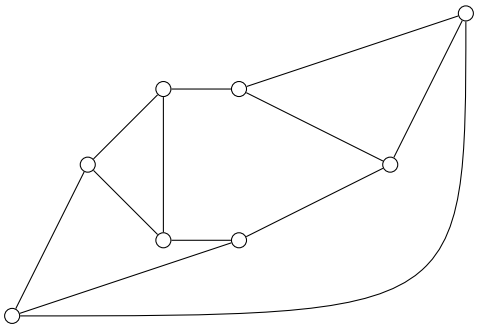


Figure 1.14: The dual graph of the triangulation of G_2

This is where Conjecture 1.21 comes in. If this graph has a Hamiltonian cycle then the Hamiltonian cycle provides a 3 edge-coloring by coloring the edges in the cycle with 2 colors and the remaining edges in a third color. This gives a proper 3-edge-coloring.

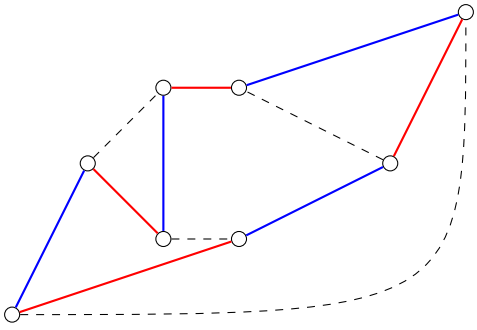
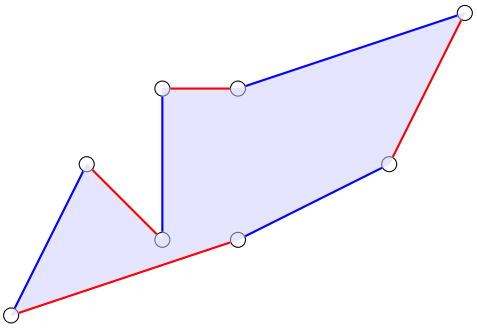


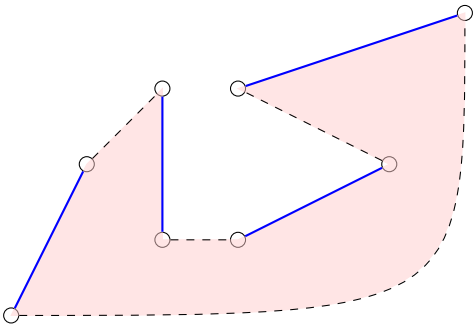
Figure 1.15: Hamiltonian cycle of the dual graph, with edges colored by 2 colors

Now the only thing left is to go from this 3 edge-coloring back to a 4 vertex-coloring. We can do this by considering the 3 edge-coloring but removing edges of one of the colors. Then we get a graph that is 2 face-colorable, this is because we are left with a graph that is 2 edge-colorable, so it must consist of a number of cycle graphs possibly contained within each other. We can color faces contained in an even number of cycles with color 1 and the other faces with color

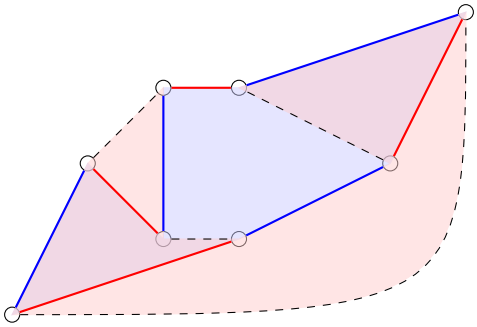
2. Then we do the same with edges of one of the other colors removed and color those faces with color 1 and 3. Finally we can overlay those two graphs to color the triangulated graph by coloring faces with color 1, 2, 3 and 4 such that faces are colored with color 4 if they were colored by both 2 and 3 in our constructions. Now taking the dual again gives a 4 vertex coloring.



Face coloring with first set of edges removed



Face coloring with second set of edges removed



Overlay of the 2 graphs

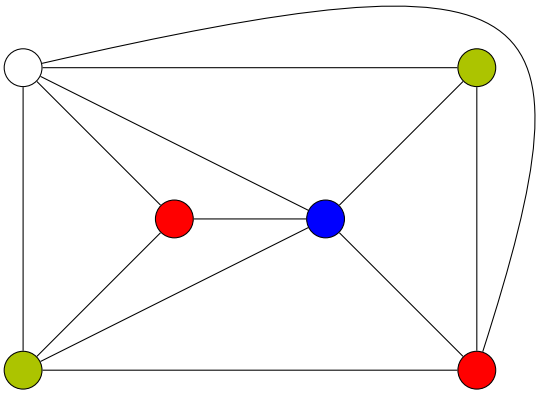


Figure 1.16: 4-vertex-coloring of G_2

This example shows how Conjecture 1.21 implies the 4-color theorem, but not the other way around. Unfortunately, this conjecture turns out to not be true, a counterexample was given in [13] by Tutte.

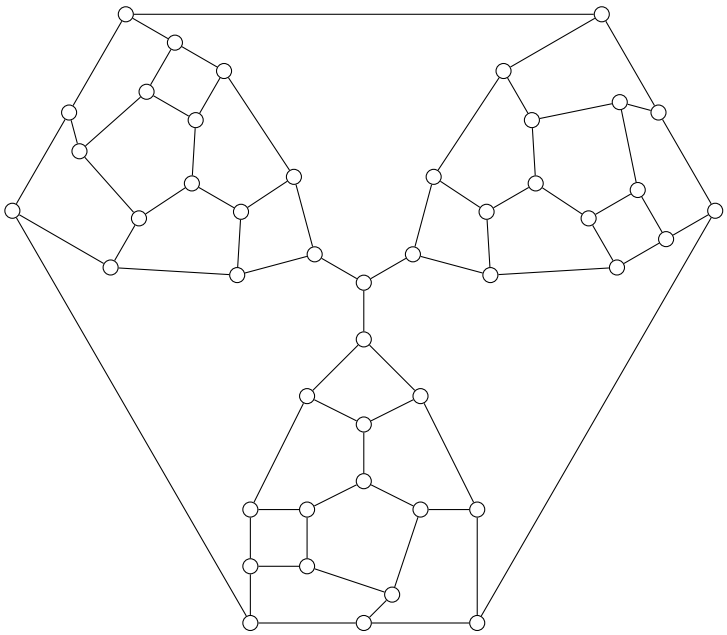


Figure 1.17: Tutte's counterexample

This conjecture is not correct, but it does raise some questions as to what, maybe somewhat weaker, claims we can make about such graphs. In the next section, we will look further into the properties of cubic graphs specifically.

1.4 Cyclically stable subgraphs

In their paper "ensembles cycliquement et graphes cubiques" [9] C. Payan and M. Sakarovitch consider maximal cyclically stable subgraphs of cubic graphs.

For a subset $S \subseteq V$ we define $G_S = (S, E')$, with E' containing all edges xy from E s.t. $x \in S$ and $y \in S$, as the subgraph of G *induced by S.*

Definition 1.25 (cyclically stable)**.** $S \subseteq X$ is cyclically stable if G_S does not contain any cycles. In other words, G_S is a forest.

We denote the maximum cardinality of all cyclically stable subgraphs of G as $s(G)$. This is also called the stability number.

Theorem 1.26. *[9, Theorem 1.] If G is a cubic multigraph with n vertices then we have:*

$$s(G) \leq \frac{3n-2}{4}$$

Proof. Take S to be a maximal subset of V such that G_S is a cyclically stable subgraph. Then G_S is a forest.
We define following values:
r_1 = number of vertices in G_{V-S} with degree 1
a = number of edges in G_S
b = number of edges in G_{V-S}
c = number of edges of G not in G_S and also not in G_{V-S}
p = number of connected components in G_S
Notice that each vertex in G_{V-S} has to be connected with an edge to at least two vertices in

G_S. If this was not the case then we could add this vertex to S without creating a cycle. Since G is a cubic graph this tells us that the maximum degree of any vertex in G_{V-S} is 1. G_{V-S} is a combination of K_1's and K_2's. From this we get that $b = \frac{r_1}{2}$
Since G_S is a forest, we have the formula:

number of edges = number of vertices - number of connected components, Which is equal to the formula $a = s(G) - p$

We also have the following equality $c = s(G) + 2p$. To see this we first look at the case where G_S consists entirely of vertices of degree zero, so $p = s(G)$. We know every vertex has degree 3 in G so in this case there must have been 3 edges removed for every vertex, $c = 3 \cdot s(G)$. The base case holds, now assume the equality holds for a forest with p connected parts. When we connect 2 previously unconnected vertices with an edge, without creating a cycle, the degree of both of these vertices goes up by one so two less edges would have had to be removed from G. Instead of p connected parts we now have $p - 1$ connected parts. In terms of our formula, we replace c by $c - 2$ and p by $p - 1$, so we get $c - 2 = s(G) + 2(p - 1)$. Working this out gives $c-2 = s(G)+2p-2$. Adding 2 to both sides gives us the same equality as before.

The sum $a + b + c$ gives all the edges of G and because G is cubic, this is equal to $\frac{3n}{2}$

$$\frac{3n}{2} = a + b + c = s(G) - p + \frac{r_1}{2} + s(G) + 2p$$

$$2s(G) = \frac{3n}{2} - p - \frac{r_1}{2}$$

$$s(G) = \frac{3n - 2p - r_1}{4}$$

Now we know that $p \geq 1, r_1 \geq 0$, which gives us

$$s(G) \leq \frac{3n - 2}{4}$$

□

Definition 1.27 (Linegraph of G)**.** A Linegraph $L(G)$ of G is a graph where for every edge in G, we define a vertex in $L(G)$. Vertices in $L(G)$ are connected if the corresponding edges are incident (they meet in a vertex).

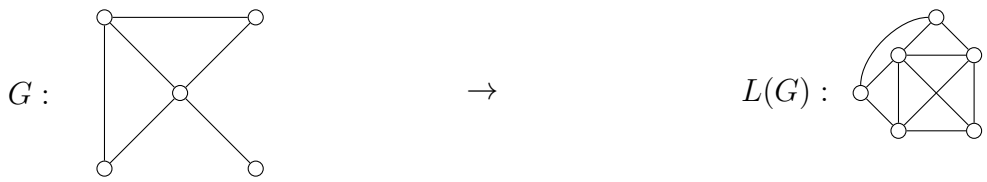


Figure 1.18: Example of a linegraph constructed from a cubic graph.

Definition 1.28 (subdivision of an edge)**.** For a given edge, subdividing it means that we remove the edge and add a new vertex which we connect to the two vertices the original edge was connected to. If we do this for all edges of a graph, we call this a subdivision of that graph.

Figure 1.19: Example of a subdivision of a graph

Theorem 1.29. *If $G = (V, E)$ is a subdivision of a cubic graph, then its linegraph $L(G)$ is a cubic graph with n vertices and we have:*

$$s(L(G)) = \frac{2n}{3}$$

Proof. We will first prove that $L(G)$ is cubic. G is a subdivision of a cubic graph, so every edge in G connects a vertex of degree 3 and a vertex of degree 2. This means it is incident to 2 edges via the first vertex and to 1 other edge via the second vertex. In total it is incident to 3 other edges. In the linegraph, the corresponding vertex is thus connected to 3 other vertices.

Now to find $s(L(G))$ we can look at $|M|$ with $M \subset E$ and M being a maximal set of edges such that these edges form paths in G (no cycles and no vertices of degree 3). If M is such a set, then all the edges in M correspond to vertices in $L(G)$ that form a cyclically stable subgraph of $L(G)$. This gives us $s(L(G)) = |M|$.

We will now prove that every vertex $v \in V$ of degree 3 belongs to exactly 2 edges in M.

It is clear that every vertex of degree 3 belongs to at least one edge of M, otherwise M is not maximal.

Suppose there exists a vertex v in G, of degree 3, that belongs to only one edge $(v, a) \in M$. Let a, b, c be the vertices adjacent to v. $M \cup \{v, b\}$ and $M \cup \{v, c\}$ cannot generate a subgraph which has vertices of degree 3, because we know that a, b, c must all have degree 2 in G. $M \cup \{v, b\}$ and $M \cup \{v, c\}$ therefore generate subgraphs, each containing a cycle. Let C_1 and C_2 be these two cycles. Since $v \in C_1 \cap C_2$, there exists $w \neq v$ such that w is in $C_1 \cap C_2$ and such that w is of degree 3 in the subgraph generated by M. This is a contradiction.

Thus, every vertex of degree 3 in G belongs to exactly 2 edges of M. From this we get:

$$s(L(G)) = |M| = \frac{2|E|}{3} = \frac{2n}{3}$$ □

Definition 1.30 (cyclically c-edge-connected graphs)**.** A graph is cyclically c-edge-connected if we have to remove at least c edges to obtain 2 separate connected components, each containing a cycle. If there exists no edge cut such that we can obtain two such separate components, each containing a cycle, we say $c = 0$.

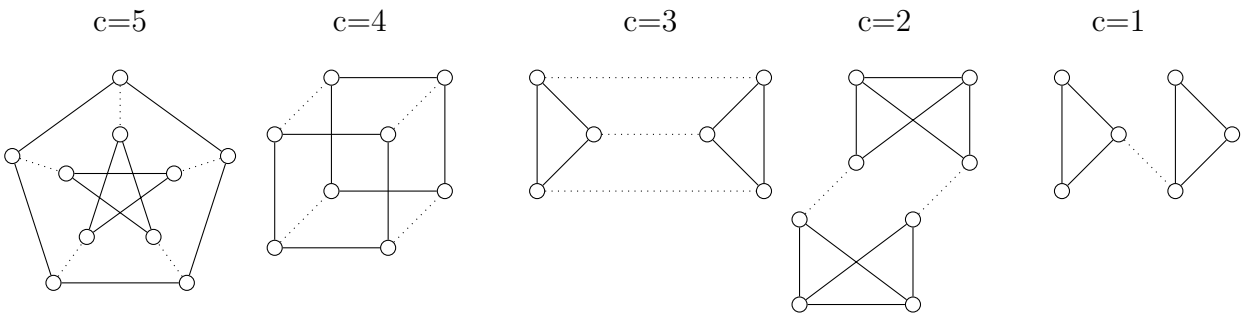


Figure 1.20: Some examples of cyclically c-edge-connected graphs for different values of c

Using two lemmas we will not prove here, we can now look at the proof of the main theorem from this section.

Lemma 1.31. [9, Lemma 4] Every minimal cyclically 4-edge-connected cubic graph contains at least one cycle with length 4.

A graph is minimal for a certain property if it does not contain a proper subgraph that also has that property. Here minimal means that there is no proper subgraph that is also cyclically 4-edge-connected.

Lemma 1.32. [9, Lemma 10] For a minimal cyclically 4-edge-connected graph $G = (V, E)$ with $|V| > 10$ and a set of vertices $W = \{x, y, u, v\}$ for which G_W is a cycle of size 4. We define $A = \{v \in V \mid \exists w \in W, (v, w) \in E\} = \{x', y', u', v'\}$. The graph $\widehat{G} = (V - W, E(V - W) \cup \{(x, u), (y, v)\}$ is a cyclically 4-edge-connected cubic graph.

Theorem 1.33. *If G is a cubic cyclically 4-edge-connected graph with n vertices we have:*

$$s(G) = \left\lfloor \frac{3n-2}{4} \right\rfloor$$

Proof. Suppose the property is false, and let $G = (V, E)$ be a minimal cubic cyclically 4-edge-connected graph such that $s(G) < \left\lfloor \frac{3n-2}{4} \right\rfloor$, with n being the minimum number of vertices for which this property holds. Let $A = \{x, y, u, v\} \subset V$ such that G_A is a cycle of length 4. This is possible because of Lemma 1.31. With $A' = \{v \in V \mid \exists w \in A, (v, w) \in E\} = \{x', y', u', v'\}$ we use Lemma 1.32 to construct the cyclically 4-edge-connected cubic graph $\widehat{G} = (V - A, E(V - A) \cup \{(x', u'), (y', v')\}$. The number of vertices in $\widehat{G}$ is $n-4$ and consequently, if $\widehat{S}$ is a maximal cyclically stable set in $\widehat{G}$, we have:

$$|\hat{S}| \geq \left\lfloor \frac{3(n-4)-2}{4} \right\rfloor = \left\lfloor \frac{3n-2}{4} \right\rfloor - 3$$

Now consider the connected components of $G_{\widehat{S}}$, the edges (x', u') and (y', v') are in $\widehat{G}$, but not in G. Because of this, we observe the following:

1. x' and u' (respectively y' and v') do not belong to the same connected component of $G_{\widehat{S}}$.

2. If x' and y' (respectively u' and v') belong to the same connected component of $G_{\widehat{S}}$, then u' and v' (respectively x' and y') do not belong to the same connected component.

Now, assume that x' and y' belong to the same connected component of $G_{\widehat{S}}$. Then $S = \widehat{S} \cup \{y, u, v\}$ (where y, u, v are the vertices in A adjacent to y', u', v', respectively) is cyclically stable in G. This implies:

$$|S| = |\widehat{S}| + 3$$

which leads to a contradiction. □

Chapter 2

AI-partitions

In this chapter we will use the concept of maximal cyclically stable subgraphs seen in the previous chapter and figure out the conditions a cubic graph needs to have for this subgraph to have a size equal to the upper bound.

2.1 Cubic graphs with stable decycling partition

A cubic graph of order n has a stable decycling partition if we have a vertex-partition $\{A, I\}$ with $A \cup I = V$ of the graph such that the following holds:

if $n \equiv 2 \ (mod\ 4)$	*then A induces a tree and I is an independent set*
if $n \equiv 0 \ (mod\ 4)$	*then A induces a forest of 2 trees and I is an independent set*
	or A induces a tree and I is a near-independent set

We define an set of vertices I in a graph G to be independent if G_I contains no edges and to be near-independent if G_I contains exactly one edge. We also call such a partition a stable decycling partition or an AI-partition. If such a partition exists, then the size of A reaches the upper bound for cyclically stable subgraphs $|A| = \left\lfloor \frac{3n-2}{4} \right\rfloor$ and thus also $|I| = \left\lceil \frac{n+2}{4} \right\rceil$.
R. Nedela, M. Seifrtová and M. Škoviera state in [8, Theorem 1.2] that Theorem 1.33 can be reformulated using this concept.

Theorem 2.1. *Every cyclically 4-edge-connected cubic graph has an AI-partition.*

This is a nice theorem, but is it as strong as possible, or can we find weaker conditions that also work? That is what we will investigate in this section.

1-connected cubic graphs

We first consider the case where a cubic graph G is 1-connected, or simply just called connected.

If we replace cyclically 4-edge-connected by 1-connected in the theorem then We can very quickly come up with counterexamples for this claim.

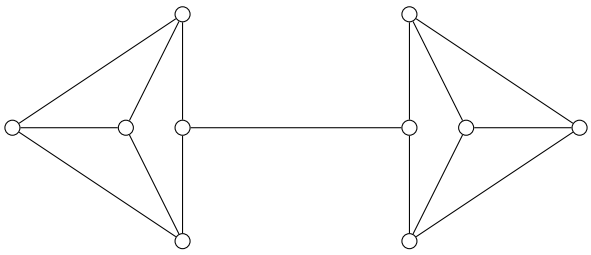


Figure 2.1: 1-connected counterexample with $|V| = 10 \equiv 2 \ (mod\ 4)$.

Figure 2.1 is constructed from 2 gadgets connected by an edge. This does not have an AI-partition because clearly the two middle vertices have to be in A, but then if we put only one vertex on each side in I, we still have cycles. If we were to put another vertex in I, this would disconnect the tree.

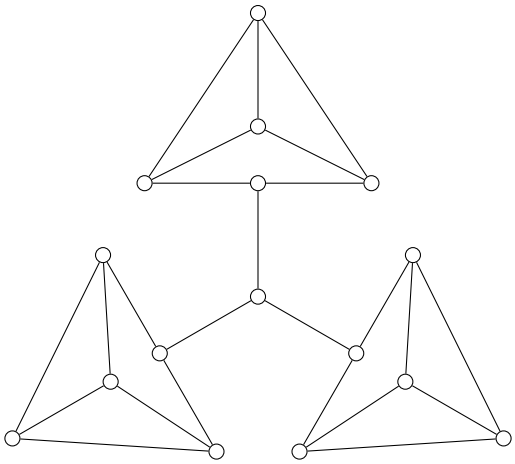


Figure 2.2: 1-connected counterexample with $|V| = 16 \equiv 0 \ (mod\ 4)$.

For Figure 2.2, we have a similar argument to that in Figure 2.1. The middle vertex has to be in A, otherwise we would have 3 trees. Now for each gadget, we have to put two vertices into I to get rid of all the cycles, but this would disconnect the graph into 4 trees if we choose an independent set of vertices and 3 trees if we choose a near-independent set of vertices.

In both these cases, we use a modified K_4 graph where we subdivide one edge as a gadget to construct the bigger graphs.

We can easily use this to construct an infinite family of graphs without AI-partition by subdividing another edge of K_4 and combining them in Figure 2.3:

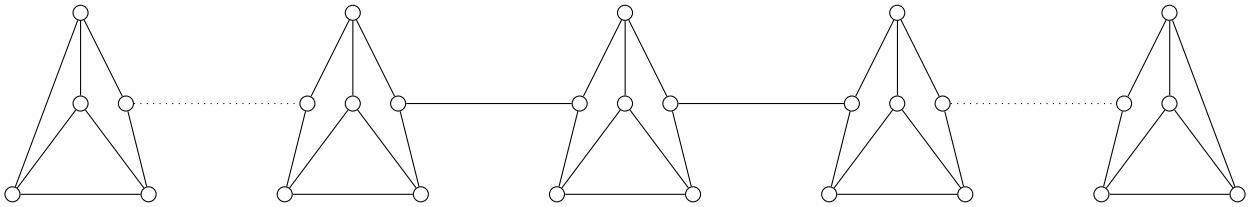


Figure 2.3: Infinite family of 1-connected counterexamples with order $6k - 2$, where k is the number of gadgets.

These graphs cannot have AI-partitions because we have to put 2 vertices from every gadget into I, but this always disconnects the graph into as many components as there are gadgets when we choose an independent set of vertices and one less if we choose a near-independent set.

2-connected cubic graphs

We can use the construction for 1-connected graphs to get a 2-connected graph without AI-partition. We do this by taking 3 of these gadgets and connecting them to each other on a cycle.

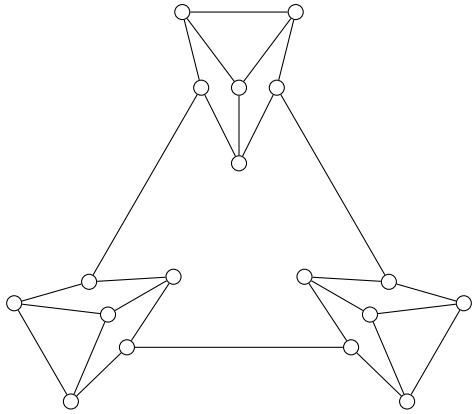


Figure 2.4: 2-connected counterexample with $|V| = 18$

Figure 2.4 does not have an AI-partition for the same reasons as before. This also gives an infinite family if we take any number of these gadgets higher than 3. This then has order $6k$, with k the number of gadgets.

The graph we have constructed is a minimal counterexample for simple graphs. However, if we allow multigraphs without loops we can find an even smaller example.

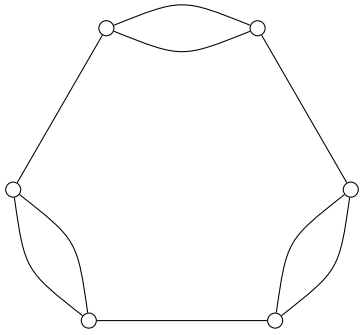


Figure 2.5: 2-connected multigraph minimal counterexample with $|V| = 6$

Here we have to put one vertex in I for every gadget of two vertices, but this always disconnects the graph.

In general, a 2-connected cubic graph looks like this, with $k \in \mathbb{N} \cup \{0\}$:

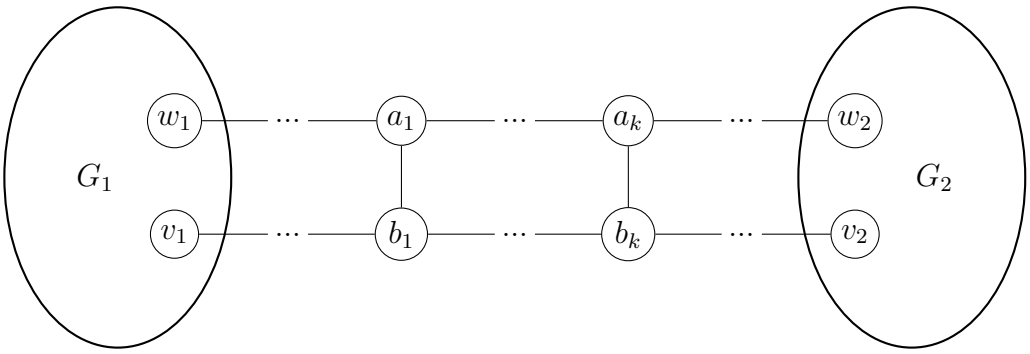
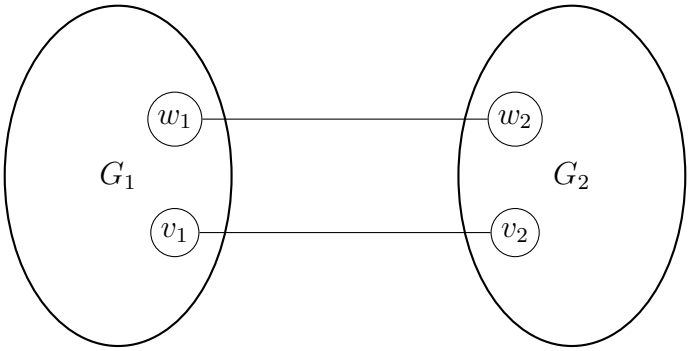


Figure 2.6: General 2-connected cubic graph

Here G_1 and G_2 are cubic graphs (who are at least 2-connected) where the edge (v_1, w_1), resp. (v_2, w_2) is removed. There are an even number (possibly zero) of vertices connected in a square pattern connecting them.

Proposition 2.2. *If we have a 2-connected graph as shown above with k even, G_1 does not have a partition and G_2 has order 0 (mod 4) then the whole graph also does not have an AI-partition.*

Proof. First consider the case with $k = 0$



We assume G_1 does not have an AI-partition. If the order of G_1 is 2 (*mod* 4), for any partition (A_1, I_1) of G_1, A_1 must induce a component with at least one cycle. If the order of G_1 is 0 (*mod* 4), for any partition (A_1, I_1) of G_1, A_1 must induce a tree and a component with at least one cycle or A_1 must induce a component with at least one cycle and I_1 is a near-independent set.

A_1 contains more than 1 cycle: In this case the whole graph can never have an AI-partition, because removing a single edge can never break all the cycles.

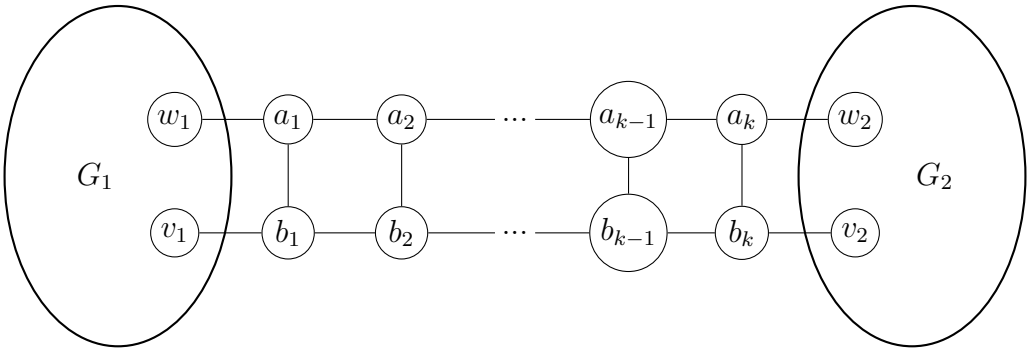
A_1 contains only 1 cycle: If we can break this cycle by disconnecting v_1 and v_2 then we can possibly have an AI-partition dependent on the structure of G_2.

If G_2 does not have an AI-partition, then for any partition (A_2, I_2) of G_2, A_2 must contain at least one cycle. There are two cases, either every partition A_2 has at least two cycles or there is a partition where A_2 has only one cycle. In the first case the whole graph can never have an AI-partition. In the second case it is possible that the edge we removed from G_2 breaks this cycle, but then w_1 and w_2 are connected in A_2 so if we want to find a partition of the whole graph we know that v_1 and v_2 are connected through A_2. We therefore did not break the cycle in G_1, so the whole graph does not have an AI-partition.

If G_2 has an AI-partition, then A_2 either consists of 2 trees or I_2 is a near-independent set. For

the first case, w_1 and w_2 were connected in G_2 so either they both are part of the same tree in A_2 or one of them is in I_2. In either case we have that one of the trees in A_2 is not connected to G_1 in any AI-partition. Now any partition (A,I) for the whole graph, A contains two trees when the order of G_1 is 2 (*mod* 4). A contains either three trees or two trees, but then I is a near-independent set when order of G_1 is 0 (*mod* 4).
For the second case, if both w_1 and w_2 in I_2 then for any AI-partition A consists of 2 trees when the order of G_1 is 2 (*mod* 4). A consists of three trees or A consists of two trees and I is near-independent when the order of G_1 is 0 (*mod* 4). Otherwise, I has 2 connected vertices when the order of G_1 is 2 (*mod* 4). I has more than two connected vertices or I has two connected vertices and A contains two trees when the order of G_1 is 0 (*mod* 4).

This proves the proposition for $k = 0$. Now consider the case with k an even number.



For every disjoint C_4 we add between the gadgets, we can only put one of the 4 vertices into I. If we do this without breaking the graph into 2 disconnected components, then either a_1 and b_1, or a_k and b_k must be both in A. This means we cannot break all the cycles in both of the gadgets at the same time. There cannot be an AI-partition in those cases. □

Example 2.3. We can use this to construct counterexamples with certain properties. For example, we can use the 2-connected counterexample we constructed earlier to construct a triangle-free graph by combining this with three Q_3's:

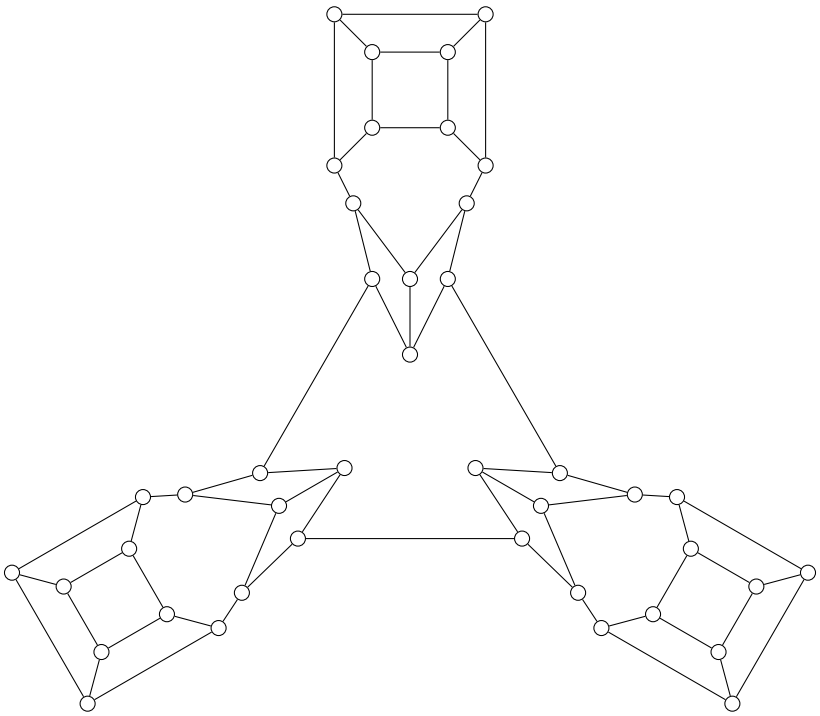


Figure 2.7: Triangle-free 2-connected counterexample with order 42

3-connected cubic graphs

In this subsection, we will only consider simple graphs. This is simply because any cubic multigraph that has at least one double edge can be at most 2-connected.

We can use the method of constructing a graph with gadgets to find a 3-connected graph without AI-partition.
We start with a cubic graph of order $0 \pmod 4$ *with an AI-partition and remove one vertex from it, this leaves 3 vertices with only 2 edges. This is our gadget. Then we take a 3-connected graph and replace the vertices with our gadgets. The resulting graph cannot have an AI-partition.*

Example 2.4. Remove one vertex from Q_3 to get a 3-connected gadget.

We can construct Figure 2.8 by placing this gadget on the vertices of another Q_3.

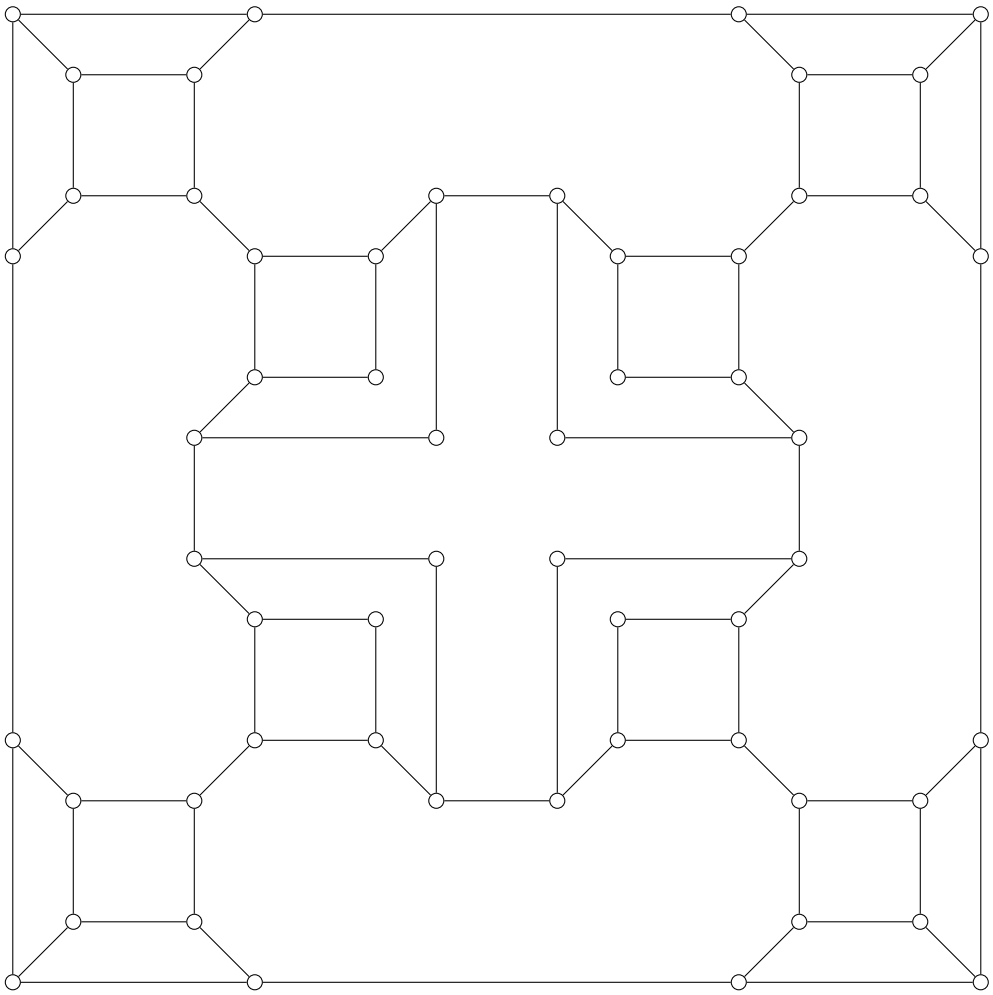


Figure 2.8: 3-connected counterexample

This graph has order 56 and does not have an AI-partition. We can show this by considering that if we want to construct an AI-partition for this graph then we have the two cases:

- I is an independent set

For any of the gadgets we must disconnect them into 2 trees or put one of the connecting vertices into I. The latter is equivalent to removing an edge in the underlying graph for every vertex. In this case the underlying graph was Q_3 so removing 8 edges from it leaves us with just 4 edges left, this means we can have no less than 4 trees in A.

- I is a near-independent set

If we allow this then we can leave one of the gadgets connected. Then we only have to remove 7 edges from the underlying graph, so we are left with 5 edges. The least amount of trees in A we can get with this is 3.

In general if we have a cubic graph of order $n_g \equiv 0 \ (mod\ 4)$, with an AI-partition, then we can make this into a gadget by removing one vertex.

We place these gadgets on the vertices of an underlying cubic graph of order n. If we want to construct an AI-partition of this graph, we need to look at the underlying graph with an edge removed for every vertex. A cubic graph with n vertices has $\frac{3n}{2}$ edges, so we are left with

$\frac{3n}{2} - n = \frac{n}{2}$ *edges. The least amount of trees we can have with* n *vertices and* $\frac{n}{2}$ *edges is* $\frac{n}{2}$. *We can conclude that for* $n > 4$ *we cannot have a partition. The constructed graph has order* $(n_g - 1)n$

We can also construct graphs by using multiple different gadgets. A graph of order $2 \pmod 4$ *has a partition in a tree and an independent set, so if we remove one of the vertices that lies in* I *from the graph to form a gadget then this gadget can still be partitioned into a tree and an independent set. This means that if we look at the partition of a graph with these gadgets, the gadgets remain connected. In terms of the underlying graph, we do not remove an edge for the vertex corresponding to this gadget. In general if we have an underlying cubic graph of order* n *with* k *gadgets of order* $0 \pmod 4$ *and* $n-k$ *gadgets of order* $2 \pmod 4$. *Constructing a partition of the graph is equivalent to removing* k *edges from the underlying graph. When we have fewer than* $n-2$ *edges left, we have at least 3 trees in* A. *This happens when* $k > \frac{3n}{2} - (n-2) = \frac{n+4}{2}$. *In conclusion, we get the following proposition.*

Proposition 2.5. *If we have an underlying graph of order* n, *with* n *at least* 6, *and at least* $\frac{n+6}{2}$ *gadgets of order* $0 \pmod 4$ *placed on its vertices then the resulting graph cannot have an AI-partition.*

Using K_4 *with one vertex removed as all the gadgets, we can generate the smallest possible examples Figure 2.9 and Figure 2.10 using this method.*

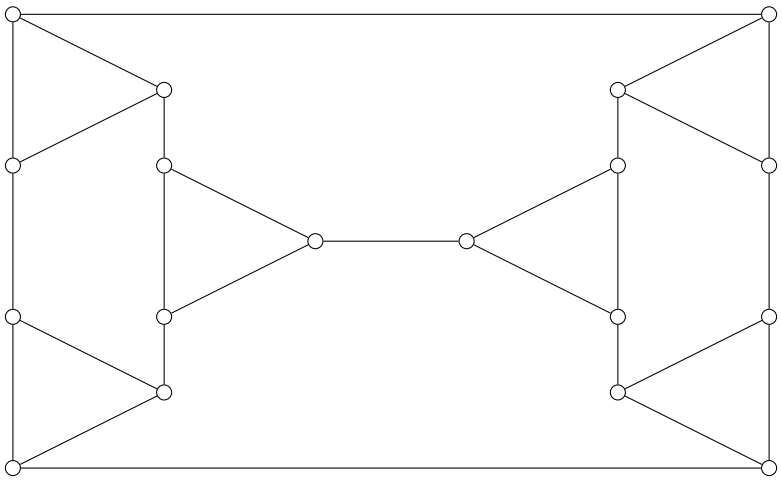


Figure 2.9: Smallest 3-connected counterexample with $|V| = 18 \equiv 2 \pmod 4$

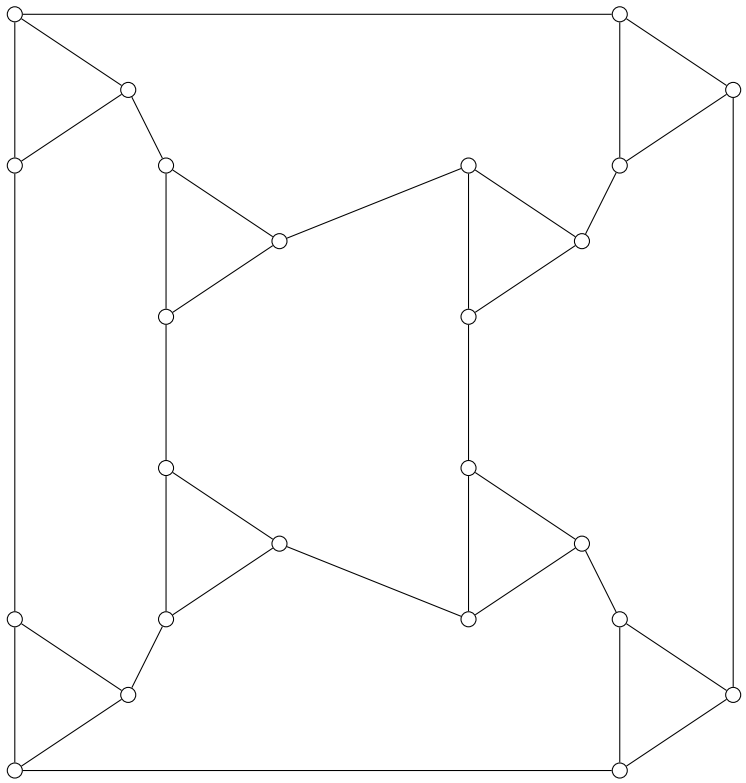


Figure 2.10: Smallest 3-connected counterexample with $|V| = 24 \equiv 0 \ (mod\ 4)$

Here we used underlying graphs Π_3 *for Figure 2.9 and* Q_3 *for Figure 2.10. With this construction we can find counterexamples with all kinds of properties we want. For example, using triangle-free gadgets gives us a triangle-free graph without AI-partition. Using gadgets and an underlying graph of high girth will also give us a graph with high girth without AI-partition...*

Clearly 3-connected is not a sufficient condition, even with some additional assumptions like planar, triangle-free, ...

Essentially 4-connected

Definition 2.6. A graph is essentially 4-connected if it is connected and if we remove any set of vertices with a size less than 4, the resulting graph is connected or has exactly two components one of which has exactly one vertex.

Example 2.7. K_4 is essentially 4-connected. This is what we call an edge-case of the definition. Removing 3 or less vertices leaves the graph connected and thus it complies with the definition, but when we remove 4 vertices there are no vertices left. K_4 is not cyclically 4-edge-connected.

The cubic multigraph with 4 vertices and two double edges is 2-connected and thus not essentially 4-connected. It is also not cyclically 4-edge-connected.

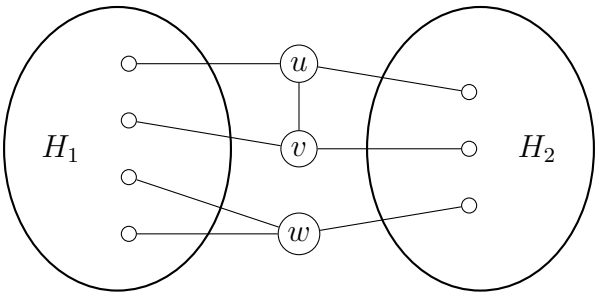
We have concluded that 3-connected is not enough for the theorem to hold, but maybe essentially 4-connected does work. This is actually the case, but the reason for this is that for cubic graphs the notions of cyclically 4-edge connected and essentially 4-connected almost coincide.

Lemma 2.8. [16, Lemma 1] A 3-connected cubic graph of order > 6 is essentially 4-connected if and only if it is cyclically 4-edge-connected.

Proof. Assume $G = (V, E)$ is essentially 4-connected, but not cyclically 4-edge-connected. Then we can remove a set M of 3 edges and split the graph into two components $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$. Each of these components contains a cycle. Since $|V| > 6$ and G is cubic, we have that $|V| \geq 8$. From this we can conclude that at least one of the components, say H_1, contains at least four vertices. If $|V_1| = 4$, then the sum of the degrees of its vertices would be equal to $3 \cdot V_1 - 3 = 9$, which is impossible because this sum cannot be odd. So $|V_1| \geq 5$. Take the endpoints of the edges in M lying in H_1. By removing them from G, we obtain a non-trivial k-cut with $k \leq 3$, a contradiction.

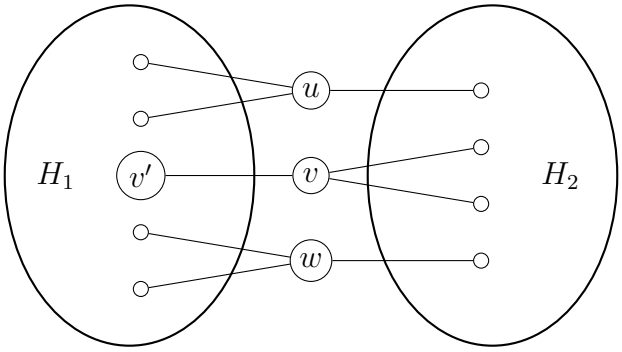
Now suppose that $G = (V, E)$ is cyclically 4-edge-connected. We first claim that we cannot remove a set M of 3 edges such that the resulting graph has two components H_1, H_2 both non-isomorphic to K_1. Otherwise, let $\{u_i, v_i, w_i\}$ be the endpoints of the edges in M lying in the component H_i for $i \in \{1, 2\}$. Since G is 3-connected, by Menger's Theorem there are three vertex-disjoint paths in G connecting any two vertices of $\{u_1, v_1, w_1\}$. Note that two of these have to lie completely in H_1, thus this component must contain a cycle. The same holds for H_2, which must contain a cycle. This is a contradiction.

Let $X = \{u, v, w\}$ be a set of 3 vertices such that removing them from G disconnects the graph into components H_1, H_2. Then we claim that X is an independent set of G. Otherwise, assume there is an edge connecting u and v. Then there cannot be an edge connecting w and u or connecting w and v.



The vertex w is connected to one vertex in one component and to two vertices in the other component, assume it is connected to a single vertex in H_2. Then removing the edge between w and this vertex and removing the two edges connecting u and v to H_1 gives 2 components both non-isomorphic to K_1, a contradiction to our previous statement.

Thus we have that every vertex from X is connected to a vertex in one component and to two in the other component. In total there are 9 edges between vertices in X and vertices in H_1 or H_2. If there are ≥ 4 edges connecting X and H_1 and ≥ 4 edges connecting X and H_2 then, without loss of generality, assume there are 5 edges connecting X and H_1 and v is connected to a single vertex v' in H_1.



Consider the set of vertices $X' = \{u, v', w\}$, removing it from G would result in two non-trivial components such that exactly one of them has 3 edges connecting it to vertices in X'. This is a contradiction. Thus we must have , by symmetry, 6 edges connecting X and H_1 and 3 edges connecting X and H_2. This means that H_2 must consist of a single vertex. Thus G is essentially 4-connected. □

Now all that is left to check is if essentially 4-connected cubic graphs of order 6 all have AI-partitions. For this we only need to consider simple graphs, because multigraphs with at least one double edge are only at most 2-connected.

We have 2 possible simple cubic graphs for order 6, Π_3 *and* $K_{3,3}$.

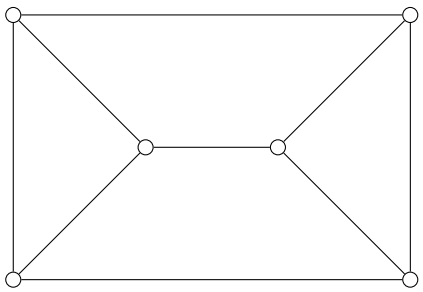


Figure 2.11: Π_3

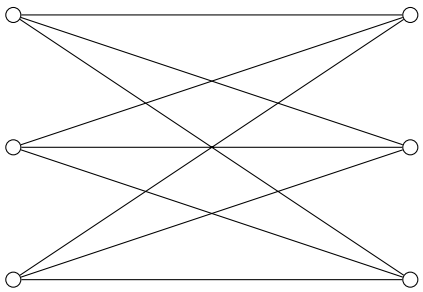
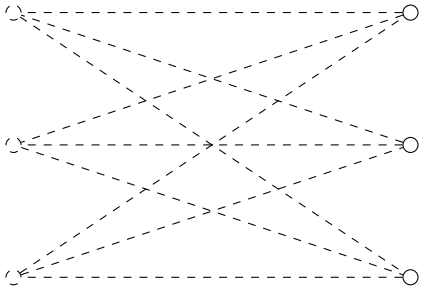


Figure 2.12: $K_{3,3}$

The first graph is the only graph of the two that is essentially 4-connected. The second one has a vertex cut of size 3 which cuts the graph in three disjoint vertices:



That does not really matter for our purposes, because they both have an AI-partition. We can conclude that the following holds.

Theorem 2.9. *Every essentially 4-connected cubic graph has an AI-partition.*

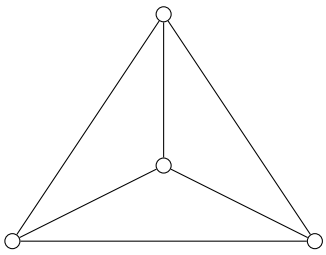
Being essentially 4-connected is a slightly weaker claim as being cyclically 4-edge-connected because of these two graphs of order 6. With this we have improved Theorem 2.1.

Cubic linegraphs

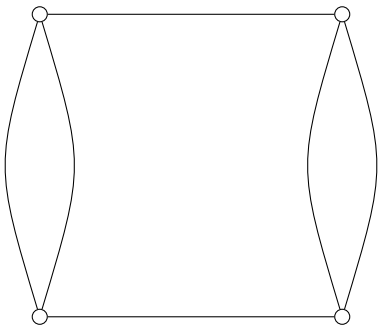
We end this subsection with a small remark about linegraphs. As we saw in Theorem 1.29, if a cubic graph G of order n is a linegraph of a subdivided cubic graph, then $s(G) = \frac{2n}{3}$. This means that such a graph can only have an AI-partition if we have the equality

$$\frac{2n}{3} = s(G) = \left\lfloor \frac{3n-2}{4} \right\rfloor$$

This only holds if $n = 6$ or $n = 12$. The order of a linegraph is equal to the number of edges in the graph where we took the linegraph of, so for $n = 6$ we construct a linegraph from a subdivided cubic graph with 3 edges before subdivision. For $n = 12$, we start with a cubic graph with 6 edges which we then subdivide. The cubic graphs with 6 or 3 edges are the following:



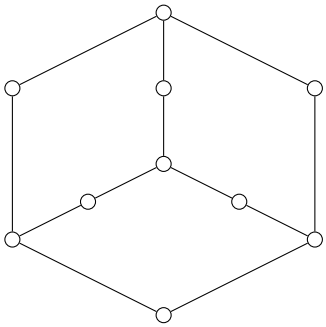
6 edges

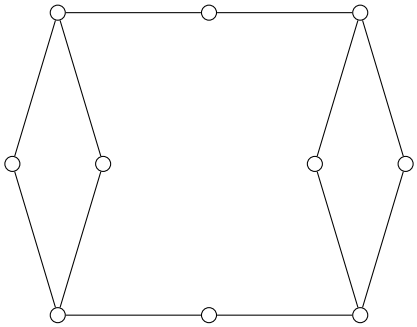


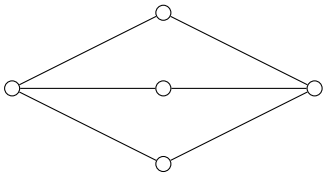
6 edges

3 edges

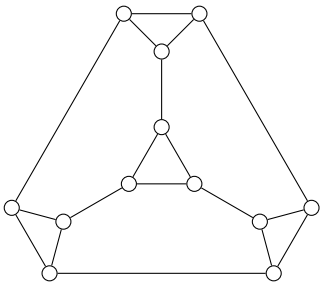
We then subdivide them:

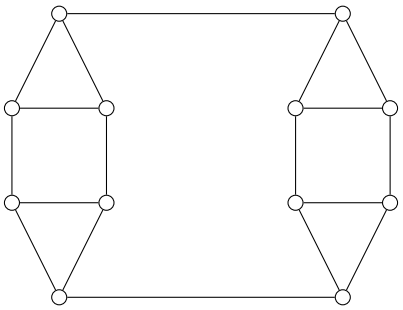


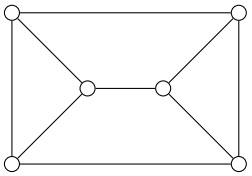




Finally, taking the linegraphs of these subdivided graphs gives us these:







Every other cubic graph that is a linegraph of a subdivided cubic graph cannot have an AI-partition.

2.2 Upper-embeddability

In this chapter we use definitions from M. Jungerman. [5]

Definition 2.10. A graph has a 2-cell embedding ϵ in a surface S if we can draw the graph on the surface such that every face is homeomorphic to a disc.

For an easy example of this, we take K_4, this can be embedded into a torus as follows.

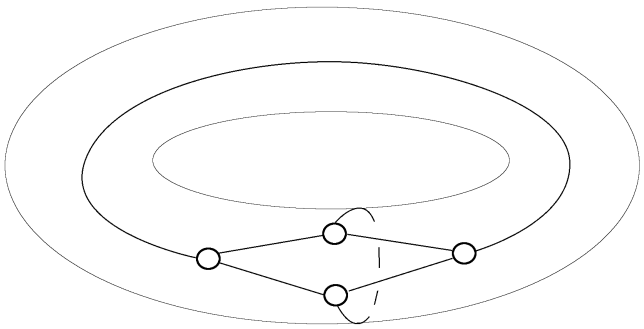


Figure 2.13: 2-cell embedding of K_4 on a torus

The face at the front of the torus is clearly homeomorphic to a disc and this is also true for the other face, although a bit harder to see directly. To make it somewhat easier to see, we can look at the graph on the 2-dimensional representation of the torus:

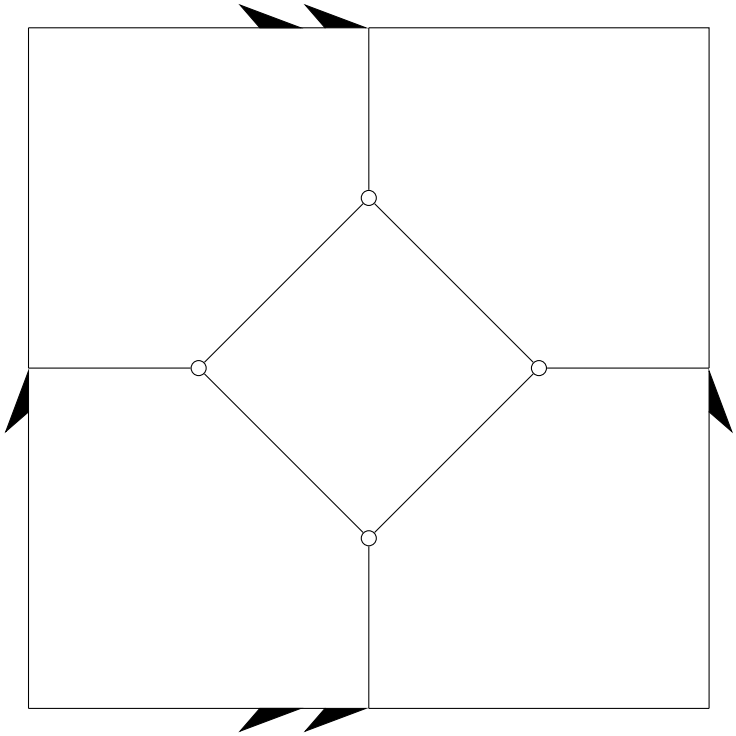


Figure 2.14: Embedding of K_4 on 2-dimensional representation of a torus

We have the following formula for the genus $\gamma(S)$ of S:

$$\gamma(S) = 1 - \frac{\chi(G)}{2}$$

Here $\chi(G) = |V| - |E| + |F(\epsilon)|$ is the Euler characteristic of the graph $G = (V, E)$ and $F(\epsilon)$ is the number of faces of G with embedding ϵ in S. We can combine this to get:

$$\gamma(S) = \frac{|E| - |V| - |F(\epsilon)| + 2}{2}$$

We define the maximum genus of a graph $\gamma_M(G)$ *as*

$$\gamma_M(G) = \max\{\gamma(S) \mid \exists \text{ a 2-cell embedding of } G \text{ in } S\}$$

We can see that a 2-cell embedding that minimizes $F(\epsilon)$ *will maximize the genus* $\gamma(S)$*. From this we get the inequality*

$$\gamma_M(G) \leq \left\lceil \frac{|E| - |V| + 1}{2} \right\rceil = \left\lceil \frac{\beta(G)}{2} \right\rceil$$

$\beta(G) = |E| - |V| + 1$ *is the first Betti number of* G*.*

Definition 2.11. A graph is upper-embeddable if the equality $\gamma_M(G) = \left\lceil \frac{\beta(G)}{2} \right\rceil$ holds.

Furthermore we can see that if we have an embedding ϵ *with maximal genus and if* $\beta(G)$ *is even, then the equality holds iff* $F(\epsilon) = 1$*. If* $\beta(G)$ *is odd then the equality holds iff* $F(\epsilon) = 2$*.*

To give an example, the graph Q_3 *is upper-embeddable. Its first Betti number is* $12 - 8 + 1 = 5$*, which is odd, so we must find an embedding with 2 faces to conclude that it is in fact upper-embeddable. However if we embed this graph in a torus with one hole, we have 4 faces.*

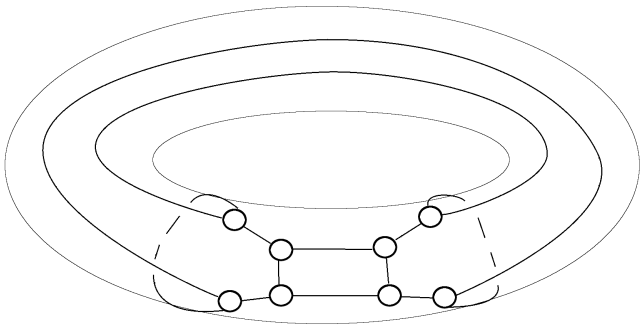


Figure 2.15: Embedding of Q_3 on a torus

But if we embed it in a torus with 2 holes we do in fact get 2 faces.

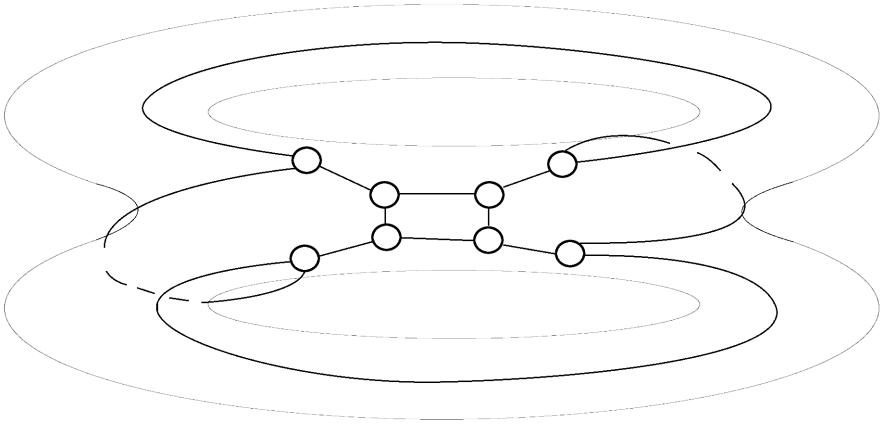


Figure 2.16: Embedding of Q_3 on a 2-torus

$$\gamma_M(Q_3) = 2 = \left\lceil \frac{\beta(Q_3)}{2} \right\rceil$$

If we draw this graph on a 2-dimensional representation of the 2-torus, it looks like this:

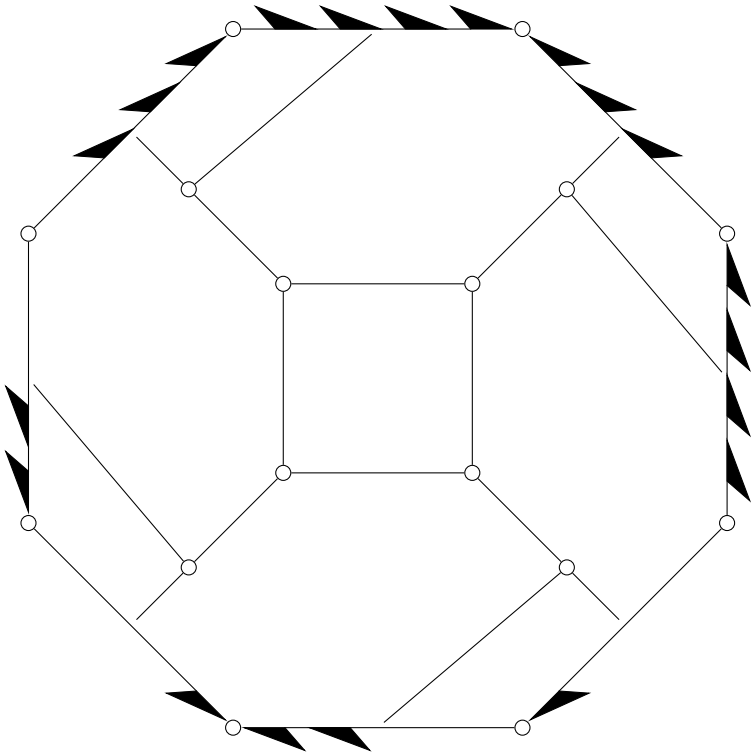
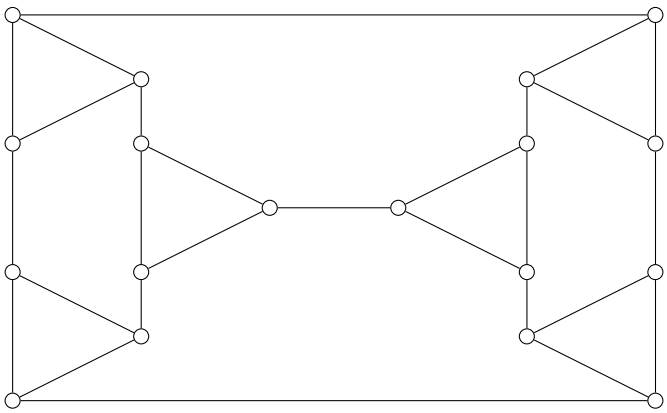


Figure 2.17: Embedding of Q_3 on a 2-dimensional representation of a 2-torus

splitting trees

Not every cubic graph is upper-embeddable, for example this graph:

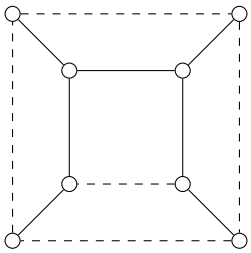


To check that this graph G is not upper-embeddable, we first calculate $\left\lceil \frac{\beta(G)}{2} \right\rceil = \frac{10}{2}$. We see that β is even and if we want G to be upper-embeddable the equality $\gamma_M(S) = 5$ must hold. This tells us that G is upper embeddable if we can find a 2-cell embedding with one face on a surface of genus 5. As we can see, checking if a graph is upper-embeddable can become difficult if we need to check this for a very high genus, this is why we can define the following

Definition 2.12. A splitting tree of a graph G is a tree T which is a spanning tree and the cotree $G - T$ has no more than one component with an odd number of edges.

We can check that if T splits G, $G - T$ has only edge even components iff $\beta(G)$ is even.

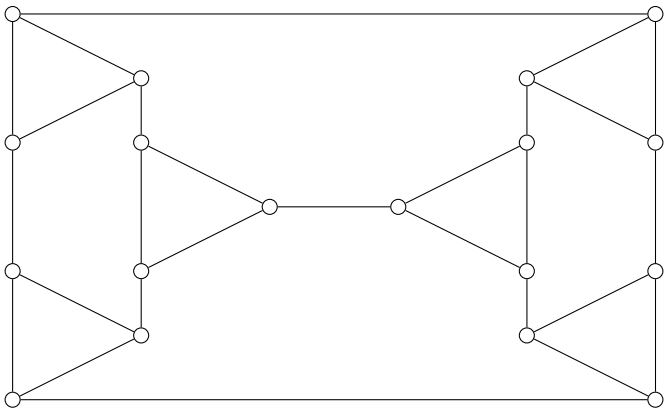
Example 2.13. A possible splitting tree of Q_3 is the following:



Here the cotree has an odd component with 1 edge.

Theorem 2.14. *[5, Theorem 2]*
G is upper-embeddable $\iff$ G has a splitting tree.

Using this we can now come back to our previous example and prove that this graph is not upper-embeddable:



Proof. We want to find a splitting tree of G. The graph has 18 vertices and 27 edges, so T must have 17 edges to be a tree. $G - T$ must therefore have $27 - 17 = 10$ edges, but if we look at our graph we see that there are 6 triangles. For each triangle we must put at least one edge in $G - T$ to break the cycles, but to get a splitting tree we must have that all components of $G - T$ have an even number of edges. From this we get that we must put 2 edges in $G - T$ for each triangle. Because we have 6 triangles, this means that $G - T$ must have $2 \cdot 6 = 12$ edges. This is a contradiction because $G - T$ had 10 edges. This graph is not upper-embeddable. □

Theorem 2.15. *G has a splitting tree $\iff$ G has an AI-partition.*

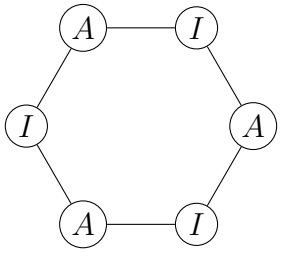
Proof. We will consider two cases separately

- G has order 2 (*mod* 4)

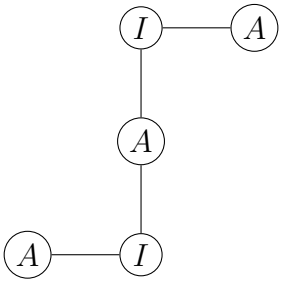
In this case we have $\beta(G)$ is even, so a splitting tree of G has only edge-even components in the cotree.

First assume that we have a splitting tree T of G, then we have only cotree components with an even number of edges. We can construct an independent set by taking vertices from the cotree components. Vertices in these components cannot have degree 3, because then this

vertex would not be in T and thus T would not be a splitting tree. The components are either paths or cycles. If there is a vertex with degree 1 in the component, take the vertex connected to it. This vertex then must have degree 2. If it has degree 1, then the component is edge-odd. Now take this vertex and every other vertex in the component that is at an even distance from it and put them in I. If there are no vertices with degree 1, then we are in a cycle with an even number of vertices, so we can start with any vertex in the component and put it and all vertices an even distance from it in I.



AI-partition for a component that is a cycle



AI-partition for a component that is a path

If we do this for every component, I is clearly an independent set and $A = G - I$ is a tree. This is the case because every vertex in I had degree 1 in T. Thus two of them cannot be connected in G and removing them from T still gives a tree, because they were all leaves.

If we start from an AI-partition, then we can construct a splitting tree by first putting every vertex of G and every edge of A in T and then by connecting each of the vertices of I with one edge to T, which is then still a tree. We can do this by choosing one of the 3 edges with which a vertex was originally connected in G. This leaves 2 adjacent edges in the cotree and because I was an independent set, 2 different vertices in I leave different edges in the cotree. From this we can conclude that the cotree has only edge-even components.

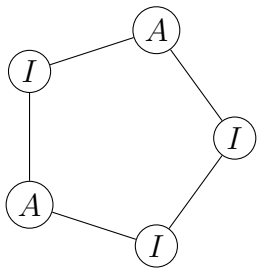
- G has order 0 (*mod* 4)

Here we have 1 edge-odd component in the cotree of a splitting tree.

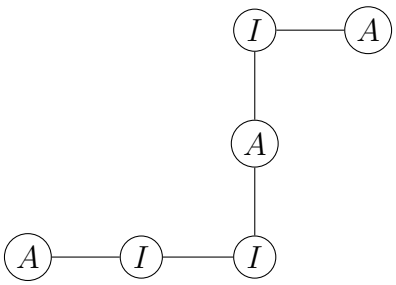
Assume that we have a splitting tree T of G. For all edge-even components we can do the same as before. For the edge-odd component we have two cases: either the component has one edge, or it has at least 3 edges.

If it has one edge, we take one of the 2 vertices and put it in I, this vertex had degree 2 in T so it will disconnect the tree. This gives us an AI-partition with A a 2-forest and I independent. However, it can also be a partition with A a tree and I near-independent if this vertex was next to a vertex from an even component which we already put in I.

If the edge-odd component has at least 3 edges and it is a path, we take a vertex which has degree 2 in the component and also the vertex next to it with degree 2. Then take all vertices with an even distance from the second vertex, except the one with degree 1 and put them in I. If we do this, then I is no longer an independent set, but since we only removed vertices with degree 1 from T, it remains a tree. This gives us an AI-partition with A a tree and I near-independent. If the component is a cycle, we take any vertex and go around the cycle in steps of 2 vertices. Put them all in I until we come to an adjacent vertex to the one we started from, this is the last one we put in I.



AI-partition when the component is a cycle

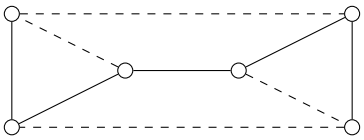


AI-partition when the component is a path

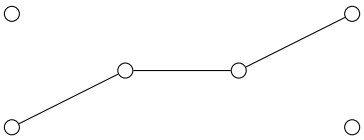
If we start from an AI-partition, we know that there are two cases. A is a 2-forest and I is independent: In this case we have at least 1 vertex in I that was originally connected to both of the trees in A. If we connect this vertex back we add 2 edges, so this leaves one edge in the cotree. Now we can just do the same as in the 2 (mod 4) case and connect every other vertex in I with one edge to A to get a splitting tree with one odd-edge component.

A is a tree and I is near-independent: In this case we can just connect all the vertices in I to A with one edge, this gives a splitting tree with one edge-odd component of at least 3 edges. □

Example 2.16. splitting tree to AI-partition for Π_3 with order $G \equiv 2$ (mod 4)

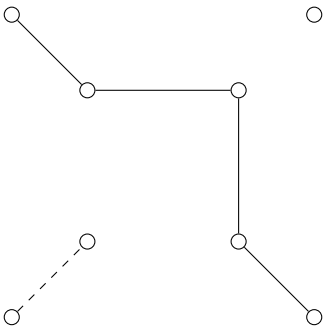


Splitting tree

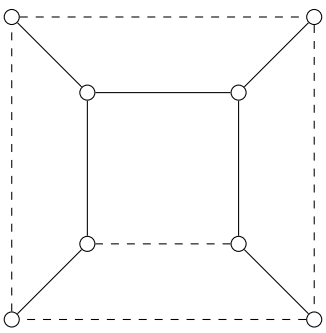


AI-partition we can construct from the splitting tree

Example 2.17. AI-partition to two possible splitting trees for Q_3 with order $G \equiv 0$ (mod 4)



AI-partition



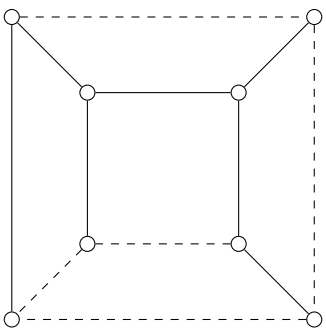


Figure 2.18: Two splitting trees we can construct from a given AI-partition

2.3 Cubic graphs with coherent decycling partitions

Definition 2.18. A cubic graph of order n has a coherent decycling partition if we have a partition $\{A, I\}$ of the graph such that the following holds:

if $n \equiv 2$ $(mod\ 4)$ then A induces a tree and I is an independent set
if $n \equiv 0$ $(mod\ 4)$ then A induces a tree and I is a near-independent set

Conjecture 2.19. [8, Conj. 1.5] G is upper embeddable and 3-connected $\iff$ G has a coherent decycling partition.

This is already proven when we used the notion of stable decycling partitions, so to prove this conjecture we actually need to prove: any 3-connected cubic graph G of order $\equiv 0$ $(mod\ 4)$ has a partition $\{A, I\}$ with A a tree and I near-independent iff G is upper-embeddable with two faces.

We already know this partition is either of this form or A is a 2-forest and I is independent iff G upper-embeddable, so Conjecture 2.19 is equivalent to the following conjecture:

Conjecture 2.20. G (of order 0 $(mod\ 4)$ and 3-connected) has an AI-partition with A consisting of two trees and I an independent set $\Rightarrow$ G has a partition (A_2, I_2) such that A_2 is a tree and I_2 is a near-independent set.

We can also look at this in terms of the splitting tree, then the conjecture is equivalent to:

Conjecture 2.21. G (of order 0 $(mod\ 4)$ and 3-connected) has a splitting tree such that $G-T$ has a single odd component with 1 edge $\Rightarrow$ G has a splitting tree such that $G-T$ has a single odd component with at least 3 edges.

Lemma 2.22. Every splitting tree T of a cubic graph G can be transformed into a splitting tree where $G-T$ has a component with 4 or more edges when order $G \equiv 2$ mod (4) and 3 or more edges when order $G \equiv 0$ mod (4).

Proof. If a cubic graph has a splitting tree, then it is upper-embeddable and therefore has an AI-partition where A is either a tree or a 2-forest. This means there are at least 2 vertices in A with degree 1. Take one of these vertices, it must be adjacent to two vertices in I.

AI-partition splitting tree

If these vertices are not connected in G, then we can choose 2 of the edges connected to each of the vertices in I such that we have 4 distinct edges and put them in the cotree. Because we have two vertices in I with a single vertex between them, we can choose these edges such that they form a single component in the cotree with at least 4 edges.

If the vertices are connected in G, then I is a near-independent set. In this case we can take the edges connecting the vertex from A to the vertices from I and the edge connecting the two vertices from I to each other. This makes a single component with at least 3 edges. □

In all the conjectures we have the condition that G must be 3-connected, this is because if we replace 3-connected by 1-connected or 2-connected in Conjecture 2.19 counterexamples have already been found.

1-connected counterexamples of the conjecture

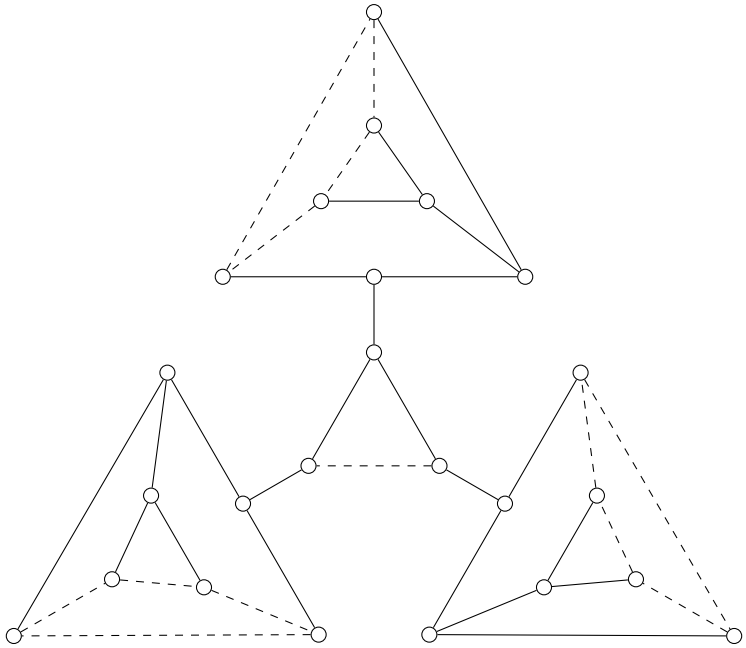
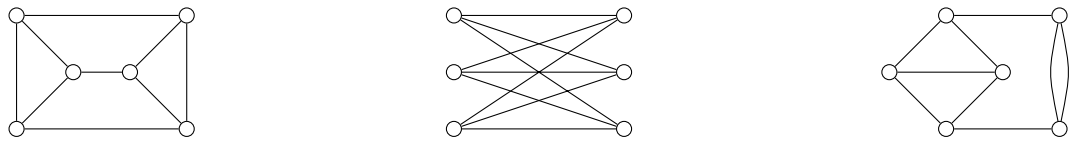
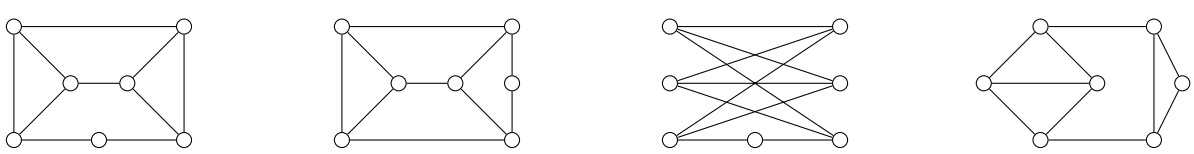


Figure 2.19: Example of a 1-connected cubic graph with stable, but not coherent decycling partition, shown in [8, fig. 2.]

This is a minimum counterexample of order 24. This is because the complement of any splitting tree for this graph can only have an odd-edge component with one edge. A possible splitting tree is shown in the figure, where dashed lines represent edges not in the tree. If we were to remove an adjacent edge to the single edge component, then the splitting tree would no longer be a tree. Adding the single edge to the tree is also not an option, because this would create a cycle. It has been checked by Jorik Jooken and Stijn Cambie using a combination of C and SageMath that there are precisely 20 simple graphs of order 24 that are 1-connected counterexamples. We can also see this by analyzing the construction of this example. This graph can be constructed by taking three gadgets and connecting them via a central triangle. These gadgets have to be of order $\equiv 2 \mod (4)$ *and they have to have an AI-partition. This means that to get a graph of order 24, we have to use gadgets of order 7. We can do this by taking cubic graphs of order 6 and subdividing one edge. There are two cubic graphs of order 6 and one cubic multigraph of order 6 with only one double edge:*



For the first graph we can construct two different gadgets, by subdividing an edge. For the second graph it doesn't matter which edge we subdivide, we always get the same gadget and for the last graph, we can subdivide one of the double edges to get a simple gadget, (we could also use multigraphs with multiple double edges, but this would give us gadgets that are not simple):



We thus have four different gadgets which we can combine in different ways to get counterexamples. This gives 4^3 combination when we combine three of them. We are only interested in all non-isomorphic combinations, so for this we get $\binom{4+3-1}{3} = 20$ combinations. This is the same number as we got from the computer search, so every simple counterexample of order 24 is constructed in this way. We can also use this construction with bigger gadgets to get even more 1-connected counterexamples. This also allows us to construct counterexamples with certain properties. We can also replace the center C_3 by another bigger cycle. For example, using a C_4 in the center and the gadget constructed from a $K_{3,3}$ graph we can construct the minimum triangle-free counterexample.

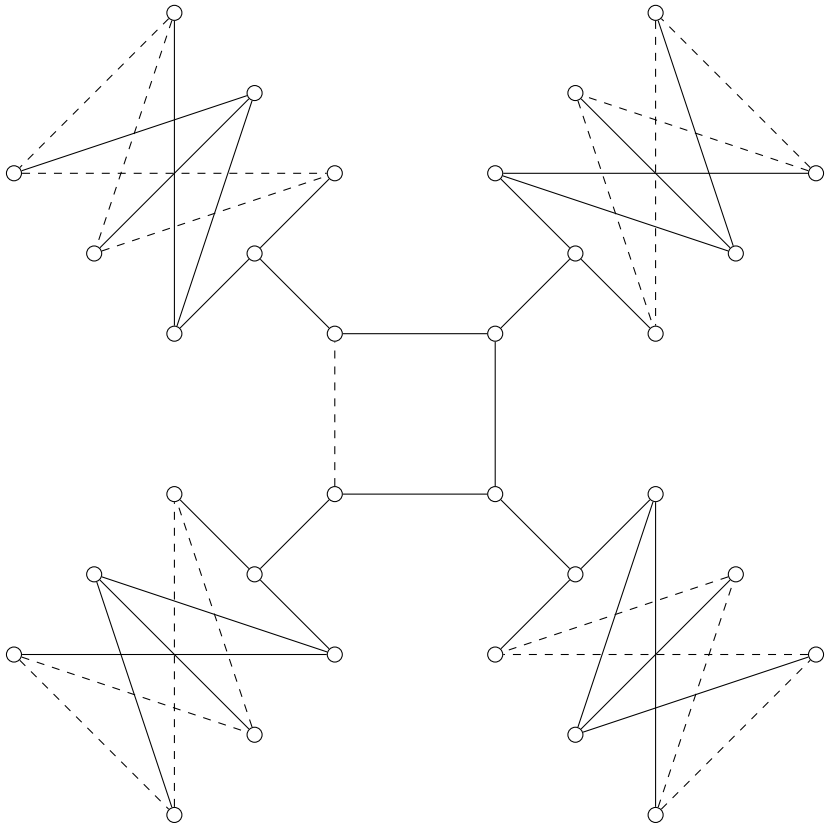


Figure 2.20: Minimum 1-connected triangle-free graph with stable, but not coherent decycling partition

If we allow multigraphs, we can construct a counterexample with order 8 using the same construction. This time we can place a C_2 in the center.

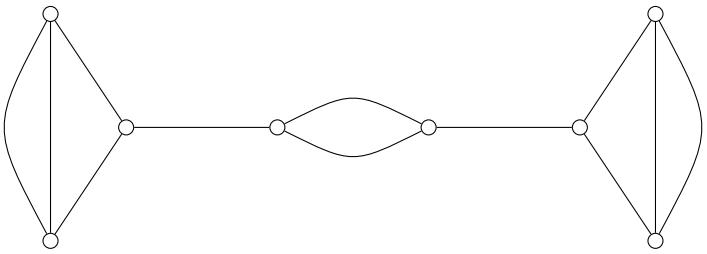


Figure 2.21: Minimum 1-connected multigraph with stable, but not coherent decycling partition

2-connected counterexamples of the conjecture

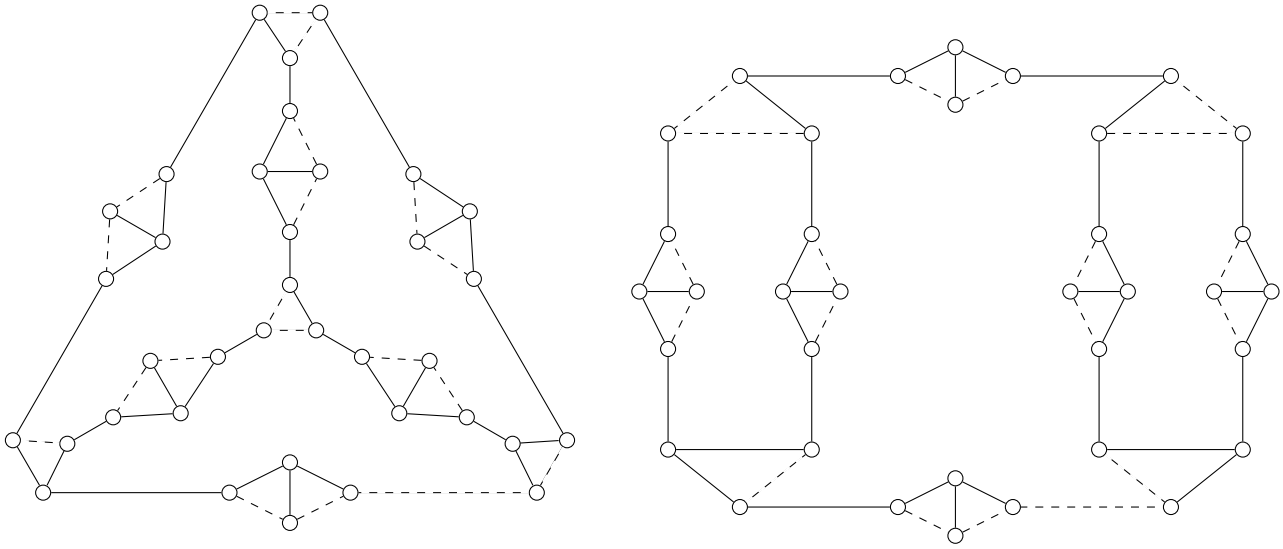
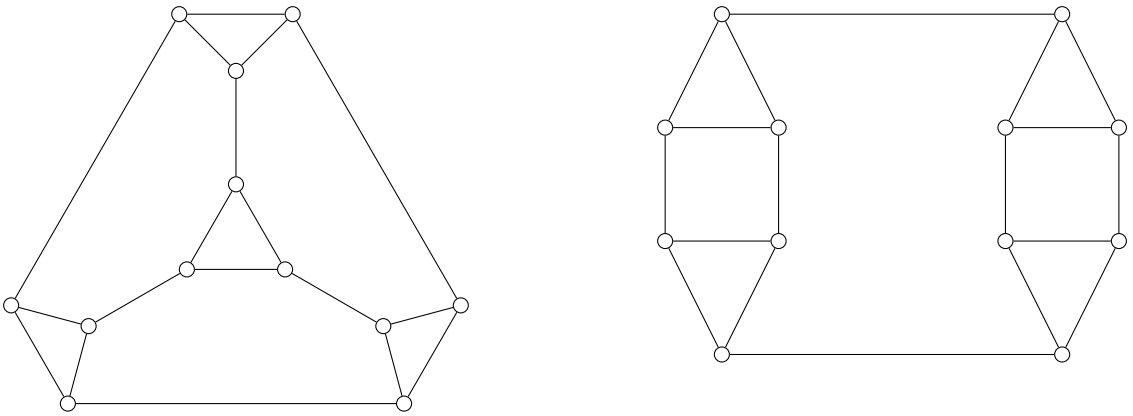


Figure 2.22: Examples of 2-connected cubic graphs with stable, but not coherent decycling partition, shown in [8, fig. 3.]

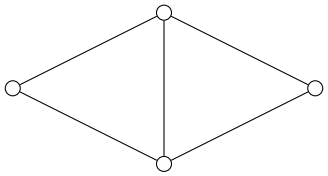
If we analyze these counterexamples, we see that they are constructed using a similar method as we used in Proposition 2.5. They have cubic graphs of order 4 as underlying graphs.

We use K_4 with one vertex removed as a gadget to place on the vertices of the underlying graph. Proposition 2.5 did not hold for underlying graphs of order 4 and we can now see why. These graphs do have an AI-partition.



For both of these graphs, we can put one vertex of every triangle into an independent set I to get a stable decycling partition into two trees and an independent set. The graphs do also have a coherent decycling partition. Then we must place two vertices from different triangles into I

that are adjacent. To make this impossible to do, we place a gadget between all the triangles on the connecting edges. In this case the gadget is the following:



Which is K_4 with one edge removed. The use of these gadgets makes sure that the graph does not have a coherent decycling partition, but it still has a stable decycling partition. We can construct different gadgets by taking a 2-connected cubic graph of order $0 \bmod (4)$ with AI-partition and removing one edge from it.

The way that we do this is by considering a coherent decycling partition (A, J) of this graph. If it does not have one, then we cannot use it but that would make this graph a counterexample to the conjecture, so then we would no longer need our construction. Add one of the two connected vertices from J back to A. This creates a cycle in A, but makes J independent. Then we just remove one of the edges of this cycle from the graph to construct our gadget. Using such gadgets will give us more counterexamples, but they will always remain 2-connected.

The examples in Figure 2.22 are actually the smallest counterexamples using this construction, even when we allow multigraphs. If we consider all possible gadgets, then a general counterexample construction with underlying graph the cubic multigraph of order 4 looks like this.

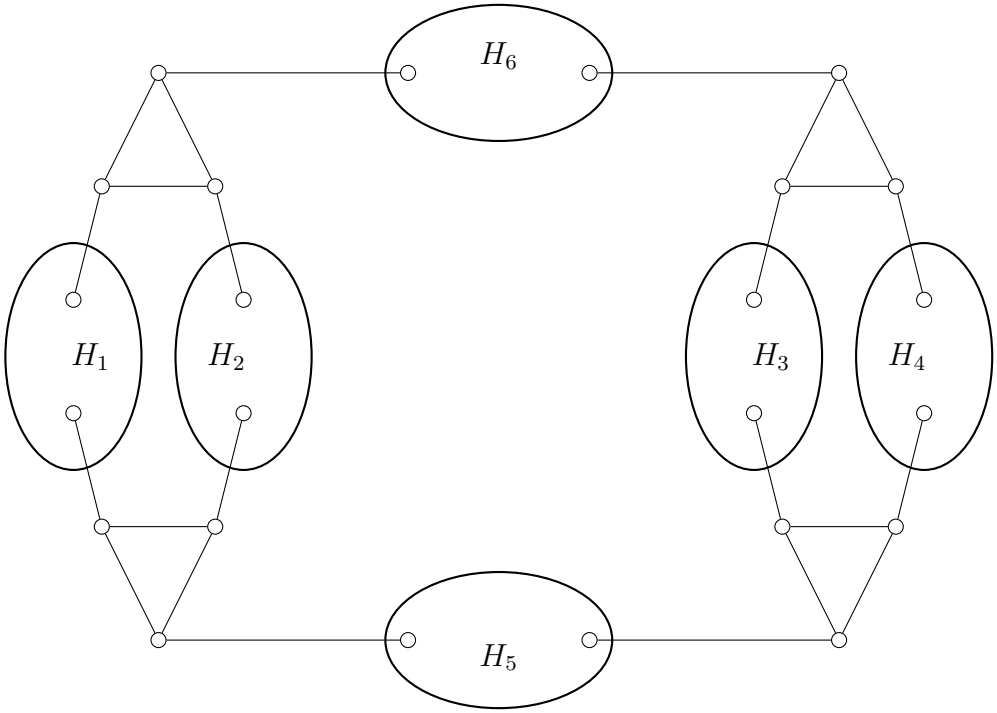


Figure 2.23: Construction for 2-connected counterexamples

3-connected counterexamples of the conjecture

Now we can consider if we can find another gadget to place on these edges to separate the triangles that leaves the graph 3-connected. For this we need to construct a gadget by taking a

cubic graph H and removing one vertex from it. We can use graphs that have stable decycling partitions to guarantee that the whole graph still has at least a stable decycling partiton. Notice that if we use a cubic graph of order $2 \bmod (4)$, then it necessarily has a partition into a tree and an independent set.

For any given stable decycling partition of H, there are two options. The vertex we removed to construct the gadget is either in A or in I. If it was in A, then the partition disconnects our gadget into three trees. This is a problem, because now the whole graph G cannot have a stable decycling partition. If the removed vertex was in I, this means that all the vertices of degree 2 in our gadget, which we will call the connecting vertices, are connected in the partition.

When we construct gadgets using a cubic graph of order $0 \bmod (4)$, we get something different. If we have a coherent decycling partition, then by removing a vertex that was in A, we get a gadget where all the connecting vertices are disconnected. Some of them could also be in I, with I still a near-independent set. By removing a vertex that was in I, we get either that the three connecting vertices are connected in A or two of them are connected and one is in I. In this second case we get that I is an independent set. We want to construct a graph with a stable decycling partition, but no coherent decycling partition thus for every gadget, the set I can only be independent. A gadget constructed from a cubic graph of order $0 \bmod (4)$ can only have partitions with the following structure.

For every coherent partition, vertices u, v are always in I and connected to each other.

If we can find such a graph G, then we can use it as a gadget by removing u to construct a 3-connected counterexample:

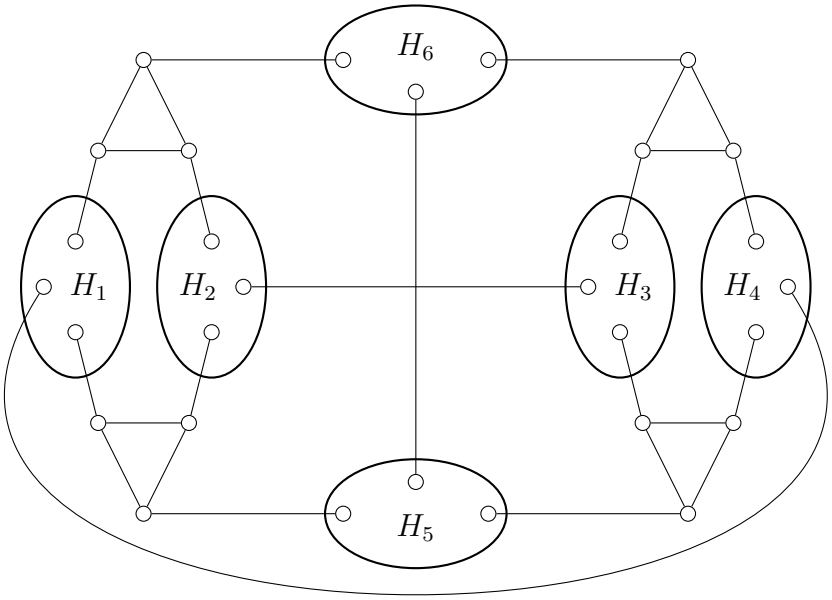


Figure 2.24: Construction for 3-connected counterexamples

Here H_1, H_3, H_5 are gadgets constructed from graphs of order $2 \bmod (4)$ and H_2, H_4, H_6 are gadgets constructed from graph G. When we look at a partition for this graph, gadgets H_1, H_3, H_5 leave everything connected and for gadgets H_2, H_4, H_6 we get an independent set

and 2 of the 3 connecting vertices are connected. A stable decycling partition then looks like this:

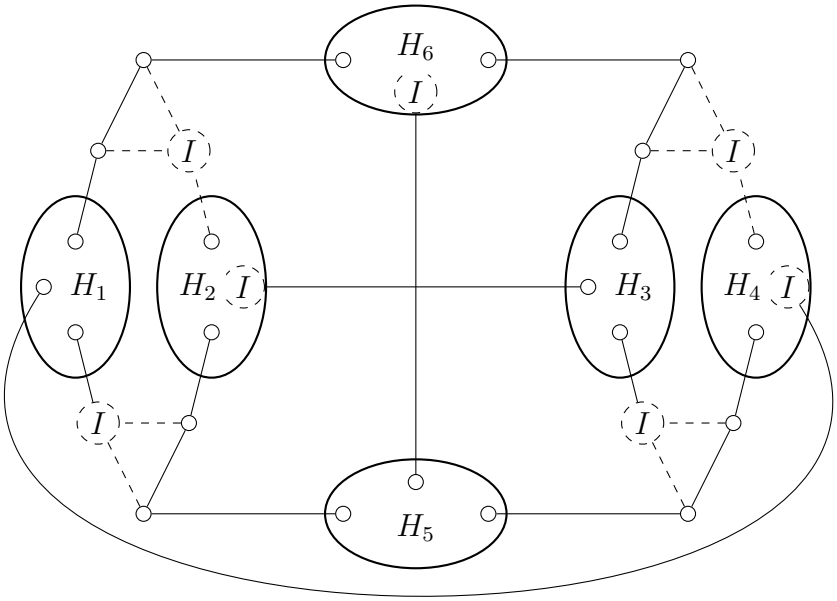


Figure 2.25: Stable decycling partition of 3-connected counterexample

This is a stable, but not a coherent decycling partition. Because of the restrictions we put on our gadgets, we cannot make a construction with these gadgets where we have a vertex that is in I and connected to a vertex of one of the triangles.

We have thus transformed the problem of finding a counterexample to finding a graph such that every coherent decycling partition of that graph always has the same two adjacent vertices in I. A graph without a coherent decycling partition, but with a stable decycling partition also works. Then we remove a vertex that is in I to construct the gadgets H_2, H_4, H_6. This is what we are trying to find, so it does not help us directly.

What this does mean is that if we find one example of a graph with a stable, but not a coherent decycling partition, then we can use this construction to generate an infinite family of such graphs.

Chapter 3

Oriented graphs

In this section we look at a recent result from the paper of S. Cambie, F. Dross, K. Knauer, H. La, and P. Valicov [3] last year about AI-partitions. Before we do this, we will first look back at Conjecture 1.21 and the counterexample from Tutte in Figure 1.17. Furthermore, we will look at some more related conjectures throughout history.

3.1 Some related conjectures

Although he did disprove the original conjecture with his counterexample, Tutte proposed that the conjecture could still be true just with an alternative condition.

Conjecture 3.1 (Tutte's conjecture)**.** [15] Every 3-connected simple bipartite cubic graph has a Hamiltonian cycle.

This conjecture has since been disproven again by counterexamples, the smallest of which is Georges graph with 50 vertices.

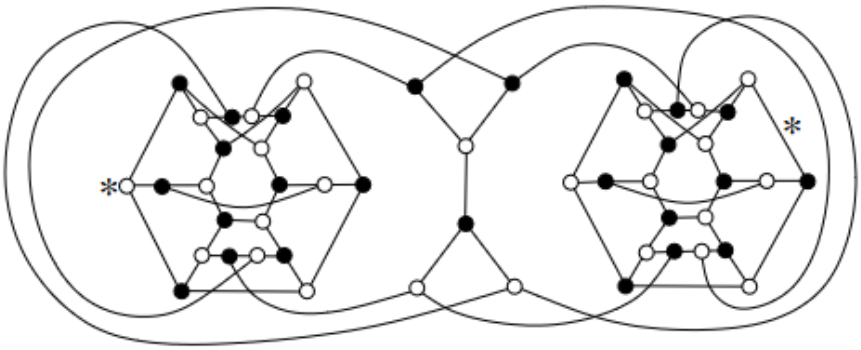


Figure 3.1: Smallest counterexample to Tutte's conjecture [2]

In 1970 Barnette conjectured the following conjecture, which combines the conjectures of Tutte and Tait.

Conjecture 3.2 (Barnette's conjecture)**.** [4, conj. 1.4.] Every 3-connected simple bipartite cubic planar graph has a Hamiltonian cycle.

Conjecture 3.2 seems to be the strongest conjecture we can propose using these conditions that has not yet been proven false. However, we can also consider Digraphs. Hochstättler [6, conjecture 5] proposed a conjecture that is equivalent to this one:

Conjecture 3.3 (Hochstättler's conjecture)**.** Every oriented graph can be vertex-partitioned into two acyclic sets.

At first glance this does not seem related to Conjecture 3.2, but if we consider the dual of Barnette's conjecture we can see that they are very similar.

Lemma 3.4. The dual of a bipartite graph is an Eulerian graph.

To make this easier to work with, we will first prove another lemma.

Lemma 3.5. A graph is Eulerian $\iff$ the graph is connected and every vertex has even degree.

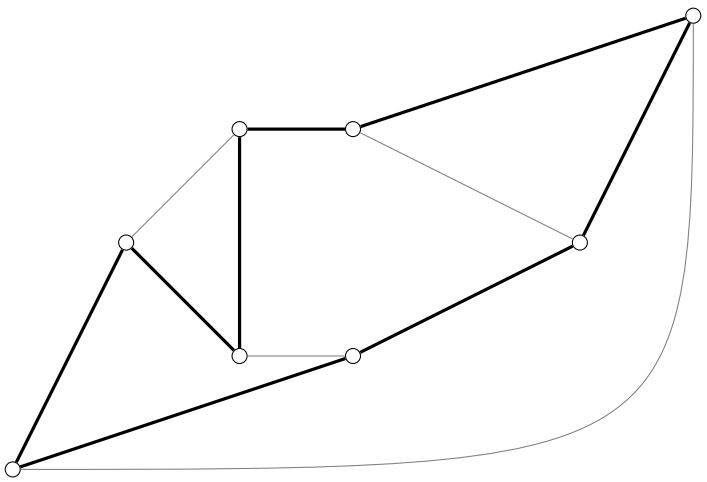
Proof. An Eulerian graph, as stated in Definition 1.23, is a graph that contains a circuit that uses every edge exactly once. This graph clearly has to be connected and for every vertex, the Eulerian circuit must arrive in that vertex and depart from that vertex equally as many times. This can only happen when every vertex has even degree.

Now assume that $G = (V, E)$ is connected and every vertex has even degree and E is not empty, otherwise it is trivially true. Because every vertex has even degree, we can always depart from a vertex if we arrived in it. There must exist at least one circuit in the graph, call the set of edges from this circuit C. If this is a Eulerian circuit we are done, otherwise take the graph $G_1 = (V, E - C)$. Every vertex in this graph still has even degree, so we can repeat this argument on the components of G_1. We will eventually get a number of components that all contain an Eulerian circuit. Now we can combine them back to get a full Eulerian circuit of G. We do this by going along our circuit C and every time we get to a vertex v_i that is in a component C_i, we traverse the Eulerian circuit in C_i to arrive back at v_i and continue going along C. □

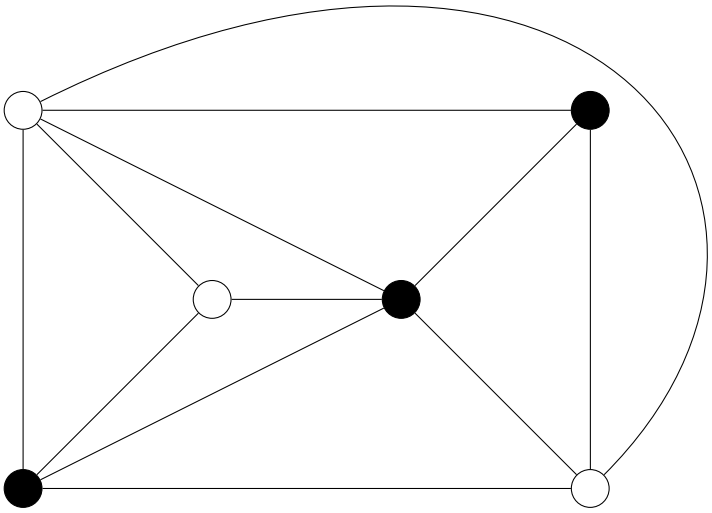
Using Lemma 3.5 it is clear that Lemma 3.4 holds.

Lemma 3.6. The dual of a Hamiltonian graph is a graph that has a vertex-partition into 2 trees.

Proof. Consider a graph with a Hamiltonian cycle.



We can partition the faces of this graph into 2 sets. The faces contained within the Hamiltonian cycle and the faces not contained within the Hamiltonian cycle. If we consider the dual of this graph, this gives us a vertex-partition into 2 sets of vertices.



These sets of vertices must each induce a tree in the dual graph. To see this, suppose that one of the sets induces a subgraph containing a cycle. Then there is a path between faces in the original graph that starts and ends in the same face, a face-cycle. Any edge between those faces cannot be in the Hamiltonian cycle. This means the Hamiltonian cycle cannot cross the face-cycle, so it clearly cannot be Hamiltonian. There is at least one vertex inside the face-cycle, which we cannot reach if we start outside of the face-cycle. □

Using Lemma 3.4, Lemma 3.6 and the fact that he dual graph of a cubic graph is a triangulation, we can state the dual Barnette conjecture.

Conjecture 3.7 (dual Barnette)**.** Every 3-connected planar Eulerian triangulation can be vertex-partitioned into two acyclic sets.

We can now see that Conjecture 3.3 is related to this dual conjecture. We also have one more conjecture that is related, stated by Neumann-Lara in 1985.

Conjecture 3.8 (Neumann-Lara)**.** Every oriented planar graph can be vertex-partitioned into two acyclic sets.

We can combine all those conjectures to get what we will call the Eulerian Neumann-Lara conjecture.

Conjecture 3.9 (Eulerian Neumann-Lara)**.** Every oriented planar Eulerian triangulation can be vertex-partitioned into two acyclic sets.

We have the following figure which tells us the relation between all the conjectures. Here we restrict all the conjectures with the condition that the graphs have to be 3-connected. They also have to be cubic, or in the dual case they have to be triangulations.

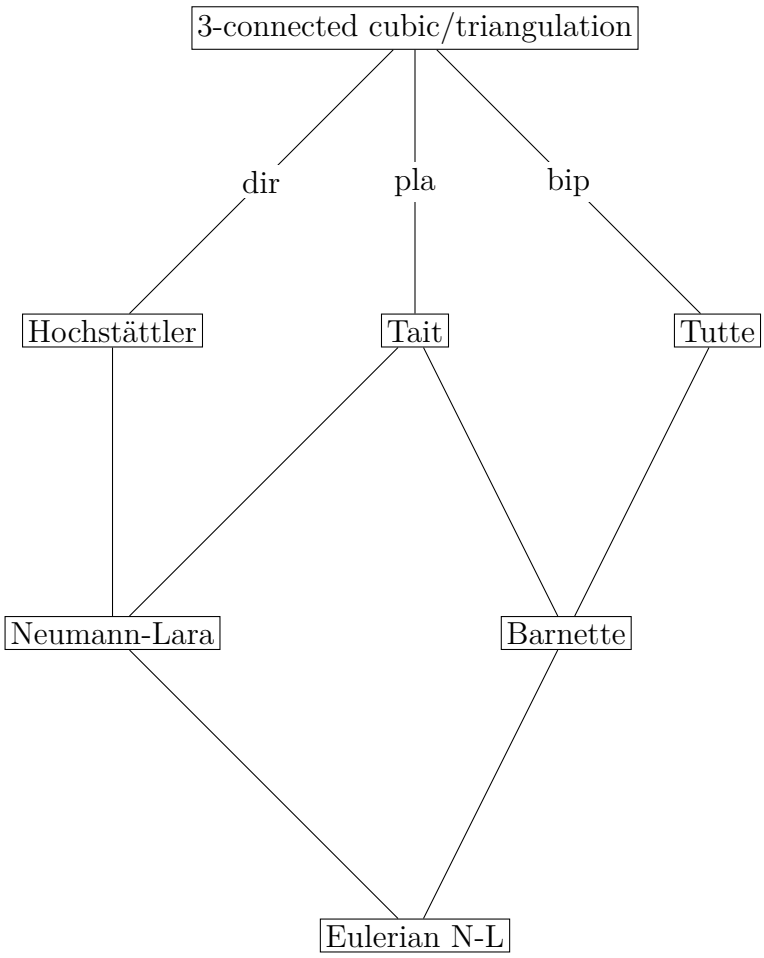


Figure 3.2: Relation between conjectures

There is one more observation that will connect these conjectures to AI-partitions. First we will need this lemma.

Lemma 3.10. Every Eulerian planar triangulation has a tripartition.

We will show the construction to find such a tripartition using an example.

Example 3.11. Take an arbitrary Eulerian planar triangulation.

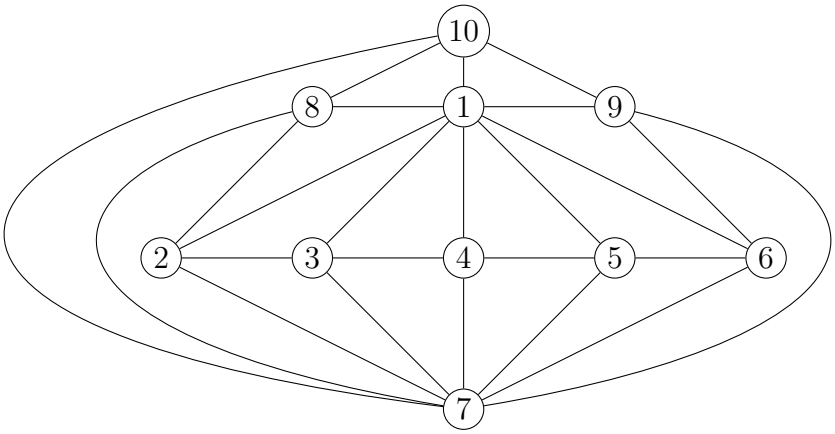
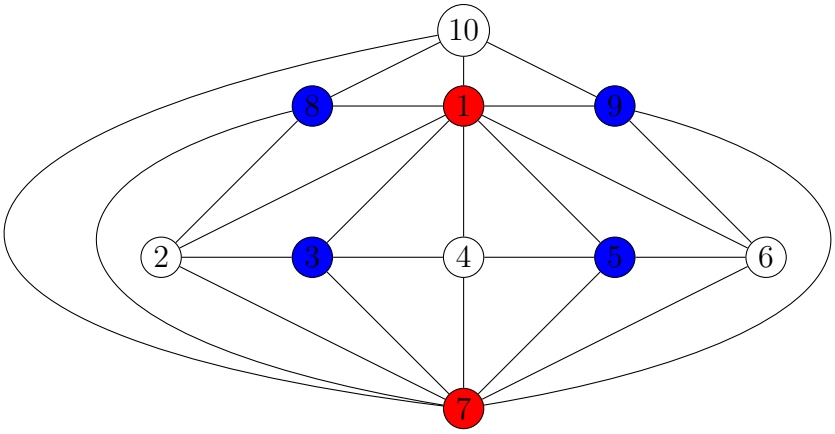


Figure 3.3: Example of a planar Eulerian triangulation

Start by coloring one vertex in red. After this, we can color all adjacent vertices using just two colors, this is possible because every vertex has an even number of neighbors. Now we use the fact that every face is a triangle. Take any face where two of the surrounding vertices are colored and color the uncolored one in the third color. Repeat this until we have a complete 3-coloring of the graph. This 3-coloring gives a partition in 3 independent sets.



$I_1 = (2,4,6,10)$, $I_2 = (3,5,8,9)$, $I_3 = (1,7)$

Figure 3.4: Tripartition of planar Eulerian triangulation example

It was proven in [12] by Mu-Tsun Tsai and Douglas B. West that such a 3-coloring is possible for Eulerian triangulations and also more general for Eulerian near-triangulations. These are graphs where every face is a triangle except possibly the outside face. This is done by a slightly different argument where we take the Eulerian circuit and vertex-color it using three colors in a fixed repeating sequence.

Theorem 3.12. *For the tripartition (I_1, I_2, I_3) of the planar Eulerian triangulation G, if $G - I_i$ has an AI-partition, then G can be partitioned into two acyclic graphs.*

The definition of AI-partition is slightly different in this context than what we saw before. For this chapter we define an AI-partition always as a partition into a connected acyclic set and an independent set. Here we are working with both graphs and oriented Digraphs, so a connected acyclic graph is not necessarily a tree, but can also be a Digraph without oriented cycles. We are also no longer working with only cubic graphs, so $|V(G)|$ can be an odd number.

Proof. Take a planar Eulerian triangulation with tripartition (I_1, I_2, I_3). If we remove the vertices in I_1 from the graph, we get the graph $G - I_1$. By symmetry the following argument also holds for $G - I_2$ and $G - I_3$.

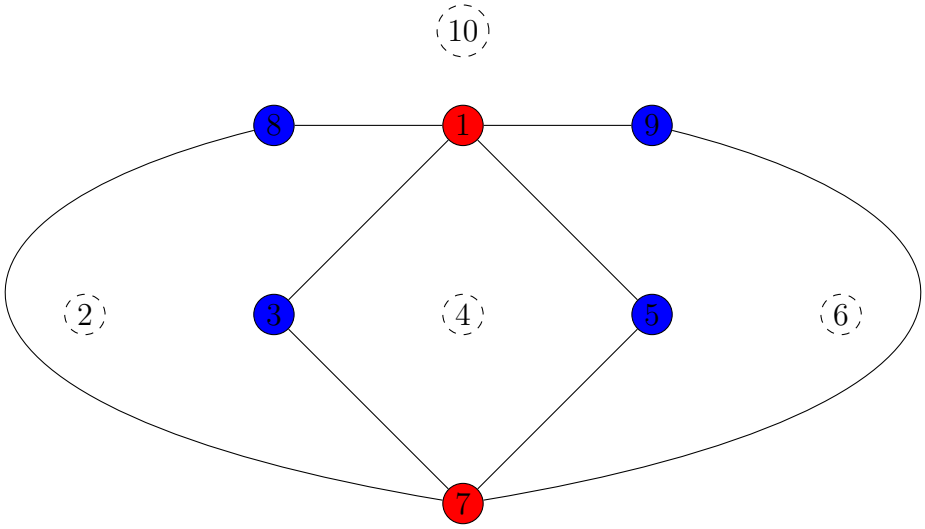
Assume $G - I_1$ has an AI-partition into a tree A and an independent set I. Now take sets $A_1 = I_1 \cup I$ and $A_2 = A$, they both induce acyclic subgraphs. A_2 is always a tree because of the definition of AI-partitions and A_1 is acyclic but can be a forest.

To see that A_1 must be acyclic, we will assume it is not and arrive at a contradiction. Say that A_1 contains a cycle C. This must be an even cycle because A_1 is the union of two independent sets and thus 2-colorable. We know that G was a triangulation, so any cycle greater than 3 must have at least one internal vertex surrounded by the cycle. This means that there is at least one vertex surrounded by the cycle in G that is not in A_1, so it must be in A_2. Similarly, the cycle cannot be the outside cycle of G because this is also a triangle in a triangulation. This tells us that there must be at least one vertex from A_2 inside the cycle and one outside

the cycle. This makes A_2 disconnected and thus no longer a tree.

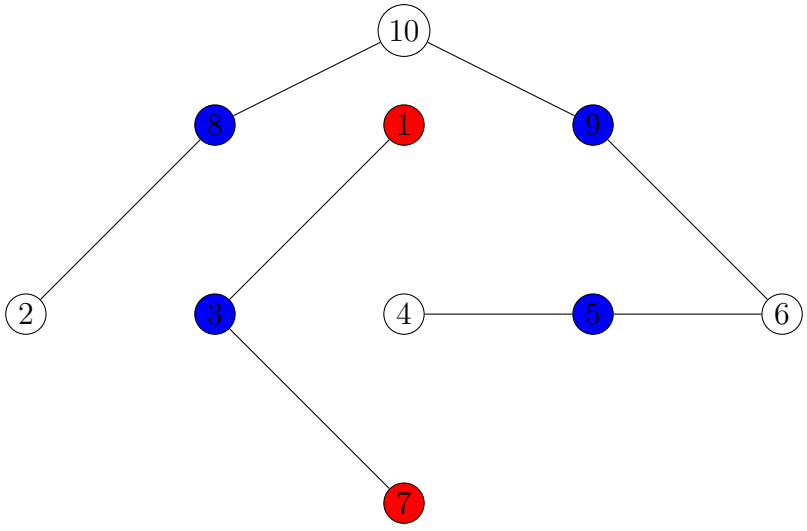
This theorem still holds when G is an oriented planar Eulerian triangulation, because if we give the acyclic subgraphs an orientation they remain acyclic. □

Example 3.13. Take the planar Eulerian triangulation with tripartition (I_1, I_2, I_3) in Figure 3.4. If we remove the vertices in I_1 from the graph, we get the graph $G - I_1$.



$I_1 = (2,4,6,10)$

The sets $A_1 = I_1 \cup I$ and $A_2 = A$ form a partition in two acyclic subgraphs.



$A_1 = I_1 \cup I = (2, 4, 5, 6, 8, 9, 10), A_2 = (1, 3, 7)$

Theorem 3.12 connects the conjectures to the concept of AI-partitions. This leads us to define a new lemma in terms of AI-partitions using the following theorem.

Theorem 3.14. *A(n oriented di)graph H is induced by two parts of the tripartition of a planar Eulerian (oriented) triangulation $\iff$ H is a 2-connected bipartite planar (oriented) graph.*

Proof. Take any 2-connected bipartite planar graph G. We can add a new vertex in every face of G and connect it to every vertex on the cycle surrounding the face. Doing this for every face ensures that the resulting graph is a planar triangulation. It is also Eulerian because G was bipartite, so every vertex already in G is now connected to an even number of new vertices and the vertices we added cannot be connected to each other. We have constructed a planar Eulerian triangulation by adding an independent set.

Now for the other direction, take a planar Eulerian triangulation H with tripartition (I_1, I_2, I_3). Clearly $H - I_i$ for $i = 1, 2, 3$ is bipartite, because it is 2-colorable. Removing vertices also cannot break planarity. To see that $G = H - I_i$ is 2-connected, consider 2 vertices u, v in G that are connected to each other. We know that they were both connected to a vertex w in I_i, because H was a triangulation. If H is a K_3, G is clearly 2-connected. Assume H has more than 3 vertices, then any vertex must have degree at least 4. If a vertex would have degree 2, this results in a cycle with length at least 4 and thus the graph would not be a triangulation. Vertices also cannot have odd degree in an Eulerian graph. Removing vertex w with degree at least 4 leaves a cycle of degree at least 4 on which u, v lie. This shows that any two adjacent vertices lie on a cycle together and thus the graph must be at least 2-connected. □

We can use this to state Conjecture 3.15 that implies Conjecture 3.9 and because of this directly relates the previous lemmas with the concept of AI-partitions.

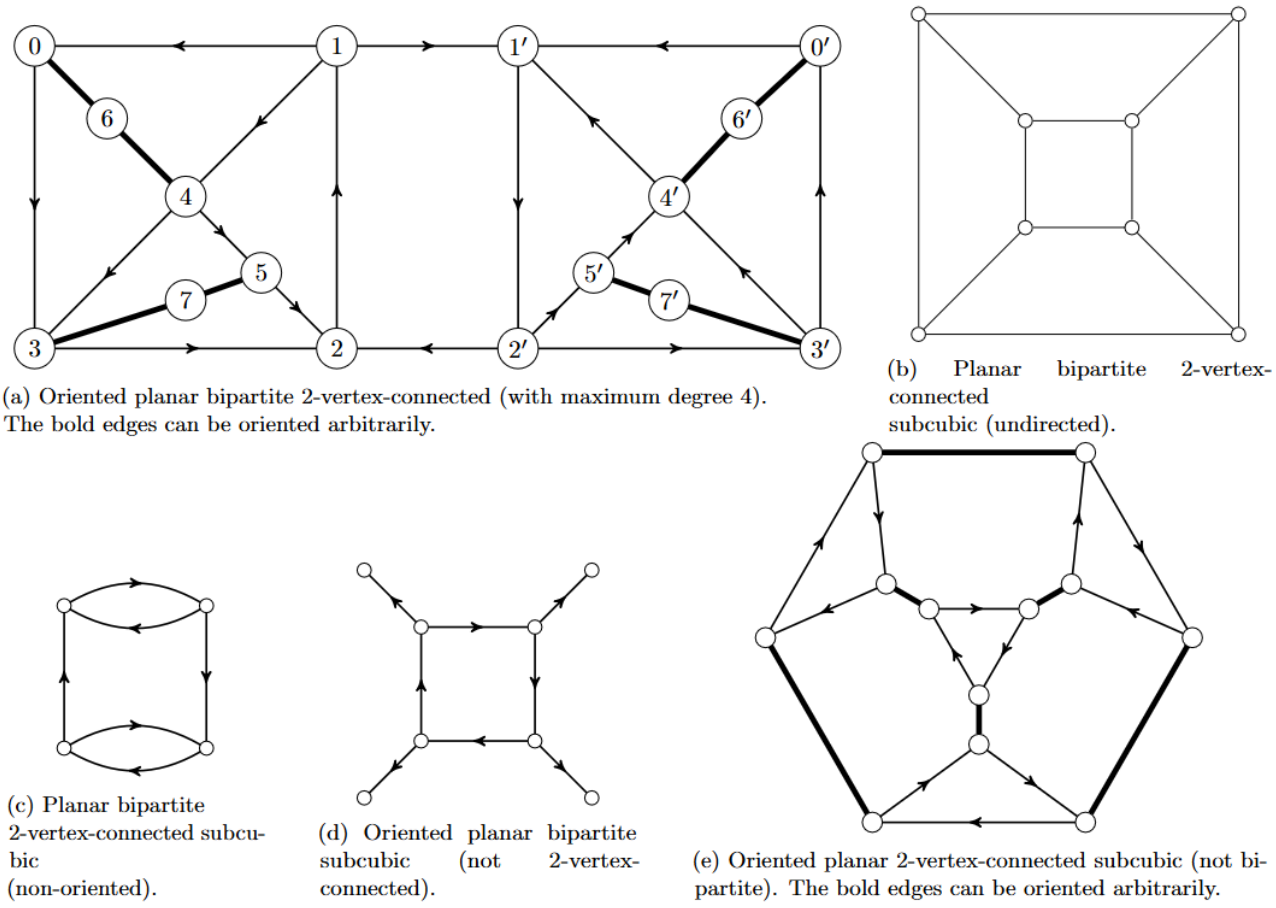
Conjecture 3.15. Every planar bipartite 2-vertex-connected oriented graph has an AI-partition into a connected acyclic set and an independent set.

Recently, a proof has been given by S. Cambie, F. Dross, K. Knauer, H. La, and P. Valicov for a more restricted version of Conjecture 3.15. Here subcubic was added as an extra requirements.

Theorem 3.16. *[3, Theorem 9] Every planar bipartite 2-vertex-connected subcubic oriented graph has an AI-partition into a connected acyclic set and an independent set.*

There are a lot of conditions in the theorem. We may ask ourselves the very natural question if all of these are absolutely necessary for the theorem to hold. In the paper they give counterexamples for when one of the conditions, subcubic, directed, simple, 2-vertex connected or bipartite is not met. Here they are shown in Figure 3.5. Thus, all of these conditions are necessary. For the case where planar has been left out, a counterexample has not yet been found.

A follow-up question we can ask is if all these examples are also vertex-minimum, examples with the smallest amount of vertices. Furthermore, if they are we can ask if they are unique. We used a SageMath program (for the actual program, see appendix A or [11]) to find minimum counterexamples of the theorem when some, but not all of the conditions are met. In the next section We will explain the working of the algorithm using pseudocode.



(a) Oriented planar bipartite 2-vertex-connected (with maximum degree 4). The bold edges can be oriented arbitrarily.

(b) Planar bipartite 2-vertex-connected subcubic (undirected).

(c) Planar bipartite 2-vertex-connected subcubic (non-oriented).

(d) Oriented planar bipartite subcubic (not 2-vertex-connected).

(e) Oriented planar 2-vertex-connected subcubic (not bipartite). The bold edges can be oriented arbitrarily.

Figure 3.5: Graphs with no AI-partition for specific properties [3, figure 15]

3.2 Checking for minimum counterexamples using an algorithm

We want a program that takes a graph as input and determines whether it does not have an AI-partition, or, in the oriented case, whether it has an orientation without an AI-partition. The easiest way to do this is to check all possible orientations and, for each oriented graph, go over all possible subsets of vertices using a brute-force algorithm. This would take a very long time so it is better to try some more clever method to do this. We will use something we will call a satisfiability graph. We start with a graph $G = (V, E)$, the goal is to find an orientation of this graph that does not have an AI-partition. First we go over all independent subsets of V and we check what kind of partitions they give us for G.

Algorithm 1: *FindPartitionsWithSingleCycleLeft*(G)

```
   input : A graph (G)
   output: List of cycles (onecycles), a list of valid independent sets (valid_subsets)
 1 foreach subset in IndependentSets(G) do
 2 |   partitioned_graph ← G_{V−subset};
 3 |   if partitioned_graph is connected then
 4 |   |   if partitioned_graph has exactly one cycle then
 5 |   |   |   onecycle ← FindTheCycle(partitioned_graph);
 6 |   |   |   add onecycle to onecycles;
 7 |   |   |   add subset to used_subsets;
 8 |   |   else
 9 |   |   |   if partitioned_graph is a tree then
10 |   |   |   |   return False;
11 |   |   |   else
12 |   |   |   |   add subset to valid_subsets;
13 Remove subsets of sets in used_subsets from valid_subsets;
14 return onecycles, valid_subsets;
```

We use the build-in function IndependentSets from sagemath to get the independent subsets of V. If a subset gives a partition into a tree we can return False. Then the undirected graph has an AI-partition, so every orientation of this graph will also have an AI-partition. If the graph does not have an AI-partition, the algorithm FindPartitionsWithSingleCycleLeft *returns all the cycles for which there is an independent set that, when removed from G, leaves that cycle as the only cycle left. We also saved all the independent sets that leave the graph connected, because those that disconnect the graph can never give an AI-partition for any orientation of G, so we do not need to check them again. The independent sets we also do not need to check again are those that leave the cycles we put in onecycles and their subsets, we will see why later. To find the onecycle, the following algorithm* FindOnecycle *was used.*

Algorithm 2: *FindOnecycle*(G)

```
   input : A graph with exactly one cycle (G)
   output: A list of vertices in the cycle
 1 while there are vertices of degree 1 in G do
 2 |   remove all degree-1 vertices from G;
 3 start ← any vertex remaining in G;
 4 cycle ← [start];
 5 for i ← 1 to |V(G)| do
 6 |   neighbors ← neighbors of the vertex cycle[i − 1];
 7 |   if one of the neighbors is already in cycle then
 8 |   |   add the other neighbor to cycle;
 9 |   else
10 |   |   add the first neighbor to cycle;
11 return cycle;
```

In algorithm SatisfiabilityOfSingleCycles *we construct our satisfiability graph. We iterate over all cycles in the set onecycles we got from* FindPartitionsWithSingleCycleLeft *and construct two vertices for every edge in this cycle. Here we actually view each edge as two arcs. We then connect the vertices representing arcs that form an oriented cycle.*

Algorithm 3: *SatisfiabilityOfSingleCycles*$(G, onecycle)$

input : *A graph (`G`), a list of cycles (`onecycles`)*
output: *list of arcs and a mapping of those arcs to components if satisfiable, else* ***False***

```
1  satgraph ← empty graph;
2  foreach cycle ∈ onecycles do
3  |  foreach edge ∈ cycle do
4  |  |  create two vertices in satgraph representing this edge in both directions;
5  |  connect all such vertices in satgraph that follow the cycle in the same direction;
6  if number of components in satgraph is 1 then
7  |  return False;
8  foreach component in components of satgraph do
9  |  if component contains both directions of any edge then
10 |  |  return False;
11 make a list of all arcs represented by vertices in satgraph;
12 assign each arc the component ID it belongs to;
13 return list of arcs and their component mapping;
```

The goal of SatisfiabilityOfSingleCycles *is to calculate whether we can orient all these cycles simultaneously. If we cannot do this then all orientations of G will have an AI-partition. For any orientation there is always one of these cycles that is not oriented, so then we can take I to be the independent set that left just that one cycle. This is then a partition into I and an acyclic graph. If it is possible to orient all the cycles at the same time, the algorithm returns arcs and values for those arcs that tell us how we can orient them. If we orient an arc from a component of satgraph in one way then this tells us how we must orient the other arcs from that component. We can now make the Digraph $d = (V, A)$ using this information. For the edges we do not have an orientation for yet, we iterate over the remaining possibilities using this algorithm.*

Algorithm 4: *FindDirectedPartition*$(G, valid\_subsets)$

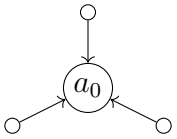
input : *A Digraph `G` and a list of subsets of $V(G)$ `valid_subsets`*
output: *partitioned graph if a partition into a connected acyclic graph and an independent set exists, else* ***False***

```
1  foreach subset ∈ valid_subsets do
2  |  partitioned_Digraph ← G_{V−subset};
3  |  if partitioned_Digraph is connected and acyclic then
4  |  |  return partitioned_Digraph ∪ G_{subset};
5  return False;
```

We can avoid checking some of the orientations of G by using Lemma 3.17, this was used to improve the code efficiency.

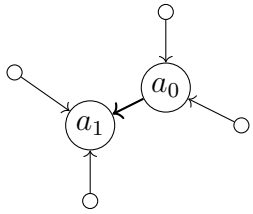
Lemma 3.17. We only have to check graphs with $\delta^-(G), \delta^+(G) > 0$ when searching for graphs without AI-partition.

Proof. Consider a vertex with minimum out-degree $\delta^+(G) = 0$:

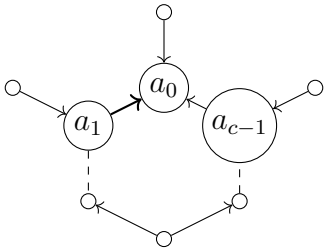


Because this vertex has no outgoing edges, it cannot be part of a directed cycle. We can thus change the direction of one of the edges without destroying a cycle. However it could be the

case that the vertex connected to this first one now has out-degree 0, because we changed one of its outgoing edges.



In this case every other edge of a_1 is an incoming one. Because of this a_1 also cannot be a part of a directed cycle, so we can again change the direction of one of the other edges such that a_1 has out-degree 1. We can repeat this process until we no longer create a vertex with out-degree 0. This is always possible because we can follow this process along an undirected cycle $a_0a_1...a_{c-1}$ and there has to be at least one vertex along this cycle that has out-degree at least 2 initially. This is because a_0 has 2 incoming edges along the cycle, this implies that there is at least one other vertex with 2 outgoing edges along the cycle. When we repeat the process of flipping edges along the cycle we can stop at the first vertex with out-degree at least 2. The same argument holds for vertices with in-degree 0. By doing this we could possibly create more directed cycles. This is not a problem because if we find a partition for this graph with more cycles it would still be a partition for the original graph.



□

Then we just have our main algorithm that brings all the others together.

Algorithm 5: Main Algorithm

```
   input : A graph G
 1 cycles_left ← FindPartitionsWithSingleCycleLeft(G);
 2 if cycles_left ≠ False then
 3 |   satisfy ← SatisfiabilityOfSingleCycles(G, cycles_left);
 4 |   if satisfy ≠ False then
 5 |   |   Assign all edges in satisfy their direction;
 6 |   |   foreach possibility for the remaining edges do
 7 |   |   |   assign remaining edges to get directed graph d;
 8 |   |   |   if (δ⁻(d) > 0 and δ⁺(d) > 0) or G is not 2-connected then
 9 |   |   |   |   solution ← Find Directed Partition(d);
10 |   |   |   |   if solution = False then
11 |   |   |   |   |   show(d);
12 |   |   |   |   |   print('has no partition');
```

3.3 Minimum counterexamples found by the algorithm

Non-oriented

Let us start with the condition of being oriented left out. We can interpret this in two ways, either we say that the graph still has to be directed, but double edges are allowed, thus making the graph non-oriented, or we say that the graph can be undirected. The counterexamples given in Figure 3.5 for these cases are minimum examples. The example given for the second case is actually 3-vertex-connected. The smallest example that is 2-vertex-connected, but not 3-vertex-connected is this graph of order 16.

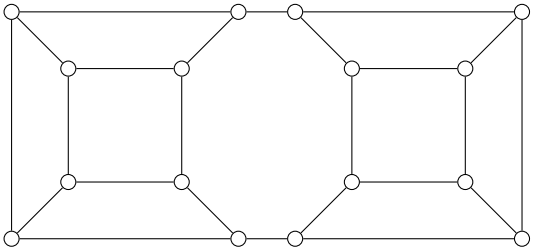


Figure 3.6: Undirected counterexample that is 2- but not 3-vertex-connected

This is an interesting example, because this is a cubic graph and in the previous chapter we would have considered this graph to have an AI-partition using our alternate definition where we allow I to be near-independent. In the context of this theorem however, we no longer use this definition and this is indeed a counterexample.

Not 2-vertex-connected

For this one the example given in Figure 3.5 is also the minimum counterexample if we require the graph to still be 1-vertex-connected. When we remove connectedness entirely as a condition we find a counterexample that is a lot smaller.

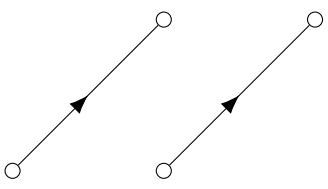


Figure 3.7: Counterexample that is not connected

Maximum degree 5

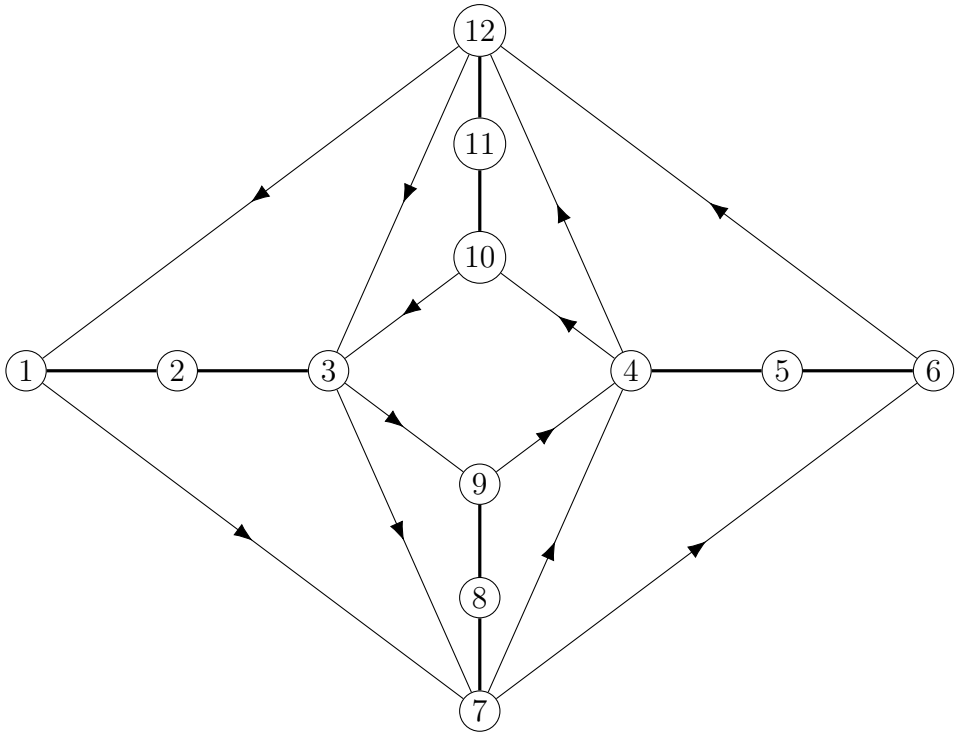


Figure 3.8: Counterexample of order 12 with maximum degree 5

In this example, bold edges can be oriented arbitrarily.

The graph in Figure 3.8 does not have an AI-partition.

Proof. In this graph we have the following cycles:

$\{10, 12\} \to \{1, 3\} \to \{7, 9\} \to \{4, 6\} \to \{10, 12\}$

Where the notation $\{x, y\}$ means that we choose either x or y, this gives us 2 distinct cycles each time. To break all the cycles, we must put both of the vertices in one of the pairs in I, but we cannot do this without disconnecting the graph.

- $\{10, 12\} \in I$ then 11 is disconnected from the rest.
- $\{1, 3\} \in I$ then 2 is disconnected from the rest.
- $\{7, 9\} \in I$ then 8 is disconnected from the rest.
- $\{4, 6\} \in I$ then 5 is disconnected from the rest.

We cannot put both of the vertices of any pair in I, so there will always be a cycle left in the graph. We conclude that this graph cannot have an AI-partition. □

Maximum degree 4

The example given in Figure 3.5 is of order 16. This is not a minimum counterexample. There exists a counterexample with maximum degree 4 of order 14.

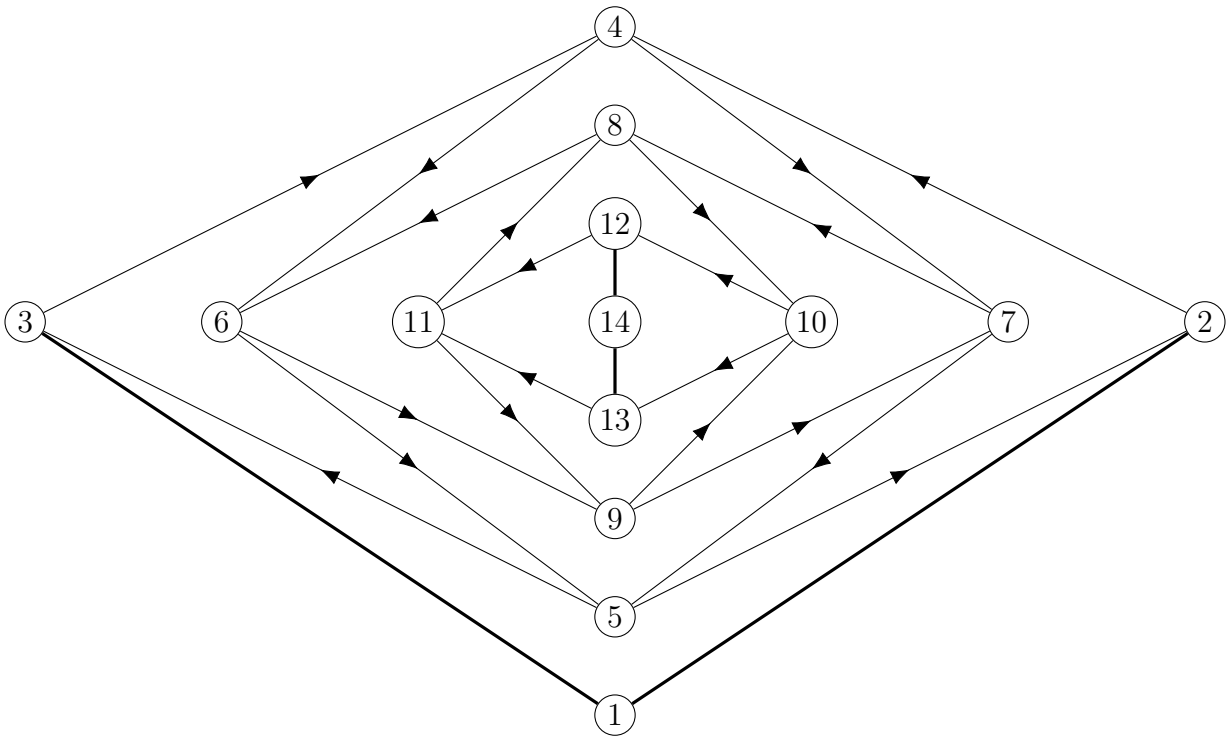


Figure 3.9: Counterexample of order 14 with maximum degree 4

In this example, Bold edges can be oriented arbitrarily.
The graph in Figure 3.9 does not have an AI-partition.

Proof. In this graph we have the following cycles:

a. $4 \to \{6,7\} \to 5 \to \{2,3\} \to 4$

b. $10 \to \{12,13\} \to 11 \to \{8,9\} \to 10$

c. $6 \to 9 \to 7 \to 8 \to 6$

Let us first look at the four cycles in a. Because we cannot remove both 6 and 7 and we cannot remove both 2 and 3 as this would disconnect the graph, we must remove either 4 or 5 to break all these cycles. Removing both of them would disconnect the graph. As we put either 4 or 5 into I, we can no longer put 2,3,6 or 7 into I because then I would no longer be independent. From the cycle in c. we then have that we must put either 8 or 9 into I. Here also we cannot do both because this would disconnect the graph. By putting 8 or 9 into I, we can no longer put 10 or 11 into I without making I not independent. What this also means is that we can no longer break all the cycles from b. because we cannot put both 12 and 13 into I. We conclude that this graph cannot have an AI-partition. □

If we look at the underlying undirected graphs of these subquintic and subquartic counterexamples, they are both unique for their order.

Not bipartite

The example given in Figure 3.5 for this is a minimum counterexample, but it is not unique. There is another counterexample of order 12. Bold edges can be oriented arbitrarily.

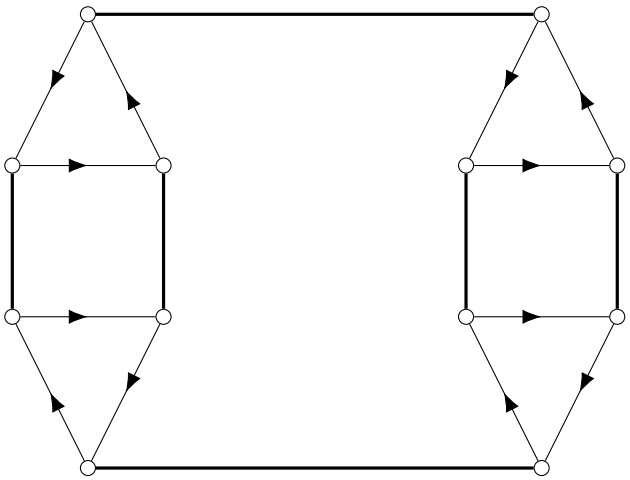


Figure 3.10: Counterexample that is not bipartite

The graph in Figure 3.11 does not have an AI-partition.

Proof. The argument for this is the same as when we consider this as an undirected graph. For every triangle, we must put one of the 3 vertices into I, but doing this will always either disconnect the graph or make I near-independent. The same argument also holds for the counterexample in Figure 3.5. □

Construction for infinite families of counterexamples

Let us take a closer look at some of the counterexamples and analyse their stucture. Looking back at Figure 3.11 we can make an interesting observation. We can split the graph into two subgraphs as follows.

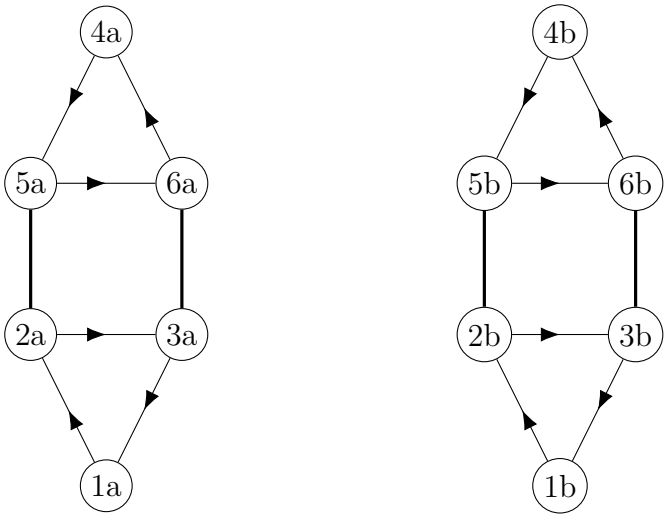


Figure 3.11: Components of non-bipartite counterexample

These components do have multiple possible AI-partitions, but in every one of these partitions either 1a or 4a, 1b or 4b resp. have to be in the independent set I. This causes the whole graph to not have an AI-partition. This is because putting 2 of those 4 vertices in I always disconnects the graph or makes I a near-independent set. Using this we can make an infinite family of counterexamples by connecting any number of these gadgets on a cycle.

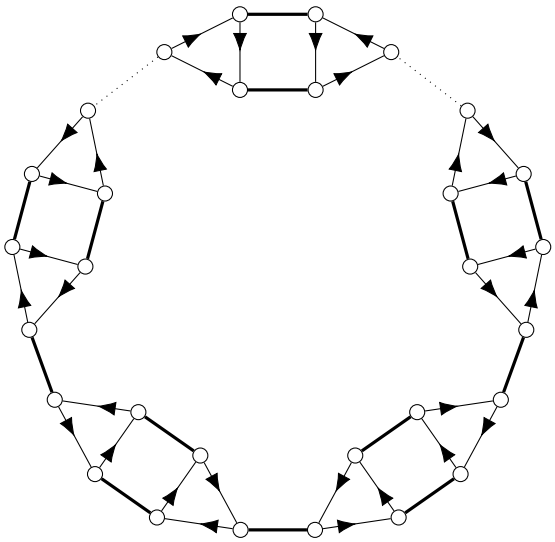


Figure 3.12: Non-bipartite counterexample of order $6k$, with k the amount of gadgets

This method works on some other counterexamples as well, Figure 3.5 (c), Figure 3.5 (a) and Figure 3.6 can be seen as constructions using the following gadgets two times.

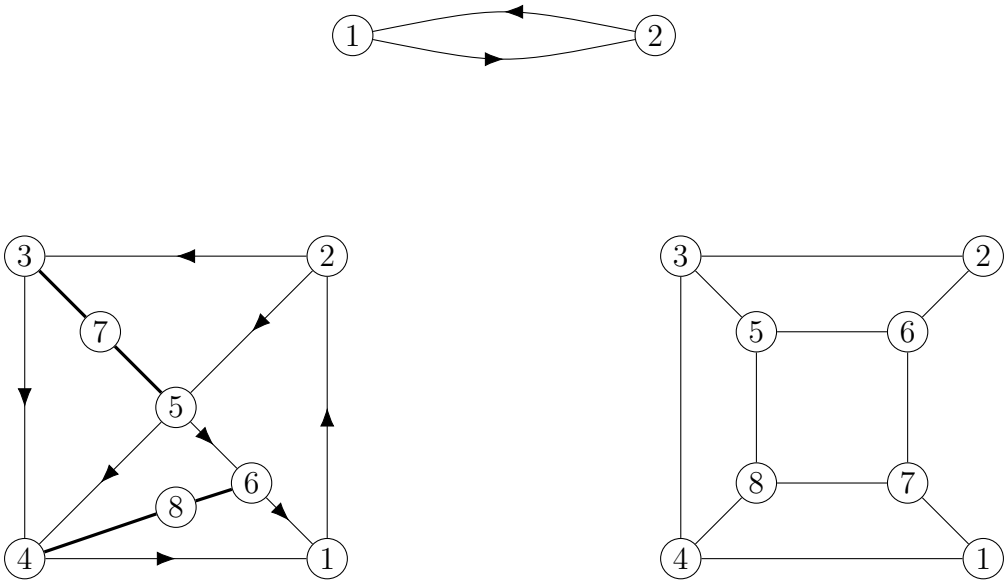


Figure 3.13: Gadgets where, for every AI-partition, one of the vertices of $\{1, 2\}$ is in I

We can construct infinite families of non-oriented counterexamples, counterexamples with maximum degree 4 and counterexamples that are undirected using the gadgets in Figure 3.13.

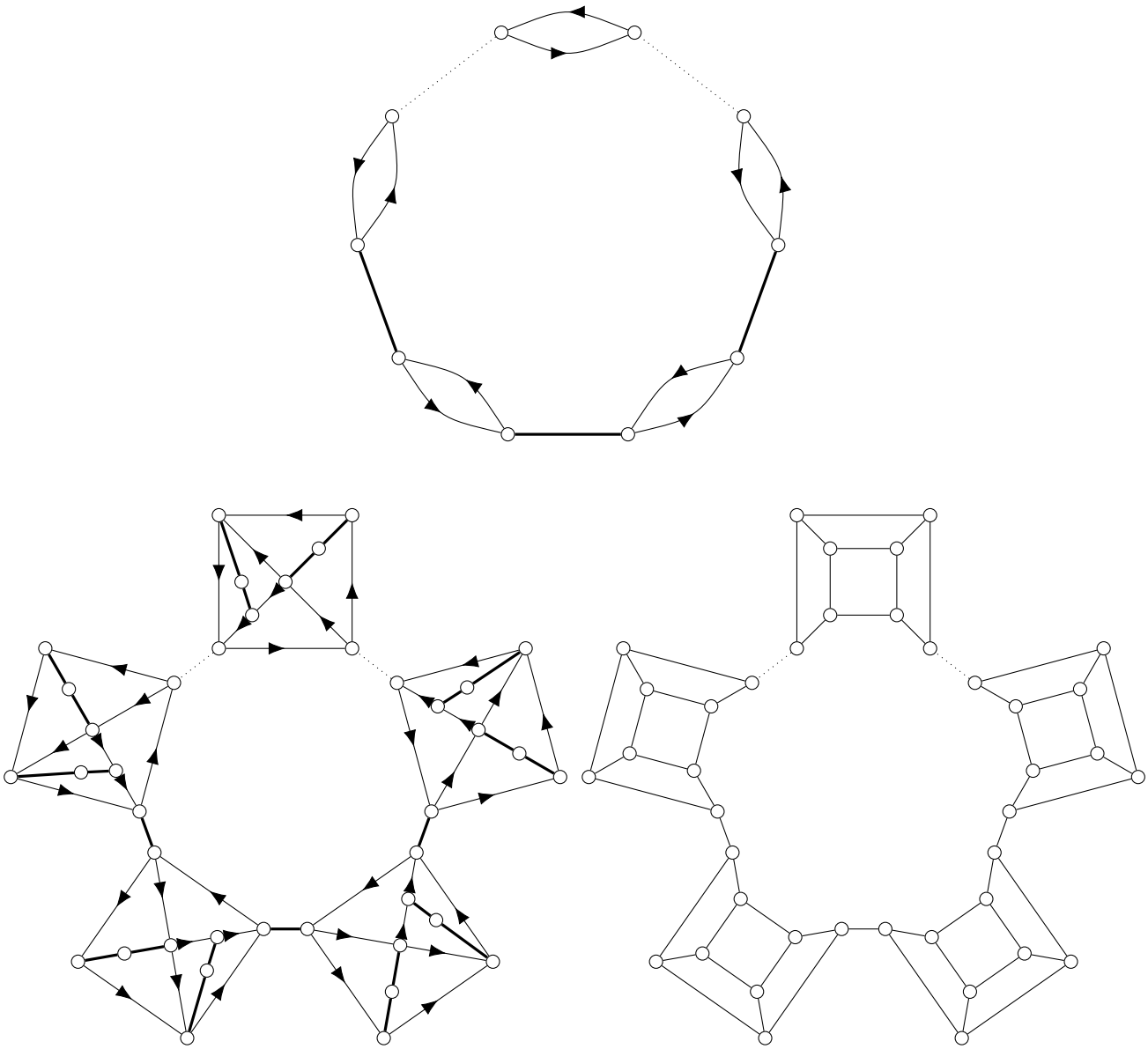


Figure 3.14: Counterexamples of non-oriented graphs, graphs with maximum degree 4 and undirected graphs of order nk, with n the amount of vertices in the gadget and k the amount of gadgets

With these gadgets, the construction always gives a bipartite graph. This is not the case in general. We call the vertices we use to connect the gadgets to each other the connecting vertices and the edges we added in between the connecting edges. In our gadgets we used, every path between their connecting vertices has an odd amount of edges. If we use an even amount of gadgets, any cycle that includes the connecting edges will be an even cycle because we have an even number of odd paths in the gadgets and an even number of connecting edges. If we use an odd number of gadgets, any cycle that includes the connecting edges will also be an even cycle because we have an odd number of odd paths in the gadgets, but the number of connecting edges is also odd.

When we use gadgets that have even paths between the connecting vertices, we do not always get a bipartite graph. In that case we have to use an even number of gadgets with even paths to

construct the infinite family. Of course we can also combine different gadgets in our construction.

non-planar

A counterexample has not yet been found when leaving out planarity. We used the algorithms to check all 2-connected subcubic bipartite graphs, but we also did not find a counterexample by doing this. However, the construction used in Figure 3.14 can give us a possibility to find a non-planar counterexample. If we can find an oriented 2-connected subcubic bipartite graph that contains at least 2 vertices of degree 2 and for all AI-partitions of this graph, one of those vertices is in I, then we can construct a 2-edge-connected (or by Lemma 1.18, 2-connected) counterexample by connecting two of such graphs. This graph would have to be non-planar, so the components are either non-planar or they are planar, but the connecting vertices are not on the same faces.

This is not the only construction we can find in our counterexamples. Take for example the counterexample in Figure 3.8. This time we can split the graph into two components again. A component with only vertex 11 *and a component with all the other vertices.*

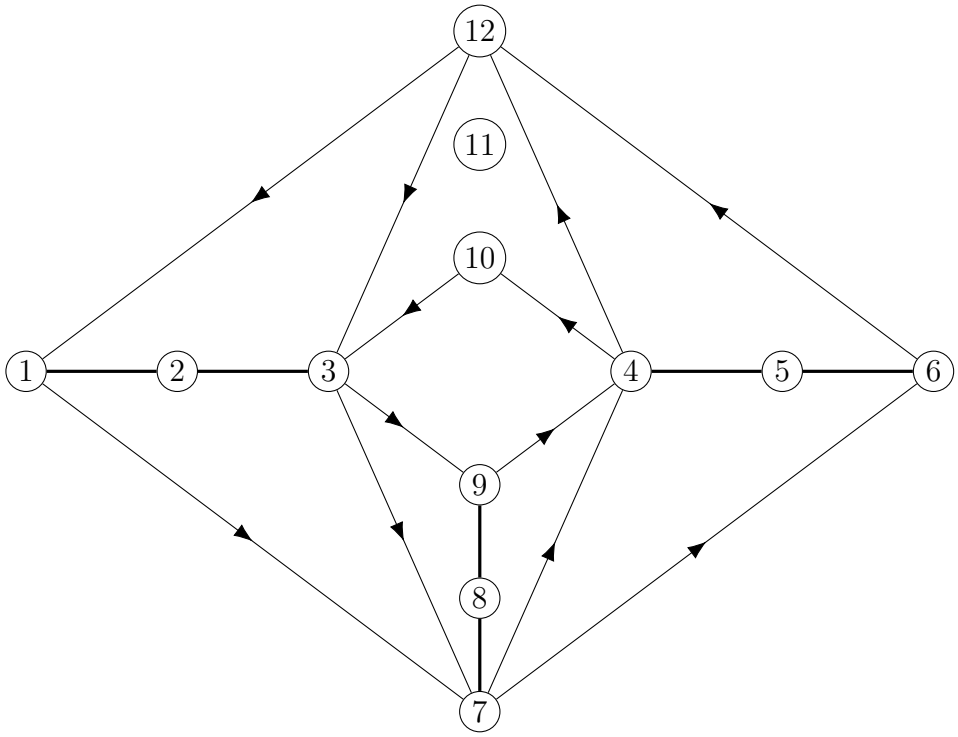


Figure 3.15: Components of subquintic counterexample

The argument here is that the bigger component has a AI-partitions, but we always have $\{10, 12\} \in I$. *Because of this we can connect this component with two edges to any other graph and the result will never have an AI-partition. With this we can strengthen our previous claim and state is as a lemma.*

Lemma 3.18. If there exists an oriented 2-connected subcubic bipartite graph that contains at least 2 vertices u, v with $deg(u), deg(v) \leq 2$ and for all AI-partitions of this graph, either u or v or both are elements of the independent set I then there exists an oriented 2-connected subcubic bipartite graph without AI-partition.

We have not been able to find such a gadget described in Lemma 3.18, but when we consider other values for the maximum degree we can find small examples. In fact if we allow subquartic graphs, with $deg(u), deg(v) \leq 3$ *the smallest examples are only of order 8. An example of these is the oriented gadget given in Figure 3.13 where for every AI-partition either vertex 1 or 2 is in* I*.*

Bibliography

[1] *K. Appel and W. Haken. Every planar map is four colorable. i. discharging.* Illinois Journal of Mathematics, *21(3):429–490, 1977.*

[2] *G. Brinkmann, J. Goedgebeur, and B. D. Mckay. The minimality of the georges-kelmans graph.* Mathematics of computation, *91:1483–1500, 2022.*

[3] *S. Cambie, F. Dross, K. Knauer, H. La, and P. Valicov. Partitions of planar (oriented) graphs into a connected acyclic and an independent set.* arXiv e-prints, *2024.*

[4] *B. Grünbaum. Polytopes, graphs, and complexes.* Bulletin of the American Mathematical Society, *76(6):1131–1201, 1970.*

[5] *M. Jungerman. A characterization of upper-embeddable graphs.* Transactions of the American Mathematical Society, *241:401–406, 1978.*

[6] *K. Knauer and P. Valicov. Cuts in matchings of 3-connected cubic graphs.* European Journal of Combinatorics, *76:27–36, 2019.*

[7] *B. D. McKay. A note on the history of the four-colour conjecture.* Journal of Graph Theory, *72(3):361–363, 2013.*

[8] *R. Nedela, M. Seifrtová, and M. Škoviera. Decycling cubic graphs.* Discrete Mathematics, *347(8):20, 2024.*

[9] *C. Payan and M. Sakarovitch. Ensembles cycliquement stables et graphes cubiques.* Cahiers du Centre d'Etudes de Recherche Operationelle, *17(2-4):319–343, 1975.*

[10] *S. K. Pootheri.* Characterizing and Counting Classes of Unlabeled 2-Connected Graphs. *PhD thesis, University of Georgia, 2000.*

[11] *J. Put. Oriented-ai-partition-search.* https://github.com/JurrePut/oriented-AI-partition-search/blob/main/Fullcode.py, *2025. GitHub repository, accessed June 2025.*

[12] *M.-T. Tsai and D. B. West. A new proof of 3-colorability of eulerian triangulations.* Ars Mathematica Contemporanea, *4(1):73–77, 2011.*

[13] *W. T. Tutte. On hamiltonian circuits.* Journal of the London Mathematical Society, *s1-21(2):98–101, 1946.*

[14] *W. T. Tutte.* Connectivity in Graphs. *University of Toronto Press, 1966.*

[15] *W. T. Tutte. On the 2-factors of bicubic graphs.* Discrete Mathematics, *1(2):203–208, 1971.*

[16] *N. Van Cleemput and C. T. Zamfirescu. Regular non-hamiltonian polyhedral graphs.* Applied Mathematics and Computation, *338:192–206, 2018.*

Appendices

Appendix A

Algorithms

```
from sage.graphs.graph_generators import graphs
from itertools import product
from sage.graphs.independent_sets import IndependentSets
import time
```

```
def find_cycle(G):
    """
    This function gets the cycle from a graph containing precisely 1
       ↪ cycle
    """
    while not all((G.degree(v) > 1 for v in G.vertices())):
        degree1vertices = [v for v in G.vertices() if G.degree(v) ==
           ↪ 1]
        G.delete_vertices(degree1vertices)
    start = G.vertices()[0]
    cycle = [start]
    for i in range(1,len(G)):
        neighbors = G.neighbors(cycle[i-1])
        if neighbors[0] in cycle:
            next_vertex = neighbors[1]
        else:
            next_vertex = neighbors[0]
        cycle += [next_vertex]
    return cycle + [start]
```

```
def is_satisfiable(G,cycles_left):
    """
    This function checks if all the cycles we got from check_partition
    can all be oriented at the same time
    """
    satgraph = Graph()
    for cycle in cycles_left:
        for c in range(len(cycle)-2):
            satgraph.add_edge(f"{cycle[c]}_{cycle[c+1]}", f"{cycle[c
                ↪ +1]}_{cycle[c+2]}")
            satgraph.add_edge(f"{cycle[c+2]}_{cycle[c+1]}", f"{cycle[c
                ↪ +1]}_{cycle[c]}")
    components = satgraph.connected_components()
    number_of_components = len(components)
    if number_of_components == 1:
        return False
    elif number_of_components == 0:
        return [(),1]
    assignments = [True] + [None]*(number_of_components-1)
    component_map = {}
    for i, component in enumerate(components):
        for node in component:
            component_map[node] = i  # Assign each node an index
                ↪ corresponding to its component
    for v in satgraph.vertices():
        parts = v.split('_')
        opposite_v = f"{parts[1]}_{parts[0]}"
        map_v = component_map.get(v)
        map_oppv = component_map.get(opposite_v)
        if map_v == map_oppv:
            return False
        if assignments[map_v] == True and assignments[map_oppv] ==
            ↪ None:
            assignments[map_oppv] = False
        elif assignments[map_v] == None and assignments[map_oppv] ==
            ↪ None:
            assignments[map_v] = True
            assignments[map_oppv] = False

    edges = []
    for v in satgraph.vertices():
        if assignments[component_map.get(v)] == True:
            parts = v.split('_')
            edges += [(f"{parts[1]}_{parts[0]}",component_map.get(v)
                ↪ //2)]
    return [edges,number_of_components/2]
```

```
def check_partition_oriented(G, independent_sets):
    '''
    This function finds AI-partitions for an oriented graph
    '''
    vertices = G.vertices()
    for independent_set in independent_sets:
        tree_set = set(vertices) - set(independent_set)
        tree_subgraph = G.subgraph(tree_set)

        # Check if the partition gives us an acyclic graph
        if tree_subgraph.is_directed_acyclic():
            g = tree_subgraph.union(G.subgraph(independent_set))
            return g  # This graph can be partitioned
    return False  # No valid partition found
```

```
def check_partition(G):
    '''
    This function find all partitions that leave one cycle
    and returns those cycles,
    stops if it finds an AI-partition
    '''
    vertices = G.vertices()
    onecycles = []
    valid_independent_sets = []
    used_independent_sets = []

    for independent_set in IndependentSets(G):
        tree_set = set(vertices) - set(independent_set)
        tree_subgraph = G.subgraph(tree_set)
        # Check if the partition gives us a graph with 1 or 0 cycles
        if tree_subgraph.is_connected():
            if len(tree_subgraph) == tree_subgraph.size():
                onecycle = find_cycle(tree_subgraph)
                used_independent_sets += [independent_set]
                if onecycle not in onecycles:
                    onecycles += [onecycle]
            elif len(tree_subgraph) == tree_subgraph.size()+1:
                return False
            else:
                valid_independent_sets += [independent_set] #only
                    ↪ connected partitions have to be checked again
                    ↪ later
    valid_filtered_independent_sets = [] #subsets of sets that result
       ↪ in one cycle cannot give an AI-partition
    for independent_set in valid_independent_sets:
        if not any(set(independent_set) < set(other) for other in
           ↪ used_independent_sets):
            valid_filtered_independent_sets.append(independent_set)
    return [onecycles,valid_filtered_independent_sets]
```

```
def main(G):
    graph_information = check_partition(G)
    if graph_information is not False:
        satisfy = is_satisfiable(G, graph_information[0])
        if satisfy is not False:
            valid_independent_sets = graph_information[1]
            unassigned = G.size() - len(satisfy[0])
            print('unassigned',unassigned, 'of', G.size())
            for c in iter(product([True, False], repeat=satisfy[1]-1))
                ↪ :
                usededges = []
                c = [True] + list(c)
                for maps in satisfy[0]:
                    if c[maps[1]] == True:
                        usededges += [maps[0]]
                    else:
                        parts = maps[0].split('_')
                        usededges += [f"{parts[1]}_{parts[0]}"]

                true_edges = [e for e in G.edges() if f"{e[0]}_{e[1]}"
                    ↪ in usededges]
                false_edges = [e for e in G.edges() if f"{e[1]}_{e[0]}
                    ↪ " in usededges]
                unassigned_edges = [e for e in G.edges() if e not in
                    ↪ true_edges and e not in false_edges]
                d0 = [e for e in true_edges] + [(e[1],e[0]) for e in
                    ↪ false_edges]

                for a in iter(product([True, False], repeat=unassigned
                    ↪ )):
                    u = 0
                    diedges = []
                    for e in unassigned_edges:
                        if a[u]:
                            diedges.append((e[0],e[1]))
                        else:
                            diedges.append((e[1],e[0]))
                        u += 1
                    diedges.extend(d0)
                    d = DiGraph(diedges)
                    if all((0 < d.out_degree(v) and 0 < d.in_degree(v)
                        ↪ ) for v in d.vertices()) or connectedness <=
                        ↪ 1:
                        oplossing = check_partition_oriented(d,
                            ↪ valid_independent_sets)
                        if oplossing is False:
                            return d
    return False
```

```
minvert = int(input('minimum number of vertices'))
maxvert = int(input('maximum number of vertices'))
mindegree = int(input('minimum degree'))
maxdegree = int(input('maximum degree'))
connectedness = int(input('minimum connectedness'))
planar = bool(input('planar graphs only? (True/False)'))

numberofgraphs = 0
for n in range(minvert,maxvert+1):
    numberofgraphs = 0
    start_time = time.time()
    print('n =',n)
    checked = 0
    for G in iter(graphs.nauty_geng(f' -d{mindegree} -D{maxdegree} -c
       ↪ -b {n}')):
        '''
        Generate undirected graphs of order n with given parameters,
           ↪ some options are:
        -d{mindegree}: only generate graphs with this as minimum
           ↪ degree
        -D{maxdegree}: only generate graphs with this as maximum
           ↪ degree
        -c: generate connected graphs
        -b: generate bipartite graphs
        -t: generate trianglefree graphs
        '''
        if G.vertex_connectivity() < connectedness:
            continue

        if planar:
            if not G.is_planar():
                continue

        oplossing = main(G)
        if oplossing is not False:
            show(oplossing)
            print('has no partition')
            for v in oplossing.vertices():
                print(f"{v}:{oplossing.neighbors_out(v)}", end=", ")
            print("")
            print('number of edges', oplossing.size())
            numberofgraphs += 1
        checked += 1
    end_time = time.time()
    print('graphs checked',checked)
    print('number of graphs found',numberofgraphs)
    print(f"Time taken = {end_time - start_time:.2f} seconds")
```

Faculty of science
Geel Huis, Kasteelpark Arenberg 11 bus 2100
3001 Heverlee, België
tel. + 32 16 32 14 01
www.kuleuven.be