Sparse Cuts, Matching-Cuts and Leafy Trees in Graphs

Paul S. Bonsma

The research presented in this thesis was carried out at the Discrete Mathematics and Mathematical Programming Group, Department of Applied Mathematics, University of Twente, Enschede, The Netherlands.

The graph shown on the cover of this thesis is one of the graphs used in the $\mathcal{NP}$-completeness proofs of Chapter 3. The bold edges indicate a *matching-cut* in this graph, consisting of two of the six minimal matching-cuts possible in this graph. The cut consisting of the eleven dashed edges separates the graph into two parts with 56 vertices. This is a *sparsest cut* of the graph with density 0.0035. The 69 open vertices form the *leaf set* of a *spanning tree* of this graph. We do not know whether this number of leaves is maximum. Readers with plenty of spare time, that want to get an impression of the hardness of this problem, are invited to prove or disprove this.

Printed by PrintPartners Ipskamp, Enschede.

Sparse Cuts, Matching-Cuts and Leafy Trees in Graphs,
by Paul S. Bonsma, University of Twente, Enschede, 2006.
ISBN: 90-365-2370-2

SPARSE CUTS, MATCHING-CUTS AND LEAFY TREES IN GRAPHS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. W.H.M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op woensdag 28 juni 2006 om 15.00 uur

door

Paul Simon Bonsma

geboren op 17 maart 1976

te Heerenveen

Dit proefschrift is goedgekeurd door de promotoren

Prof. dr. ir. H.J. Broersma en
Prof. dr. G.J. Woeginger

Preface

This thesis is the result of my research that was carried out at the DMMP group at the university of Twente, under supervision of Hajo Broersma and Gerhard Woeginger, starting in 2001.

I would like to thank Hajo Broersma for offering me this opportunity, and for his detailed checking of all of my papers. I also want to thank him for the enjoyable times we had, first during the coffee breaks in Enschede, and later during my stay in Durham. I want to thank Gerhard Woeginger for his valuable suggestions, and for introducing me to many of my research subjects, some of which became part of this thesis.

I want to thank Prof.dr. C. Hoede, dr.ir. J.P.M. van den Heuvel, Prof.dr. D. Kratsch, dr. H.L. Bodlaender, and Prof.dr.ir. E.C. van Berkum, for participating in my graduation committee. I am thankful to Andrzej Proskurowski for the many discussions and suggestions on the topic of matching-cuts.

I enjoyed the pleasant working environment at the DMMP chair. I want to thank everyone from DMMP for this, and in addition I want to thank Dini and Johann for their help. The many Ph.D. students that worked here during these five years have also contributed to my enjoyment at work, and also outside of it, at courses and workshops. I want to thank Adriana, Tobias, Tim, Jacob Jan and all other (former) Ph.D. students at DMMP for this.

I thank my parents, my friends, and Cindy's parents for their support and interest. I would also like to thank my brother Erwin; even though at times I thought only my supervisors and reviewers would read my papers, he read them simply out of curiosity. Most of all I want to thank Cindy, for her love and continuous support, even while she was very busy with her own research.

Paul Bonsma May 2006

Contents

Chapter 1

Introduction

1.1 Motivation

The design and analysis of telecommunication networks has become an important application area for graph theory. In this section we give an informal introduction to the problems studied in this thesis, formulated in terms of telecommunication applications. In the subsequent sections the problems will be defined more formally. Our examples do not necessarily reflect the way these problems are applied in practice, but illustrate in a simple way some relationships with practical network design problems.

A (telecommunication) *network* consists of a number of *nodes*, that need to communicate with each other through *links* between node pairs. In general, communication should be possible between every pair of nodes, but it is often inefficient or too costly to add direct links between every node pair. So only between certain pairs (direct) links are added. A node a joined by a link to the node b is called a *neighbor* of b. Neighbors communicate through their common link. If a node a needs to communicate with a node b that is not one of its neighbors, the message is sent to a neighbor of a, which in turn sends it to a neighbor, and so on, until the message reaches node b. So the message follows a *path* through the network. The intermediate nodes on this path are called *relay nodes*. If every node pair can communicate in this direct or indirect way, the network is called *connected*. The choice of which links to create determines many properties of the network, and is an important part of the design of the network. For this choice, considerations like costs and reliability are important. We will come back to this later.

To model networks mathematically, graphs can be used. A graph consist of a set of *vertices*, and a set of *edges* joining certain pairs of vertices. When used to model networks, the vertices correspond to the network nodes, and the edges correspond to the links.

Communication networks as described above occur in many forms: the links can be metal wires, optical fibers or wireless links, the nodes can vary from

simple, tiny sensors to supercomputers, and the scale of the network can vary from multiple chips communicating in a single computer, to computer networks in a building, to optical networks the size of a continent. Every variant leads to different cost models and to new questions, which in turn lead to a wealth of graph theoretic problems. We will describe three of them in more detail, as well as their relation to the results in this thesis.

Max-Leaf Spanning Tree Suppose the locations of nodes are known, and it is known between which pairs links may be added (e.g. only between nodes that are spatially close to each other, because of link cost or signal strength considerations). See Figure 1.1(a) for an example showing nodes and possible links between them. Since adding links is costly, we want to add a minimum number of links, such that all nodes can communicate with each other. This will lead to a *spanning tree* shape, as shown in Figure 1.1(b). In this network, no links need to be added, since there is a path between every node pair. No links can be removed without destroying this property. Nodes with precisely one neighbor in this network are called *leaves*, and the other nodes are called *routing nodes*. Unlike the leaves, routing nodes will be used to relay messages from other nodes. Implementing this added functionality to the nodes adds another cost. Hence we want to minimize the number of routing nodes, or alternatively, to maximize the number of leaves. The corresponding graph theoretic problem is known as the *Max-Leaf Spanning Tree* problem.

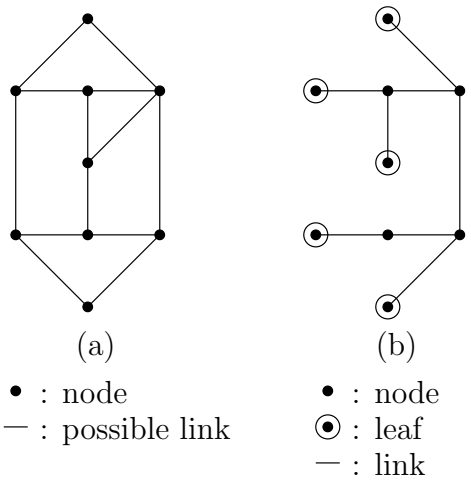


Figure 1.1: Link possibilities, and a corresponding connected network with minimum number of links

A closely related problem is the following (for more details see [47]): small sensor nodes are distributed through a space, for instance an office building. These nodes make measurements (smoke detection, temperature, etc.), and can

communicate wirelessly with nearby nodes. The measured data has to be sent to a central node that collects the data, though not every node can reach this central node directly. So some nodes will only make measurements and send their data periodically, but other nodes need to 'listen' to these transmissions and relay them. Because of this, nodes of the latter type consume much more energy, and therefore we want to minimize the number of nodes of this type. So we want to select a small subset of the nodes as relay nodes, such that all other nodes can send to at least one of these nodes (the set is *dominating*), and such that every relay node can reach the central node via other relay nodes (the set is *connected*). The corresponding graph theoretic problem is known as the *Minimum Connected Dominating Set* problem. This problem is closely related to Max-Leaf Spanning Tree; the non-leaf nodes in a spanning tree form a connected dominating set. Even though for Minimum Connected Dominating Set it is not important how the links are established, both problems are equally hard to solve from an algorithmic viewpoint.

In Chapter 5 and Chapter 6 we present new methods for finding spanning trees with many leaves, and small connected dominating sets.

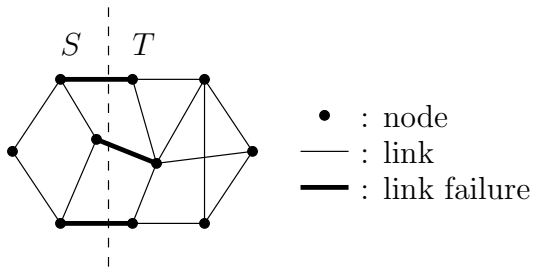


Figure 1.2: An isolated set of failures determining the weakest part of the network

Sparse cuts and matching-cuts Another important aspect in designing telecommunication networks is *reliability*. Elements of the network (nodes or links) can fail, and it is desirable to design the network in such a way that the most likely failures have little impact on the global performance of the network. There are many ways to express the reliability of networks. We explain one of those that is related to the results in this thesis.

Consider a subset M of links. Let x_M denote the number of node pairs that cannot communicate anymore, if all the links in M fail simultaneously. We say that the set M that has the highest ratio $x_M/|M|$ between the number of disconnected node pairs and the number of links in the set, determines the weakest part of the network. This is because for larger numbers of links, it is less likely that they all fail simultaneously. It can be shown that the set M that determines this weakest part, always separates the network into two connected parts. The node sets of these parts will be called S and T. See Figure 1.2 for an

example of a network and a set of three links that determines its weakest part, and the corresponding sets S and T. The partition $\{S, T\}$ that maximizes the ratio between the number of disconnected pairs (which is $|S||T|$ when the two parts are connected) and the number of links between S and T is called a *sparsest cut* of the network. So according to our criteria, a sparsest cut determines the weakest part of the network. In Chapter 2, we show where the sparsest cuts occur for some special classes of graphs. One of these classes is the class of product graphs, which are frequently used as a structure for telecommunication networks [5, 19]. In Chapter 2, it is also shown that sparsest cuts are closely related to all-to-all flows in the graph, another important application of graph theory to telecommunication.

A set of failing links is called an *isolated set* if no node is directly linked by more than one of these failing links. The set of failing links illustrated in Figure 1.2 is an example of an isolated set. A network is called *immune to isolated link failures* if there is no isolated set that can disconnect the network [25]. The graph theoretic concept that corresponds to an isolated set is a *matching*. A matching that disconnects the graph when removed is called a *matching-cut*. A graph without matching-cuts is called *matching immune*. In Chapter 3 we discuss the problem of deciding whether a given graph is matching immune. Another relevant question is how to construct networks that are immune to isolated link failures, using a minimum number of links (and thus with minimum costs). In [26], a class of matching immune graphs is defined, with minimum number of edges, for each number of vertices. In Chapter 4 we show that this definition characterizes all such graphs.

1.2 Notation and terminology

In this section we give definitions and notations that will be used often, or that are non-standard. For standard graph theoretic terminology not defined here we refer to [9] or [20]. For more on complexity of algorithms, we refer to [31].

Our basic mathematical notations are fairly standard. We distinguish between $\subseteq$ and $\subset$; these symbols denote subset resp. proper subset. We write $S-x$ instead of $S\backslash\{x\}$, and $S+x$ instead of $S\cup\{x\}$, when this is needed to improve the readability. A set S is *minimal* for a property ϕ if S satisfies ϕ, and has no proper subset satisfying ϕ. $S \subseteq X$ is *minimum* for property ϕ if it satisfies ϕ and there is no set $T \subseteq X$ with $|T| < |S|$ that also satisfies ϕ. *Maximal* and *maximum* are defined analogously.

Graphs There are two main ways to define graphs, depending on whether edge labels are used. Since we consider multi-graphs and use contractions in some of the chapters, we need to introduce the more detailed definition using edge labels: a *(multi-) graph* is a triple (V, E, ψ), where V and E are disjoint finite sets called resp. the *vertex set* and *edge set*, and ψ is an *incidence function* that maps the edges to multi-sets of exactly two vertices. If $\psi(e) = \{v, v\}$, then e is called a *loop*. If there are distinct $e \in E$ and $f \in E$ with $\psi(e) = \psi(f)$, then

e and f are called *parallel edges*. If parallel edges exist between a vertex pair, then the set of all of these edges is called a *multi-edge*. A multi-edge consisting of two edges is also called a *double edge*. Instead of $\{u, v\}$ we will often write uv or equivalently vu for edges. Instead of $\psi(e) = uv$ we will write $e = uv$ for short. If a graph has no multi-edges or loops, it is called a *simple graph*. Note that the graphs we defined are undirected.

Consider the multi-set $E' = \cup_{e \in E} \psi(e)$. If edge labels do not matter, a graph can alternatively be denoted by (V, E'). In this case, E' is called the edge set. In most cases we will use the latter notation for graphs (also for multi-graphs), and thus denote edges by their end vertices, as long as there is no cause for confusion. Note that if the graph is simple, E' is a set, and its elements are also sets.

For a graph G, $V(G)$ denotes its vertex set, and $E(G)$ denotes its edge set. The *order* of G is $|V(G)|$, and the *size* of G is $|E(G)|$. If $e = uv$, u and v are called the *end vertices* of e, and e is an edge *between* u and v. Then u and v are called *adjacent*, and u is called a *neighbor* of v (and vice versa). Edge e is *incident* with u and v, and vice versa. If two edges e and f are both incident with the same vertex, e and f are *adjacent*. For a vertex $v \in V(G)$, $N_G(v)$ denotes the set of all neighbors of v in graph G. If it is clear which graph is considered, we write $N(v)$ instead; the same holds for other notations using graphs as a subscript. We use $d_G(v) = |N_G(v)|$ to denote the *degree* of v. A vertex with degree 0 is called an *isolated vertex*, and a vertex with degree 1 a *leaf*. $L(G)$ denotes the set of leaves of G. $\delta(G) = \min\{d_G(v) : v \in V(G)\}$ is the *minimum degree* of G, and similarly $\Delta(G) = \max\{d_G(v) : v \in V(G)\}$ is the *maximum degree* of G. If there is no cause for confusion, we write δ resp. Δ instead. A graph with $\delta = \Delta = k$ is called *k-regular*. A 3-regular graph is also called a *cubic graph*.

Graphs are often represented by a drawing. Formally, in a *drawing* of a graph, vertices are mapped to distinct points in the plane, and edges are mapped to curves between the points corresponding to their end vertices. If different curves only meet in common end points, the drawing is called a *planar embedding* of the graph. A graph that allows a planar embedding is called *planar*.

The graph obtained by adding an edge e to a graph G (and in case of labeled edges, updating the incidence function such that the result remains a graph) is denoted by $G + e$. Similarly, $G - e$ denotes the deletion of edge e. The graph obtained by adding a vertex v is denoted by $G + v$. The operation of *deleting* a vertex v from G consists of removing v from the vertex set, and removing all edges incident with v from the edge set. The resulting graph is denoted by $G - v$. Similarly, we write $G - S$ ($G + S$) for the graph obtained by removing (adding) a vertex or edge set S from (to) the graph. The operation of *subdividing* an edge uv consists of removing uv, adding a new vertex w, and adding edges uw and vw. *Suppressing* a vertex w of degree two is the inverse of this operation.

To define contractions and related operations we use the incidence function again. The *identification* of vertices u_1 and u_2 consists of the following steps: introduce a new vertex u, and in every image of ψ replace u_1 and u_2 by u. Delete u_1 and u_2. Occasionally we will use the label u_1 or u_2 also for the new

vertex. The *contraction* of an edge e with $\psi(e) = \{u_1, u_2\}$ consists of removing e and identifying u_1 and u_2. The contraction of e in G is denoted by $G \cdot e$. A contraction in a simple graph may result in parallel edges, and a contraction in a graph with parallel edges may result in loops. A contraction of a loop corresponds to the deletion of this loop (and relabeling the vertex). An edge expansion can be seen as the reverse of a contraction. So the *edge expansion of u into u_1u_2* consists of the following steps: introducing new vertices u_1 and u_2, introducing an edge e with $\psi(e) = u_1u_2$, replacing every occurrence of u in images of ψ by u_1 or u_2, and deleting u. Note that there is only one way to contract a particular edge of a graph (apart from the resulting vertex label), but in general there are many ways to expand a vertex into an edge since we may choose whether to replace u by u_1 or by u_2. An edge expansion of u into u_1u_2 is called a *non-trivial edge expansion* if $d(u_1) \geq 2$ and $d(u_2) \geq 2$ in the resulting graph. We defined these operations using the incidence function to make clear that edges in $G \cdot e$ correspond uniquely to edges in G, even though the labels of their end vertices may change. The remaining definitions are again formulated in terms of graphs without edge labels.

(V', E') is a *subgraph* of the graph $G = (V, E)$ if it is a graph and $V' \subseteq V$ and $E' \subseteq E$. It is a *spanning subgraph* if $V' = V$. If $M \subseteq E$, then

$$G[M] = (\{v \in V : \exists uv \in M\}, M)$$

is the subgraph of G *induced by* M. If $S \subseteq V$, then

$$G[S] = (S, \{uv \in E : u \in S \wedge v \in S\})$$

is the subgraph of G *induced by* S. A graph H is called an *induced subgraph* of G if $H = G[S]$ for some $S \subseteq V$, and an *edge induced subgraph* of G if $H = G[M]$ for some $M \subseteq E$. A graph G is *minimal* for a property ϕ if G satisfies ϕ, and has no subgraph satisfying ϕ, other than G itself. *Maximal* is defined analogously.

The graph with an empty vertex set is called the *empty graph*. A graph is called *complete* if between every pair of vertices one edge is present, and it has no loops. K_n denotes a complete graph of order n. K_3 is also called a *triangle*. A *diamond* is a graph that can be obtained by deleting an edge from a K_4. A *cycle* is a minimal 2-regular graph. C_n denotes a cycle of order n, also called an *n-cycle*. (C_1 and C_2 are not simple.) The *wheel* W_n is obtained by adding a new vertex v to C_{n-1}, and one edge between v and u, for all vertices $u \neq v$. A *path* is a graph (V, E) of the form $V = \{v_0, v_1, \ldots, v_k\}$, $E = \{v_0v_1, v_1v_2, \ldots, v_{k-1}v_k\}$, with all v_i distinct, for $k \geq 0$. Here v_0 and v_k are called the *end vertices* of the path, and the other vertices the *internal vertices*. A path with end vertices u and v is also called a *(u, v)-path*. Note that K_1 is also a path. P_n denotes a path of order n. The *length* of a path or cycle is its number of edges. In simple graphs, paths and cycles can be characterized by an ordered list of their vertices, such that successive vertices in the list are adjacent. Paths and cycles will sometimes be denoted by such a list, and proofs will use this natural vertex order. A graph $G = (V, E)$ is *bipartite* if there is a partition $\{A, B\}$ of V such that $G[A]$ and $G[B]$ contain no edges. Then $\{A, B\}$ is called a *bipartition* of G. $K_{m,n}$ denotes

a *complete bipartite graph*, which is a simple graph with bipartition $\{A, B\}$ such that $|A| = m$, $|B| = n$, and if $u \in A$ and $v \in B$, then $uv \in E$. $K_{1,n}$ is also called a *star*, and $K_{1,3}$ is called a *claw*.

The Cartesian product graph or *product graph* $G \times H$ of two graphs G and H is defined as follows: $G \times H$ has vertex set $V(G) \times V(H)$, and an edge set containing all edges of the form

$$\{(u, x), (v, x)\}$$

if $uv \in E(G)$ and $x \in V(H)$, and

$$\{(u, x), (u, y)\}$$

if $xy \in E(H)$ and $u \in V(G)$. The *n-cube* Q_n is defined as follows: $Q_1 = K_2$, and $Q_n = K_2 \times Q_{n-1}$ (with possibly different vertex labels).

G is *connected* if for every pair of vertices u and v, G has a (u, v)-path (as a subgraph), and *disconnected* otherwise. A *tree* is a connected graph without cycles. $S \subseteq V(G)$ is called a *connected set* if $G[S]$ is connected. A *component* of a graph G is a maximal connected subgraph of G. $S \subset V(G)$ is a *vertex cut* of G if $G - S$ is disconnected; if $|S| = k$, S is also called a *k-vertex cut*. G is *k-connected* if $k < |V(G)|$ and G has no l-vertex cut with $l < k$. If $\{v\}$ is a vertex cut, then v is called a *cut vertex* of G.

For two disjoint non-empty sets $S \subset V$ and $T \subset V$, $[S, T]$ denotes the set of edges of G with one end vertex in S and one end vertex in T. An *edge cut* of G is a set $M \subseteq E$ such that $M = [S, \overline{S}]$ for some $S \subset V$, $S \neq \emptyset$, where $\overline{S}$ denotes $V \backslash S$. If $\{e\}$ is an edge cut, e is called a *bridge*. If a set $S \subseteq V$ exists such that $M = [S, \overline{S}]$ and $A \subseteq S$ and $B \subseteq \overline{S}$ ($u \in S$ and $v \in \overline{S}$), then the edge cut M is said to *separate* A and B (u and v). Both vertex cuts and edge cuts will be called *cuts* if there is no cause for confusion.

$M \subseteq E(G)$ is a *matching* if no two edges in M are adjacent. A matching M *saturates* $A \subseteq V(G)$ if every vertex of A is incident with an edge in M. A matching M in a graph G is *perfect* if all vertices of G are saturated by M. $S \subseteq V(G)$ is a *dominating set* if all vertices in G are either in S, or adjacent to a vertex in S. A spanning path of G is also called a *Hamilton path*, and a spanning cycle a *Hamilton cycle*. A graph with a Hamilton cycle is called *Hamiltonian*. The *girth* of a graph is the minimum length of a cycle in the graph.

Algorithms An *algorithm* is a well-defined procedure that, given an input satisfying certain rules, will terminate and return an output. We consider algorithms for two types of problems, which we will now introduce in a slightly informal way.

An *optimization problem* consists of a set of *instances*, a set of *feasible solutions* for each instance, an *objective function* that assigns a real value (the *objective value*) to a combination of an instance and a feasible solution, and a *goal* which is either minimization or maximization. An *optimal solution* for an instance of an optimization problem with minimization (maximization) as its goal is a feasible solution with minimum (maximum) objective value among all

feasible solutions. (All problems we study are non-degenerate and have optimal solutions.) Algorithms for optimization problems can do different things: ideally, when given an instance to the problem, the algorithm outputs an optimal solution, or the objective value of an optimal solution. But since execution speed is also an important consideration in algorithm design, often other types of algorithms are studied. Some algorithms output a feasible solution x such that the ratio between the objective value of x and the objective value of an optimal solution is bounded in some way (approximation algorithms). In this thesis we study algorithms that output a feasible solution of which the objective value is bounded by a function on the instances. (The value of this function is closely related to the size of the instance.)

A *decision problem* consists of a set of instances, and a YES/NO *question* about these instances. An instance is a *YES-instance* (*NO-instance*) for the problem if the answer to the question is YES (NO). Two instances are called *equivalent* if they are both YES-instances or both NO-instances. This notion is important since often we are able to prove that instances are equivalent without knowing whether they are YES-instances or NO-instances. An algorithm is an algorithm for the decision problem if given an instance for the problem, the output is the correct answer to the question. A special kind of decision problems exists that is closely related to optimization problems: here the question is whether a given set of feasible solutions is non-empty. Given an optimization problem, we obtain such a decision problem by asking the question 'does the instance have a feasible solution with objective value at most (at least) x?'.

For an introduction to complexity of algorithms (instance encodings, polynomial time algorithms, the problem class $\mathcal{NP}$, $\mathcal{NP}$-completeness), we refer to [31].

1.3 An overview of the results

In Chapter 2 sparsest cuts are studied. The *density* of an edge cut $[S, \overline{S}]$ is defined as $d(S, \overline{S}) = \frac{|[S,\overline{S}]|}{|S||\overline{S}|}$. An edge cut with minimum density among all edge cuts of the graph is called a *sparsest cut*. The density of a sparsest cut of a graph G is denoted by $d(G)$. Sparsest cuts were studied in [43]. In [43], it was shown that finding a sparsest cut is $\mathcal{NP}$-hard. In Chapter 2 we characterize sparsest cuts for three graph classes. For product graphs $G \times H$, we show that a sparsest cut exists that corresponds directly to a sparsest cut of G or a sparsest cut of H.

Theorem *If $M_G \subseteq E(G)$ is a sparsest cut of a graph G, and $M_H \subseteq E(H)$ is a sparsest cut of a graph H, then one of the following edge sets is a sparsest cut of $G \times H$:*

$$\{\{(u,x),(v,x)\} : uv \in M_G, x \in V(H)\},$$

$$\{\{(u,x),(u,y)\} : xy \in M_H, u \in V(G)\}.$$

The sparsest cut density of $G \times H$ *is* $d(G \times H) = \min\{d(G)/|V(H)|, d(H)/|V(G)|\}$.

A graph G is a *unit interval graph* if a function $I : V(G) \to \mathbb{R}$ exists such that $uv \in E(G)$ if and only if $I(u) - 1 \leq I(v) \leq I(u) + 1$. I is called a *unit interval representation* of G. We prove the following result for sparsest cuts in unit interval graphs.

Theorem *Every unit interval graph* G *with unit interval representation* I *has a sparsest cut* $[S, \overline{S}]$ *such that for all* $u \in S$, $v \in \overline{S}$, $I(u) \leq I(v)$.

This result immediately leads to a polynomial time algorithm to find a sparsest cut for these graphs. The third graph class we study is the class of complete bipartite graphs.

Theorem *Let* $m \leq n$ *and* $n \geq 2$. *The density of a sparsest cut of* $K_{m,n}$ *is* $\min\{\frac{1}{2}, \frac{m}{n+m-1}\}$.

If $n > m$ then a sparsest cut can be obtained by separating one vertex on the larger side of the bipartition from the rest of the vertices. If $n = m$ it can be obtained by separating two vertices, one from either side of the bipartition, from the rest of the vertices. When $m \leq n \leq m + 1$, in addition to these cuts many other sparsest cuts exist.

To prove these three results, entirely different techniques are used, which together give a good overview of the techniques that are available to prove results of this kind. To prove the result for product graphs, a well-known duality result between sparsest cuts and concurrent flows is used.

A *matching-cut* is an edge cut that is also a matching. In Chapter 3, the Matching-Cut problem is studied, which is the problem of deciding whether a given graph has a matching-cut. Chvátal [17] studied this problem under the name of the Decomposable Graph Recognition problem, and proved the problem to be $\mathcal{NP}$-complete for graphs with maximum degree four, and gave a polynomial time algorithm for graphs with maximum degree three. Patrignani and Pizzonia [48] also proved the $\mathcal{NP}$-completeness of the problem using a different reduction, unaware of Chvátal's result. They also posed the question whether the Matching-Cut problem is $\mathcal{NP}$-complete for planar graphs. In Chapter 3 an affirmative answer is given, that generalizes Chvátal's result:

Theorem *Matching-Cut is* $\mathcal{NP}$*-complete for planar graphs with maximum degree four, and for planar graphs with girth five.*

The reduction is from Planar Graph 3-Colorability and differs from the reductions used to prove the earlier results. In addition, for certain graph classes polynomial time algorithms to find matching-cuts are described. These classes include claw-free graphs, co-graphs, graphs with fixed bounded treewidth, in

particular outerplanar graphs, and planar graphs with girth at least six. (For definitions of these classes, see Chapter 3.)

In Chapter 4, graphs without a matching-cut are studied, which are called *(matching) immune*. Farley and Proskurowski [26] proved that for all immune graphs $G = (V, E)$, $|E| \geq \lceil 3(|V|-1)/2 \rceil$, and constructed a large class of immune graphs attaining this lower bound for every value of $|V|$, called ABC graphs. In Chapter 4, we prove their conjecture:

Theorem *Every immune graph $G = (V, E)$ with $|E| = \lceil 3(|V| - 1)/2 \rceil$ is an ABC graph.*

In Chapter 5 we present two lower bounds for the maximum number of leaves over all spanning trees of a graph.

Theorem *Every connected graph G on n vertices without triangles with $\delta(G) \geq 3$, has a spanning tree with at least $\lceil (n + 4)/3 \rceil$ leaves.*

Theorem *Every connected graph G on n vertices with $\delta(G) \geq 3$, that contains D diamonds induced by vertices of degree three, has a spanning tree with at least $\lceil (2n - D + 12)/7 \rceil$ leaves.*

The proofs use the fact that spanning trees with many leaves correspond to small connected dominating sets. Both of these lower bounds are best possible for their respective graph classes. For both bounds simple polynomial time algorithms are given that find spanning trees satisfying the bounds.

In Chapter 6, the second bound above is used to find a new algorithm for the decision version of the Max-Leaf Spanning Tree problem. This problem asks whether a connected graph G on n vertices has a spanning tree with at least k leaves.

Theorem *The Algorithm in Chapter 6 is an algorithm for Max-Leaf Spanning Tree, and has time complexity $g(n)+f(k)$, where g is a polynomial of low degree, and $f(k) \in O(8.12^k)$.*

Note that for instances with fixed k, the time complexity of this algorithm is bounded by a polynomial in the input size, with degree independent of k. When we view k as the parameter of the problem, such an algorithm is called a *Fixed Parameter Tractable* (FPT) algorithm. A more formal definition can be found in Chapter 6. This is the current best FPT algorithm for the problem.

The five chapters can be read independently. A part of Chapter 2 has previously been published in [11]. The results in Chapter 3 have been published in [10]. Chapter 6 is based on joint research with Gerhard Woeginger and Tobias Brueggemann [12].

Chapter 2

Characterizations of sparsest cuts in various graph classes

2.1 Introduction

The *density* of an edge set $[S, T]$ is defined as

$$d(S,T) = \frac{|[S,T]|}{|S||T|}.$$

This can be interpreted as the ratio between the number of edges between S and T, and the maximum number of edges between S and T when edges may be added to the graph. An edge cut $[S, \overline{S}]$ of G with minimum density is called a *sparsest cut* of G. $d(G)$ denotes the density of a sparsest cut of G.

Finding a sparsest cut or the density of a sparsest cut is $\mathcal{NP}$-hard [43]. Accordingly, it is unlikely that there are straightforward methods to prove that a cut is a sparsest cut.

In this chapter, we characterize sparsest cuts for four graph classes: Cartesian product graphs, unit interval graphs, cactus graphs (for definitions see below) and complete bipartite graphs. These are very restricted and well-structured graph classes, but surprisingly, three of these proofs are still non-trivial, even for the complete bipartite graphs.

For each of these graph classes, we use entirely different techniques to prove that the cuts we construct are sparsest cuts. Together this gives a good overview of techniques that are available for proving this type of results.

In Section 2.2 we show that every Cartesian product graph $G \times H$ has a sparsest cut that can be derived directly from a sparsest cut of G or of H. We prove that the constructed cut is a sparsest cut using a flow problem (maximum uniform concurrent flow) that is an 'approximate dual' to the sparsest

cut problem. By this we mean that firstly, the value of a maximum uniform concurrent flow in G is a lower bound for the density of a sparsest cut in G. Secondly, the ratio between the maximum flow value and the minimum cut density is bounded [3, 4, 40]. Thirdly, for certain graphs, the maximum flow value is equal to the minimum cut density, and since it is also an lower bound, a flow construction can be used to prove that a cut is a sparsest cut. Graphs for which these values are equal are called bottleneck graphs [43]. Examples of bottleneck graphs are: cycles C_n, trees (paths P_n in particular), complete graphs K_n and n-cubes Q_n [44]. Examples of non-bottleneck graphs are expanders [40]. Since not every product graph $G \times H$ is a bottleneck graph (we show that this depends on whether G and H are bottleneck graphs), we cannot directly prove that the cut we construct is a sparsest cut using a flow construction. Therefore, for an arbitrary product graph we construct an auxiliary graph that has similar cut densities, and that is a bottleneck graph. Using a flow construction we determine the sparsest cut for this graph, which can be used to prove that the cut we construct in the product graph is a sparsest cut.

Since our cut construction for $G \times H$ makes use of a sparsest cut of G or H, this does not lead to a polynomial time algorithm to construct sparsest cuts for all product graphs. It does however lead to a polynomial time sparsest cut algorithm when G and H are part of a class for which polynomial time sparsest cut algorithms exist.

In Section 2.3 we study unit interval graphs. A graph G is a *unit interval graph* if a function $I : V(G) \to \mathbb{R}$ exists such that $uv \in E(G)$ if and only if $I(u) - 1 \leq I(v) \leq I(u) + 1$. I is called a *unit interval representation* of G. We show that every unit interval graph G with unit interval representation I has a sparsest cut $[S, \overline{S}]$ such that for all $u \in S$, $v \in \overline{S}$, $I(u) \leq I(v)$. Since a unit interval representation can be found in polynomial time (see e.g. [18], or [37] for a more general algorithm that recognizes all interval graphs), this yields a polynomial time algorithm to find sparsest cuts for this graph class. To prove this result, we develop some basic but useful techniques and lemmas for comparing densities in a graph. Using these techniques, we show that for every cut not of the aforementioned type, a different cut can be found with lower or equal density. These techniques are also used to show that a sparsest cut in a cactus always contains one or two edges. *Cactus graphs* are connected graphs in which every edge is part of at most one cycle (cactus graphs generalize trees and cycles).

Finally, in Section 2.4 we study complete bipartite graphs $K_{m,n}$ with $m \leq n$ and $n \geq 2$. For every cut we define two integer variables, such that we can express the density of the cut using only these two variables. We then give the integer values for which this expression is minimum. So basically, we prove that the cut we choose is a sparsest cut by comparing its density with the densities of all other cuts. This way we show that $d(K_{m,n}) = \min\{\frac{1}{2}, \frac{m}{m+n-1}\}$. Using this result, we can prove that $K_{m,n}$ is not a bottleneck graph when $m \geq 2$, $n \geq 3$.

For complete graphs and disconnected graphs finding a sparsest cut is trivial. To summarize, our results show that we can find a sparsest cut in polynomial time for complete graphs, disconnected graphs, complete bipartite graphs, cac-

tus graphs, unit interval graphs, and products of (products of) these, etc.

2.2 Sparsest cuts in Cartesian product graphs

2.2.1 Preliminaries

For the proofs in this section, it is more convenient to consider the reciprocal of the edge density of a cut. This is denoted by $d^{-1}(S,\overline{S}) = \frac{|S||\overline{S}|}{|[S,\overline{S}]|}$. Similarly, $d^{-1}(G)$ denotes the maximum value of $d^{-1}(S,\overline{S})$ over all $[S,\overline{S}]$. Also the flow problem mentioned in the introduction will be stated as a congestion minimization problem instead of a flow maximization problem. In addition, our definitions have to be generalized for edge-weighted graphs. The definitions in this section will use a capacity function $c : E(G) \to \mathbb{R}^+ - 0$ on the edges of G. For unweighted graphs, these definitions should be read as having $c(e) = 1$ for every e.

The sum of the capacities of edges in the cut S is denoted by $c[S,\overline{S}]$.

$$d^{-1}(S,\overline{S}) = \frac{|S||\overline{S}|}{c[S,\overline{S}]}.$$

Note that if $c(e) = 1$ for every edge e, then this definition coincides with our previous definition of $d^{-1}(S,\overline{S})$, so this definition is indeed a generalization. The definitions of $d^{-1}(G)$ and sparsest cut are generalized in the same way to edge weighted graphs.

A *flow* (P,f) on G is a set of paths P of G and a function $f : P \to \mathbb{R}^+$. If P is a set of paths, P_{uv} is the subset of P consisting of all (u,v)-paths for $u,v \in V(G)$. (Because we consider undirected flows, $P_{uv} = P_{vu}$.) P_e is the subset of P consisting of all paths that contain edge $e \in E(G)$. A flow (P,f) is called a *uniform flow* if $\sum_{p\in P_{uv}} f(p) = 1$ for all vertices $u \neq v$. The *edge-load* (also called edge-congestion) $\lambda(e)$ of an edge $e \in E(G)$ is defined as

$$\lambda(e) = \frac{\sum_{p\in P_e} f(p)}{c(e)}.$$

The *network load* $\lambda(P,f)$ of a flow is equal to the maximum edge-load: $\lambda(P,f) = \max_{e\in E(G)} \lambda(e)$. The goal of the uniform concurrent flow problem is to find a uniform flow that minimizes $\lambda(P,f)$. $\lambda(G)$ denotes the minimum network load over all uniform flows in G. A uniform flow (P,f) with $\lambda(P,f) = \lambda(G)$ is called an *optimal flow*.

Observe that for any cut $[S,\overline{S}]$, $d^{-1}(S,\overline{S})$ is a lower bound for $\lambda(G)$, so $d^{-1}(G) \leq \lambda(G)$. *Bottleneck graphs* are the graphs G for which $d^{-1}(G) = \lambda(G)$.

Not every graph is a bottleneck graph, so we can not always use the value $d^{-1}(G)$ to show that a certain flow is optimal, and vice versa. But the uniform concurrent flow problem can be formulated as a linear program (LP), so we can use its dual problem for this purpose. (For more information on linear

programming and duality we recommend [16].) The following LP describes the problem of finding an optimal flow in $G = (V, E)$:

$$\begin{array}{lrcll} \min & \lambda & & & \\ \text{s.t.} & & \sum_{p \in P_{uv}} f(p) & \geq 1 & \forall u, v \in V \\ & c(e)\lambda & - \sum_{p \in P_e} f(p) & \geq 0 & \forall e \in E \\ & \lambda & & \geq 0 & \\ & & f(p) & \geq 0 & \forall p \in P \end{array}$$

For the description of the dual problem we need the following notations: Let $t : E \to \mathbb{R}^+$ be a *distance function* on the edges of G ($0 \in \mathbb{R}^+$). A distance function is called a *normalized distance function* if $\sum_{e \in E} c(e)t(e) = 1$. We will only consider normalized distance functions. $d_t(u, v)$ denotes the length of a shortest path between u and v measured over this distance function. The *value of a distance function* t is defined as $\sum_{u,v \in V} d_t(u, v)$. The *distance bound* $D(G)$ of G is the maximum of these values:

$$D(G) = \max_{t: E \to \mathbb{R}^+} \sum_{u,v \in V} d_t(u, v).$$

A corresponding distance function is called an *optimal distance function*. The following LP describes the problem of finding an optimal distance function:

$$\begin{array}{lrcll} \max & \sum_{u,v} d_t(u, v) & & & \\ \text{s.t.} & & \sum_e c(e)t(e) & \leq 1 & \\ & d_t(u, v) & - \sum_{e: p \in P_e} t(e) & \leq 0 & \forall u, v \in V(G), \forall p \in P_{uv} \\ & d_t(u, v) & & \geq 0 & \forall u, v \in V(G) \\ & & t(e) & \geq 0 & \forall e \in E(G) \end{array}$$

It can be checked that the second LP is the dual of the first LP. Note that to ensure that solving the first LP gives the value of $\lambda(G)$, and that solving the second LP gives the value of $D(G)$, we have to choose P to be the set of *all possible paths* on the graph. This set is clearly not polynomially bounded by the size of the input, and therefore the number of variables resp. inequalities of the two LPs are not polynomially bounded. We remark that there are other, more complicated LP-formulations of the problems without this problem [51]. However, for our purposes, this large number of variables resp. inequalities does not matter.

The set of all possible paths P is finite. Also, since we assume G to be connected, $P_{uv} \neq \emptyset$ for every u and v, and in that case a feasible solution for the first LP is easily found. Clearly, the value of the first LP is bounded. Therefore, since these two LPs are duals, it follows from the theorem of strong linear programming duality [16] that $\lambda(G) = D(G)$.

Note also that for every cut $[S, \overline{S}]$, there is a corresponding distance function with value $d^{-1}(S, \overline{S})$: assign a weight of $t(e) = 1/c[S, \overline{S}]$ to every edge e, and then $\sum_{u,v \in V} d_t(u, v) = (|S||\overline{S}|)/c[S, \overline{S}]$, since there are $|S||\overline{S}|$ vertex pairs that have distance $1/c[S, \overline{S}]$, and the other vertex pairs have distance 0.

To summarize, we have shown that the following relations hold between these three values:

Lemma 2.1 $d^{-1}(G) \leq D(G) = \lambda(G)$.

An alternative proof of this result can be found in [51].

The *product graph* $G \times H$ of two graphs G and H is defined as follows: $G \times H$ has vertex set $V(G) \times V(H)$, and an edge set containing all edges of the form

$$((u, x), (v, x))$$

if $(u, v) \in E(G)$ and $x \in V(H)$, and

$$((u, x), (u, y))$$

if $(x, y) \in E(H)$ and $u \in V(G)$. Edges of the first type are called *horizontal edges* and edges of the second type are called *vertical edges*. If G and H have edge weights c, then the capacity of edges in $G \times H$ is equal to $c(u, v)$ for edges of the first type and $c(x, y)$ for edges of the second type. The subgraph of $G \times H$ induced by the vertices (u, x) for a certain fixed $x \in V(H)$ and all $u \in V(G)$ is called the *G-layer* corresponding to x. *H*-layers corresponding to vertices in G are defined analogously.

2.2.2 Results

For convenience, the theorems are only formulated for products of two graphs G and H. We will also assume G and H have uniform capacities, and therefore write $|[S, \overline{S}]|$ instead of $c[S, \overline{S}]$. It can be verified that the results which will be established can be generalized to graphs with non-uniform capacities. Throughout this section, $n = |V(G)|$ and $m = |V(H)|$.

In the first theorem we construct a uniform flow and a normalized distance function in $G \times H$ using optimal flows and distance functions in G and H. Lemma 2.1 will show that the network load of the constructed flow and value of the distance function are equal and thus optimal.

Uniform concurrent flow in product graphs and generalizations of product graphs was previously studied in [52]. In [52], also an optimal flow in a product graph is constructed using flows in the factors, but the construction is different from our construction below.

Theorem 2.2 *For any two graphs G and H,*

$$D(G \times H) = \max\{mD(G), nD(H)\} =$$

$$\max\{m\lambda(G), n\lambda(H)\} = \lambda(G \times H).$$

Proof: To prove these equalities we first show that

$$D(G \times H) \geq \max\{mD(G), nD(H)\}$$

and

$$\lambda(G \times H) \leq \max\{m\lambda(G), n\lambda(H)\}.$$

To prove the first inequality, consider the following argument: if t is an optimal normalized distance function on the edges of G we can define a distance function t' on the edges of $G \times H$, that is not normalized, as follows.

$$t'((u,x),(v,x)) = t(u,v)$$

for every horizontal edge of $G \times H$, and

$$t'((u,x),(u,y)) = 0$$

for every vertical edge of $G \times H$. Because a distance of 0 is assigned to vertical edges, the distance in $G \times H$ between (u,x) and (v,y) is the same as the distance in G from u to v, regardless of the choice of x and y (if $u = v$ then the length is 0). Therefore a vertex pair u and v in G corresponds to m^2 vertex pairs in $G \times H$ with the same distance. This way, all vertex pairs in $G \times H$ have been considered and we have shown that

$$\sum_{p,q \in V(G \times H)} d_{t'}(p,q) = m^2 \sum_{u,v \in V(G)} d_t(u,v) = m^2 D(G).$$

The total distance assigned to edges in $G \times H$ is m times the total distance assigned to edges in G, so to normalize the distance function we can divide all edge distances by m. Then the value of the constructed distance function becomes $mD(G)$. A similar construction can be done using an optimal distance function on H, and $D(G \times H) \geq \max\{mD(G), nD(H)\}$ follows.

To prove the second inequality, a uniform flow in $G \times H$ is constructed from optimal flows in G and H.

The path set P_{uv} in an optimal flow (P, f) of G can be used to construct a corresponding path set in a G-layer of $G \times H$ from (u,x) to (v,x). For a flow from (u,x) to (u,y), we use the path set P'_{xy} from an optimal flow (P', f') in H.

To construct a path set from (u,x) to (v,y) with a total flow of 1, first we use the path set from (u,x) to (v,x) with a total flow of $\frac{1}{2}$, then we use the path set from (v,x) to (v,y) with a total flow of $\frac{1}{2}$. These path sets can be combined (in an arbitrary manner) to form a path set from (u,x) to (v,y) with a total flow of $\frac{1}{2}$. Then the same is done using (u,y) as the connection point, and together these path sets give the desired path set with a total flow of 1. Note that if $u = v$ or $x = y$, then the flow is not actually split into two path sets.

In every G-layer corresponding to a fixed $x \in V(H)$ of $G \times H$, the path set from (u,x) to (v,x) is used:

- $m - 1$ times for a flow of $\frac{1}{2}$, once for every vertex pair (u,y) and (v,x) with $y \neq x$.

- $m-1$ times for a flow of $\frac{1}{2}$, once for every vertex pair (u,x) and (v,y) with $y \neq x$.
- Once for a flow of 1 from (u,x) to (v,x).

So in every G-layer every path set is used for a total flow of $\frac{1}{2}(m-1)+\frac{1}{2}(m-1)+1=m$. Using this flow construction the maximum load of the horizontal edges is equal to $m\lambda(G)$. Similarly, the maximum load of the vertical edges is equal to $n\lambda(H)$, so $\lambda(G\times H) \leq \max\{m\lambda(G), n\lambda(H)\}$.

Now we have, using Lemma 2.1 for G, H resp. $G\times H$:

$$D(G\times H) \geq \max\{mD(G), nD(H)\} =$$

$$\max\{m\lambda(G), n\lambda(H)\} \geq \lambda(G\times H) = D(G\times H).$$

Therefore all inequalities must be equalities and the optimality of the constructed distance function and flow follows. □

The following lemma is a similar statement for the sparsest cut density $d(G\times H)$, and will be proved by a construction using sparsest cuts of G and H.

Lemma 2.3 *For any two graphs G and H,*

$$d^{-1}(G\times H) \geq \max\{md^{-1}(G), nd^{-1}(H)\}.$$

Proof: We will show that each cut $[S,\overline{S}]$ in G corresponds to a cut $[S',\overline{S'}]$ in $G\times H$ with value $d^{-1}(S',\overline{S'}) = md^{-1}(S,\overline{S})$. S' is defined as follows: $(u,x)\in S'$ if and only if $u\in S$. Therefore $|S'|=m|S|$ and $|\overline{S'}|=m|\overline{S}|$. If $(u,v)\in[S,\overline{S}]$, then $((u,x),(v,x))\in[S',\overline{S'}]$ for every $x\in V(H)$, so $|[S',\overline{S'}]| = m|[S,\overline{S}]|$. It follows that $d^{-1}(S',\overline{S'}) = \frac{m^2}{m}d^{-1}(S,\overline{S}) = md^{-1}(S,\overline{S})$. Thus $d^{-1}(G\times H) \geq md^{-1}(G)$. Analogously, $d^{-1}(G\times H)\geq nd^{-1}(H)$. □

Combining the above theorem and lemmas leads to the following result.

Corollary 2.4 *If G and H are bottleneck graphs, then $G\times H$ is a bottleneck graph.*

Proof: Using Lemma 2.3, the fact that G and H are bottleneck graphs, Theorem 2.2 and Lemma 2.1 respectively, we have

$$d^{-1}(G\times H) \geq \max\{md^{-1}(G), nd^{-1}(H)\} = \max\{m\lambda(G), n\lambda(H)\} =$$

$$\lambda(G\times H) \geq d^{-1}(G\times H).$$

Therefore $\lambda(G\times H) = d^{-1}(G\times H)$, and $G\times H$ is a bottleneck graph. □

We can also conclude that if G and H are bottleneck graphs, then the cut constructed in Lemma 2.3 is a sparsest cut. The next theorem states that this is true in general, which allows a corollary similar to Corollary 2.4 to be formulated about non-bottleneck graphs.

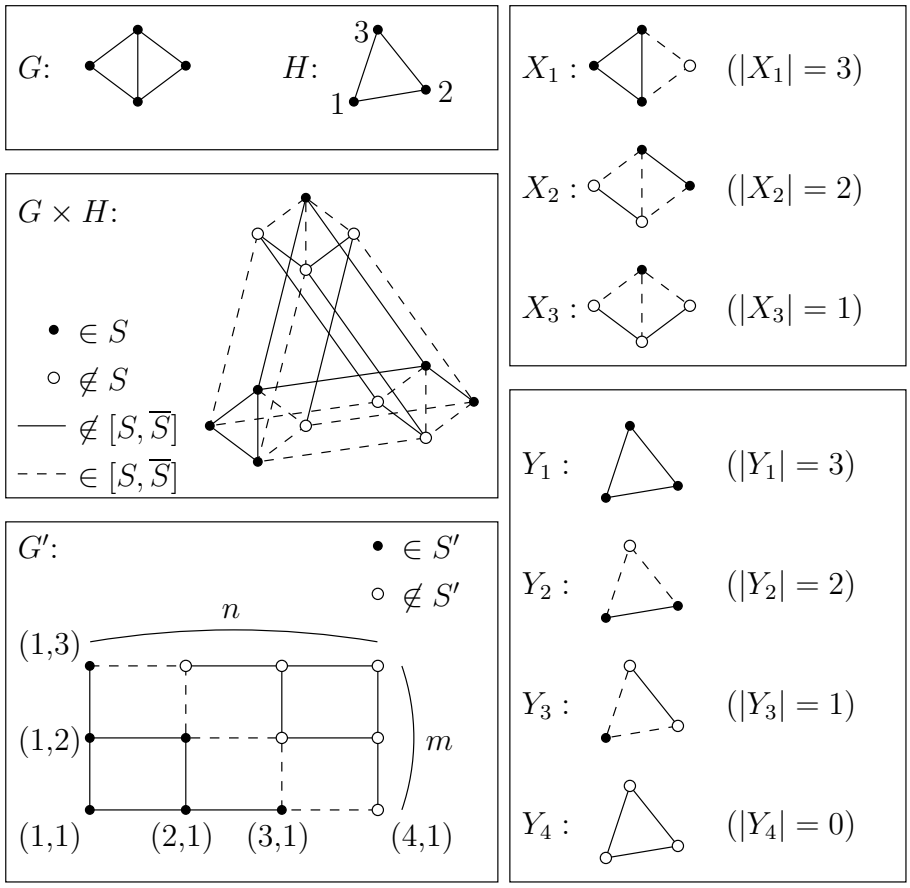


Figure 2.1: An example of the construction of G' and S' from $G \times H$ and S.

Theorem 2.5 $d^{-1}(G \times H) = \max\{md^{-1}(G), nd^{-1}(H)\}$.

Proof: It suffices to show that $d^{-1}(S, \overline{S}) \leq \max\{md^{-1}(G), nd^{-1}(H)\}$ holds for every $S \subset V(G \times H)$, which together with Lemma 2.3 proves our claim. This is done by constructing a new graph G' with non-uniform edge capacities. First it is shown that $d^{-1}(G') = \max\{md^{-1}(G), nd^{-1}(H)\}$. To conclude the proof it is shown that every cut $[S, \overline{S}]$ in $G \times H$ corresponds to a cut $[S', \overline{S'}]$ in G' with $d^{-1}(S, \overline{S}) \leq d^{-1}(S', \overline{S'})$. For an example of the construction in this proof see Figure 2.1.

For the construction of G', take two paths P_n and P_m. Label the vertices of P_n (P_m) along the path with labels $1, \ldots, n$ ($1, \ldots, m$). Set edge capacities in P_n to $c(i, i+1) = i(n-i)/d^{-1}(G)$ for $i = 1, \ldots, n-1$. Set edge capacities in P_m to $c(i, i+1) = i(m-i)/d^{-1}(H)$ for $i = 1, \ldots, m-1$. Now it

can be verified that $d^{-1}(P_n) = d^{-1}(G)$ and $d^{-1}(P_m) = d^{-1}(H)$ (and that every edge in P_n (P_m) gives a sparsest cut). Define $G' = P_n \times P_m$. Note that $|V(G')| = |V(G \times H)| = nm$. Using the fact that paths are bottleneck graphs, Corollary 2.4 and Theorem 2.2 imply that $d^{-1}(G') = \max\{md^{-1}(G), nd^{-1}(H)\}$.

Now we consider a cut $[S, \overline{S}]$ in $G \times H$. We will construct a cut $[S', \overline{S'}]$ in G' with $|S'| = |S|$ (and thus $|\overline{S'}| = |\overline{S}|$) and $c[S', \overline{S'}] \leq |[S, \overline{S}]|$.

Using S, we can define a cut $X(v)$ in G for every vertex $v \in V(H)$:

$$X(v) = \{u \in V(G) : (u, v) \in S\}.$$

Number the vertices $V(H) = \{v_1, \ldots, v_m\}$ such that $i < j \Rightarrow |X(v_i)| \geq |X(v_j)|$. Now we write X_i instead of $X(v_i)$.

These cuts are now defined such that $\sum_{i=1}^{m} |[X_i, \overline{X}_i]|$ is equal to the number of horizontal edges in the cut $[S, \overline{S}]$.

Next we define a set of n cuts in H using S:

$$Y_k = \{v_i : k \leq |X_i|\}.$$

Because $|X_i|$ is decreasing, $Y_k = \{v_1, \ldots, v_p\}$ for some p.

We will prove that $\sum_{k=1}^{n} |[Y_k, \overline{Y}_k]|$ does not exceed the number of vertical edges in $[S, \overline{S}]$ by constructing a mapping of the edges in these cuts to the vertical edges in $[S, \overline{S}]$ that is an injection.

Consider an edge $(v_i, v_j) \in E(H)$, and suppose $i < j$ and therefore $|X_i| \geq |X_j|$. Between the G-layer in $G \times H$ corresponding to v_i and the one corresponding to v_j, there are at least $|X_i| - |X_j|$ vertical edges in $[S, \overline{S}]$. Label an arbitrary subset of $|X_i| - |X_j|$ of these edges with the numbers $|X_j| + 1, \ldots, |X_i|$. This labeling of vertical edges is done for every edge in H. The mapping is as follows: if $(v_i, v_j) \in [Y_k, \overline{Y}_k]$, then w.l.o.g. $|X_i| \geq k$ and $|X_j| < k$, so this edge can be mapped to the edge $((u, v_i), (u, v_j))$ that was labeled with label k. Now for every edge in $\cup_{k=1,\ldots,n}[Y_k, \overline{Y}_k]$ a unique edge in $[S, \overline{S}]$ is assigned, which proves our claim.

Next we will construct a cut $[S', \overline{S'}]$ in G':

$$S' = \{(k, i) : k \leq |X_i|\},$$

so the horizontal edges in $[S', \overline{S'}]$ (corresponding to edges of P_n) are of the form $((|X_i|, i), (|X_i| + 1, i))$. Using the definition of Y_k, we can rewrite S' as:

$$S' = \{(k, i) : v_i \in Y_k\},$$

and using the fact that $Y_k = \{v_1, \ldots, v_p\}$ for some p, we know that the vertical edges in $[S', \overline{S'}]$ (corresponding to edges of P_m) are of the form $((k, |Y_k|), (k, |Y_k| + 1))$. Now we have

$$\begin{aligned}[S', \overline{S'}] = \quad & \{((|X_i|, i), (|X_i| + 1, i)) : |X_i| \neq 0 \wedge |X_i| \neq n\} \quad \cup \\ & \{((k, |Y_k|), (k, |Y_k| + 1)) : |Y_k| \neq 0 \wedge |Y_k| \neq m\}.\end{aligned}$$

The capacity of the cut $[S', \overline{S'}]$ is equal to:

$$c[S', \overline{S'}] = \sum_{i=1}^{m} |X_i|(n - |X_i|)/d^{-1}(G) + \sum_{k=1}^{n} |Y_k|(m - |Y_k|)/d^{-1}(H) \leq$$

$$\sum_{i=1}^{m} |[X_i, \overline{X}_i]| + \sum_{k=1}^{n} |[Y_k, \overline{Y}_k]| \leq |[S, \overline{S}]|.$$

The first equality follows from the definition of the capacities in G', the first inequality follows from the fact that $d^{-1}(G) \geq \frac{|X_i|(n-|X_i|)}{|[X_i, \overline{X}_i]|}$ for any i, and a similar statement for $d^{-1}(H)$, and the last inequality follows from the proofs above.

Now we have proved that for any cut $[S, \overline{S}]$ in $G \times H$, there is a cut $[S', \overline{S'}]$ in G' with

$$d^{-1}(S, \overline{S}) = \frac{|S||\overline{S}|}{|[S, \overline{S}]|} \leq \frac{|S'||\overline{S'}|}{c[S', \overline{S'}]} \leq \max\{md^{-1}(G), nd^{-1}(H)\},$$

so $d^{-1}(G) \leq \max\{md^{-1}(G), nd^{-1}(H)\}$. □

This proves that the cut constructed in Lemma 2.3 is a sparsest cut of $G \times H$. So every graph $G \times H$ has a sparsest cut that consists only of horizontal or only of vertical edges. We also have the following corollary.

Corollary 2.6 *If G and H are not bottleneck graphs, then $G \times H$ is not a bottleneck graph.*

In view of Corollaries 2.4 and 2.6, there is one question left: what if one graph (say G) is a bottleneck graph and the other graph (H) is not? Interestingly, the answer only depends on the values of $D(G)$ and $D(H)$, not on $d^{-1}(G)$ and $d^{-1}(H)$.

Corollary 2.7 *If G is a bottleneck graph and H is not a bottleneck graph, then $G \times H$ is a bottleneck graph if and only if $nD(H) \leq mD(G)$.*

Proof: If $nD(H) > mD(G)$, then

$$D(G \times H) = \max\{mD(G), nD(H)\} =$$

$$nD(H) > \max\{md^{-1}(G), nd^{-1}(H)\} = d^{-1}(G \times H),$$

so $G \times H$ is not a bottleneck graph.

If $nD(H) \leq mD(G)$, then we know that $nd^{-1}(H) < nD(H) \leq mD(G) = md^{-1}(G)$, so

$$d^{-1}(G \times H) = \max\{md^{-1}(G), nd^{-1}(H)\} = md^{-1}(G) =$$

$$mD(G) = \max\{mD(G), nD(H)\} = D(G \times H),$$

and $G \times H$ is a bottleneck graph. □

2.3 Sparsest cuts in unit interval graphs

2.3.1 Results

In this section we characterize sparsest cuts of unit interval graphs. In order to give a short expression for the form of a sparsest cut, we use the following definition.

Definition 2.8 *Let $I : V(G) \to \mathbb{R}$ be a unit interval representation for the graph G. If A and B are disjoint non-empty subsets of $V(G)$, we write $A \prec_I B$ if for all $u \in A$ and $v \in B$, $I(u) \leq I(v)$ holds.*

Let G be a unit interval graph with unit interval representation I. We show that G has a sparsest cut $[S, \overline{S}]$ such that $S \prec_I \overline{S}$. This is done by considering an arbitrary cut $[S, T]$, and partitioning S and T into $S_1, \ldots, S_k$ resp. $T_1, \ldots, T_l$ with $k - 1 \leq l \leq k$ such that $S_1 \prec_I T_1 \prec_I S_2 \prec_I \ldots \prec_I T_{k-1} \prec_I S_k$, and in addition $S_k \prec_I T_k$ if $l = k$. For a given cut $[S, T]$ and unit interval representation I of G, a partition of S and T into non-empty subsets with this property is called an *I-partition* of S and T. Observe that w.l.o.g. we can always find an I-partition. See Figure 2.2 for an example.

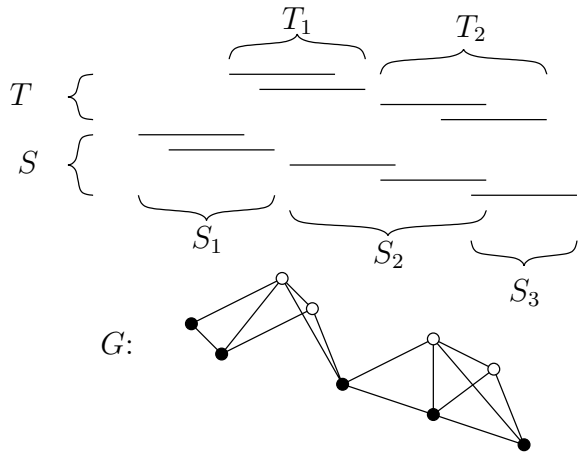


Figure 2.2: An I-partition for S and T

We show that if $k > 1$, then we can reassign these subsets into disjoint non-empty subsets S' and T' with $S' \cup T' = V(G)$, such that $d(S', T') \leq d(S, T)$. For this purpose, we need to express the density between S and T as a weighted average of densities between subsets of S and T.

Observation 2.9 *If A, B and C are disjoint non-empty subsets of $V(G)$, then* $d(A, B \cup C) = \frac{d(A,B)|B| + d(A,C)|C|}{|B| + |C|}$.

Proof:

$$d(A, B \cup C) = \frac{|[A, B \cup C]|}{|A|(|B| + |C|)} = \frac{|[A, B]| + |[A, C]|}{|A|(|B| + |C|)} =$$

$$\frac{\frac{|[A,B]|}{|A||B|}|B| + \frac{|[A,C]|}{|A||C|}|C|}{|B| + |C|} = \frac{d(A, B)|B| + d(A, C)|C|}{|B| + |C|}$$

□

The following definition is introduced to improve the readability of our proofs. Since we mainly compare densities with the density of a sparsest cut, it makes sense to normalize densities in the following way.

Definition 2.10 *Let G be a graph with sparsest cut $[S, \overline{S}]$ and $d = d(S, \overline{S})$. Let $A, B \subseteq V(G)$ with $A \cap B = \emptyset$. The* normalized density *between A and B is $e(A, B) = d(A, B) - d$.*

Note that if $A \cup B \neq V(G)$, then $e(S, T)$ may be negative. Normalized densities can also be expressed as weighted averages:

Observation 2.11 *If A, B and C are disjoint non-empty subsets of $V(G)$, then $e(A, B \cup C) = \frac{e(A,B)|B|+e(A,C)|C|}{|B|+|C|}$.*

Proof:

$$e(A, B \cup C) = d(A, B \cup C) - d = \frac{d(A, B)|B| + d(A, C)|C|}{|B| + |C|} - d =$$

$$\frac{e(A, B)|B| + d|B| + e(A, C)|C| + d|C|}{|B| + |C|} - d = \frac{e(A, B)|B| + e(A, C)|C|}{|B| + |C|}.$$

□

The following simple lemma is the key to our approach.

Lemma 2.12 *If $[A \cup B, C]$ is a sparsest cut of G, with A and B disjoint and non-empty, then $e(A, B) \geq 0$. If $e(A, B) = 0$, then $[A, B \cup C]$ or $[B, A \cup C]$ is also a sparsest cut of G.*

Proof: $[A \cup B, C]$ is a sparsest cut so $e(A \cup B, C) = 0$. Since this is a weighted average of $e(A, C)$ and $e(B, C)$ (Observation 2.11), we may assume $e(A, C) \leq 0$ and $e(B, C) \geq 0$. If $e(A, B) < 0$, then

$$e(A, B \cup C) = \frac{e(A, B)|B| + e(A, C)|C|}{|B| + |C|} < 0,$$

a contradiction with the fact that $[A \cup B, C]$ is a sparsest cut. This shows that $e(A, B) \geq 0$. If $e(A, B) = 0$, then similarly we get $e(A, B \cup C) \leq 0$. So $0 = e(A \cup B, C) \leq e(A, B \cup C) \leq 0$, and therefore $[A, B \cup C]$ is also a sparsest cut.

If instead we assume $e(B, C) \leq 0$ and $e(A, B) = 0$, then we find in the same way that $[B, A \cup C]$ is also a sparsest cut. □

Corollary 2.13 *If $[S,T]$ is a sparsest cut in a connected graph G, then $G[S]$ and $G[T]$ are connected.*

Alternatively, the above corollary states that every sparsest cut is a *minimal* edge cut. Before we continue with unit interval graphs, we demonstrate the usefulness of Corollary 2.13 by characterizing sparsest cuts for cactus graphs. Recall that a graph is a cactus if it is connected and every edge is part of at most one cycle.

Proposition 2.14 *If M is a sparsest cut of a cactus G, then $|M| \leq 2$.*

Proof: Let $M = [S, \overline{S}]$. Since G is connected and M is a sparsest cut, $G[S]$ and $G[\overline{S}]$ are connected (Corollary 2.13). Suppose $|M| \geq 3$. Let e, f and g be distinct edges in M. Because $G[S]$ and $G[\overline{S}]$ are connected, a cycle C_1 exists that contains e and f, and no other edges of M. Similarly, a cycle C_2 exists that contains e and g, and no other edges of M. So C_1 and C_2 are different cycles, but both contain e, a contradiction with the fact that G is a cactus.□

It follows that for cactus graphs, a sparsest cut can be found in polynomial time by considering all possible sets of one or two edges.

The property that is stated in the next lemma is the only property of unit interval graphs we will use in the proof of our main result (Theorem 2.16).

Lemma 2.15 *Let $I : V(G) \to \mathbb{R}$ be a unit interval representation for the graph G. If $A, B, C \subseteq V(G)$ with $A \prec_I B \prec_I C$, then $d(B, A \cup C) \geq d(A, C)$.*

Proof: Let $A_1 \subseteq A$ be the vertices in A that have at least one neighbor in C. Similarly, $C_1 \subseteq C$ are the vertices in C that have at least one neighbor in A. Let $\alpha = |A_1|/|A|$, and $\gamma = |C_1|/|C|$.

$$d(A,C) = \frac{|[A,C]|}{|A||C|} \leq \frac{|A_1||C_1|}{|A||C|} = \alpha\gamma \leq \min\{\alpha, \gamma\}.$$

Now we will show that $\min\{\alpha, \gamma\}$ is a lower bound for $d(B, A \cup C)$. Suppose $a \in A$ and $c \in C$ are adjacent, so $I(a) + 1 \geq I(c)$. Consider $b \in B$. $I(a) \leq I(b)$ (since $A \prec_I B$), and $I(b) \leq I(c)$ (since $B \prec_I C$). It follows that $I(a)+1 \geq I(b)$, so a and b are adjacent, and $I(b)+1 \geq I(c)$, so b and c are adjacent. This proves that if a vertex $a \in A$ has a neighbor in C, then every vertex in B is adjacent to a, and a similar statement holds for vertices in C. Now we write

$$d(B, A \cup C) = \frac{|[B, A \cup C]|}{|B|(|A| + |C|)} \geq \frac{|B|(|A_1| + |C_1|)}{|B|(|A| + |C|)} = \frac{\alpha|A| + \gamma|C|}{|A| + |C|},$$

which is a weighted average of α and γ, so

$$d(B, A \cup C) \geq \min\{\alpha, \gamma\}.$$

□

Theorem 2.16 *Let $I : V(G) \to \mathbb{R}$ be a unit interval representation for the graph G. G has a sparsest cut $[S,T]$ such that $S \prec_I T$.*

Proof: Consider a sparsest cut $[S,T]$ and I-partition $\{S_1,\ldots,S_k\}$ and $\{T_1,\ldots,T_l\}$ of S and T, that has $k+l$ minimum, among all such sparsest cuts and I-partitions. We use the following shorthand notation: $S_{i\ldots j} = S_i \cup \ldots \cup S_j$, and $T_{i\ldots j} = T_i \cup \ldots \cup T_j$.

If $k = 1$, then $S = S_1 \prec_I T_1 = T$, and we have found the desired sparsest cut. Otherwise, we distinguish two cases: $l = k-1$, and $l = k \geq 2$.

Case 1: $l = k-1$.
For all $1 \leq t \leq k-1$, $e(S_{1\ldots t}, S_{t+1\ldots k}) \geq 0$ (Lemma 2.12). If this is an equality, then $[S_{1\ldots t}, S_{t+1\ldots k} \cup T_{1\ldots k-1}]$ or $[S_{t+1\ldots k}, S_{1\ldots t} \cup T_{1\ldots k-1}]$ is also a sparsest cut (Lemma 2.12). The first cut has an I-partition with $2t < 2k-1 = k+l$ classes, and the second cut has an I-partition with $2(k-t) < 2k-1 = k+l$ classes, both contradictions with our choice of $[S,T]$. We conclude that $e(S_{1\ldots t}, S_{t+1\ldots k}) > 0$ for every $1 \leq t \leq k-1$.

Since $S_{1\ldots t} \prec T_t \prec S_{t+1\ldots k}$ and $S_{1\ldots t} \cup S_{t+1\ldots k} = S$, it follows from Lemma 2.15 that $e(T_t, S) \geq e(S_{1\ldots t}, S_{t+1\ldots k}) > 0$, for all $1 \leq t \leq k-1$. Since

$$e(S,T) = \frac{e(T_1,S)|T_1| + \ldots + e(T_{k-1},S)|T_{k-1}|}{|T_1| + \ldots + |T_{k-1}|},$$

we have $e(S,T) > 0$, a contradiction with the fact that $[S,T]$ is a sparsest cut. This concludes the case $l = k-1$.

Case 2: $l = k \geq 2$.
We again have $e(T_t, S) > 0$ for all $1 \leq t \leq k-1$ (see the previous case), but it is possible that $e(T_k, S) < 0$, so we cannot immediately obtain a contradiction this way.

First we show that for every $2 \leq t \leq k$,

$$e(S_t, T) > e(T_{1\ldots t-1}, S)\frac{|S|}{|T|}, \tag{2.1}$$

and for every $1 \leq t \leq k-1$,

$$e(T_t, S) > e(S_{t+1\ldots k}, T)\frac{|T|}{|S|}. \tag{2.2}$$

For a fixed t with $2 \leq t \leq k$, we denote $T_L = T_{1\ldots t-1}$ and $T_H = T_{t\ldots k}$. We showed that $e(T_i, S) > 0$ for all $i < k$, so we have $e(T_L, S) = \alpha > 0$ ($e(T_L, S)$ is a weighted average of $e(T_i, S)$ for $i = 1,\ldots,t-1$). Since

$$0 = e(T_L \cup T_H, S) = \frac{\alpha|T_L| + e(T_H,S)|T_H|}{|T_L| + |T_H|},$$

we have $e(T_H, S) = -\alpha\frac{|T_L|}{|T_H|}$. Now we consider the cut $[T_H, V(G)\backslash T_H]$. Since

$$0 \leq e(T_H, V(G)\backslash T_H) = \frac{e(T_H,S)|S| + e(T_H,T_L)|T_L|}{|S| + |T_L|},$$

we have $e(T_H, T_L) \geq -e(T_H, S)\frac{|S|}{|T_L|} = \alpha\frac{|S|}{|T_H|}$. Finally, using Lemma 2.15 and $T_L \prec_I S_t \prec_I T_H$ we obtain $e(S_t, T) \geq e(T_L, T_H) \geq \alpha\frac{|S|}{|T_H|} > e(T_L, S)\frac{|S|}{|T|}$. By symmetry (since $l = k$), (2.2) can be proved the same way.

Using (2.1) and (2.2), we now prove by induction on i that for all $1 \leq i \leq k-1$,

$$e(T_{1\ldots i}, S) > e(S_{i+1\ldots k}, T)\frac{|T|}{|S|}. \tag{2.3}$$

If $i = 1$, then (2.3) is equal to (2.2) for $t = 1$.

If $i > 1$ then our induction hypothesis is that $e(T_{1\ldots i-1}, S) > e(S_{i\ldots k}, T)\frac{|T|}{|S|}$. When we combine this with (2.1) we get

$$e(S_i, T) > e(T_{1\ldots i-1}, S)\frac{|S|}{|T|} > e(S_{i\ldots k}, T).$$

Since $e(S_{i\ldots k}, T)$ is a weighted average of $e(S_i, T)$ and $e(S_{i+1\ldots k}, T)$, it follows that

$$e(S_{i\ldots k}, T) > e(S_{i+1\ldots k}, T)$$

We combine this with the induction hypothesis:

$$e(T_{1\ldots i-1}, S) > e(S_{i\ldots k}, T)\frac{|T|}{|S|} > e(S_{i+1\ldots k}, T)\frac{|T|}{|S|}$$

From (2.2) we see that $e(S_{i+1\ldots k}, T)\frac{|T|}{|S|}$ is also a lower bound for $e(T_i, S)$. Since $e(T_{1\ldots i}, S)$ is a weighted average of $e(T_{1\ldots i-1}, S)$ and $e(T_i, S)$, it follows that

$$e(T_{1\ldots i}, S) > e(S_{i+1\ldots k}, T)\frac{|T|}{|S|},$$

which concludes the induction proof.

Using (2.3) resp. (2.1), we obtain a contradiction for the case $l = k$:

$$e(T_{1\ldots k-1}, S) > e(S_k, T)\frac{|T|}{|S|} > e(T_{1\ldots k-1}, S).$$

We showed that both cases with $k > 1$ lead to a contradiction, so with our choice of S and T, k must be 1, and thus $S \prec_I T$. □

Unit interval graphs can be recognized in polynomial time, and a unit interval representation can be found in polynomial time [18]. It follows that for unit interval graphs sparsest cuts can be found in polynomial time.

Corollary 2.17 *There exists a polynomial time algorithm that for any graph G, returns a sparsest cut of G if G is a unit interval graph.*

Proof: The algorithm is as follows. First decide whether G is a unit interval graph. If so, construct a unit interval representation I. Note that I can be

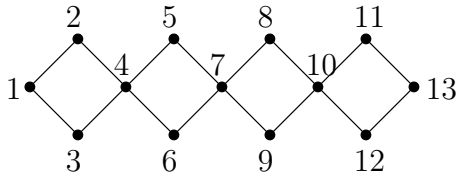


Figure 2.3: A non-interval graph and order satisfying the property from Lemma 2.15

constructed such that $I(u) \neq I(v)$ for every u and v. Start with $S = \{v_1\}$ such that $S \prec_I \overline{S}$, and calculate $d(S, \overline{S})$. Successively, add the unique vertex to S that maintains $S \prec_I \overline{S}$, and calculate $d(S, \overline{S})$. After this is exhausted, choose the cut with the lowest density.

Only $|V(G)| - 1$ cuts are considered, and since these are all of the cuts with $S \prec_I \overline{S}$, by Theorem 2.16, one of these cuts is a sparsest cut. □

2.3.2 Discussion

Since Lemma 2.15 is the only property of unit interval graphs that we used to prove Theorem 2.16, our result actually holds for a larger graph class. Namely the class of graphs for which a complete order $\prec$ on $V(G)$ exists, and corresponding partial order $\prec$ subsets of $V(G)$ (deduced from the vertex order the same way as $\prec_I$ is deduced from I), such that $A \prec_I B \prec_I C$ implies $d(B, A \cup C) \geq d(A, C)$. We can show that this graph class contains more than only unit interval graphs; consider for instance graphs consisting of a number of 4-cycles joined in a path structure, as illustrated in Figure 2.3. The illustrated graph G satisfies the aforementioned property: consider the vertex order given by the the vertex labels. For all choices $A \prec B \prec C$, either $|[A, C]| = 0$, or $|[A, C]| = 1$, $|[B, A \cup C]| = 2$ and $|B| = 1$. It can now be checked that $d(B, A \cup C) \geq d(A, C)$. In addition, G is clearly not a unit interval graph.

We do not know if the generalization of unit interval graphs defined above contains or is equal to a known generalization of unit interval graphs, or how many non-unit interval graphs satisfy the above property.

An obvious question is whether these results can be generalized to all interval graphs G. We will show that this is not true in a strong sense. G is an *interval graph* if we can assign intervals of the real line to the vertices such that two vertices are adjacent if and only if their intervals overlap. Formally, this means that there are functions $I_l : V(G) \to \mathbb{R}$ and $I_h : V(G) \to \mathbb{R}$ with $I_l(v) \leq I_h(v)$, such that $uv \in E(G)$ if and only if $I_l(u) \leq I_h(v)$ and $I_l(v) \leq I_h(u)$. This is called an *interval representation* of G. If G has an interval representation with $I_h(u) = I_l(u) + 1$ for all u, note that this corresponds to our previous definition of a unit interval representation.

So for a straightforward generalization of Theorem 2.16 to all interval graphs, we can for instance ask the following question. For all interval graphs, can we describe a non-trivial (partial) order $\prec$ on the vertices, based on an interval

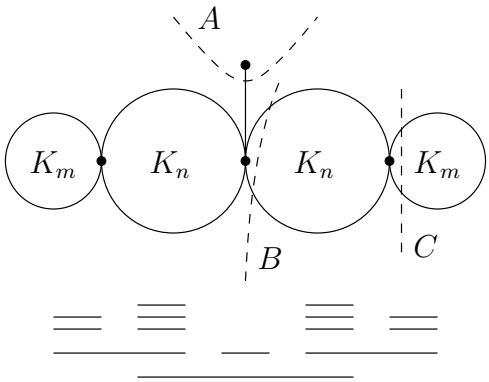


Figure 2.4: An interval graph with an unexpected sparsest cut

representation, such that we can prove that there is a sparsest cut $[S, \overline{S}]$ with $u \in S$, $v \in \overline{S} \Rightarrow u \prec v$? Such a cut is called a cut that *satisfies* $\prec$.

For a connected unit interval graph, a unit interval representation leads to a complete vertex order that is unique apart from direction, and interchangability of vertices u and v that have $N(u)-v = N(v)-u$. Note that for general interval graphs it is not possible to define a meaningful complete order this way, since one interval can be contained in another interval. However we can at least define a useful partial order on the vertices, when an interval representation is given: $u \prec v$ if $I_l(u) \leq I_v(v)$ and $I_h(u) \leq I_h(v)$. But also when we want to use this partial order to generalize Theorem 2.16, this leads to problems since this order depends on the chosen representation (consider e.g. connected interval graphs with a cut vertex v such that $G-v$ has many components). So we may want to study an even weaker statement: for all interval graphs G, does there exist an interval representation that leads to a partial order on the vertices, such that there is a sparsest cut satisfying this partial order? Figure 2.4 shows an example that shows that even this statement is not true: the graph shown has in essence a unique interval representation, and we will show that it has a single sparsest cut that is obtained by separating the single vertex 'in the middle' from the rest of the vertices.

The graph in Figure 2.4 has $2n + 2m - 2$ vertices, and consists of two K_n blocks, two K_m blocks and one K_2 block ($m \geq 2$, $n \geq 2$). Consider the three cuts illustrated in the figure. The first cut has $|[A, \overline{A}]| = 1$, and $|A| = 1$. The second has $|[B, \overline{B}]| = n-1$ and $|B| = m+n-2$. The third has $|[C, \overline{C}]| = m-1$ and $|C| = m-1$. Observe that regardless of the choice of n and m, one of these is a sparsest cut. These cuts have densities resp.

$$d(A, \overline{A}) = \frac{1}{2n + 2m - 3} \approx \frac{1}{2(n+m)},$$

$$d(B, \overline{B}) = \frac{n-1}{(m+n)(m+n-2)} \approx \frac{n}{(n+m)^2},$$

$$d(C,\overline{C}) = \frac{m-1}{(m-1)(2n+m-1)} = \frac{1}{2n+m-1} \approx \frac{1}{2n+m}.$$

Now it can be checked that when $m \geq 3$ and $n \geq m+2$, $[A,\overline{A}]$ is the unique sparsest cut, even though this cut does not satisfy any partial order corresponding to a representation. We conclude that a straightforward generalization of Theorem 2.16 to all interval graphs does not exist.

2.4 Sparsest cuts in complete bipartite graphs

In this section we give an explicit expression for the density of a sparsest cut of $K_{m,n}$, and in the proof of Theorem 2.18 construct all corresponding cuts. We prove these are sparsest cuts by comparing this density with all other cut densities in an efficient way. The expression for the sparsest cut density will show that $K_{m,n}$ is not a bottleneck graph when $m \geq 2$ and $n \geq 3$.

Theorem 2.18 *If $[S,\overline{S}]$ is a sparsest cut of $K_{m,n}$ with $m \leq n$ and $n \geq 2$, then* $d(S,\overline{S}) = \min\{\frac{1}{2}, \frac{m}{n+m-1}\}$.

Proof: First we give an expression for the density of an arbitrary edge cut $[S,\overline{S}]$ of $K_{m,n}$. Let $\{A,B\}$ be a bipartition of the vertices, with $|A| = m$, $|B| = n$. If $|S \cap A| = x$ and $|S \cap B| = y$, then

$$d(S,\overline{S}) = \frac{|[S\cap A, \overline{S}\cap B]| + |[S\cap B, \overline{S}\cap A]|}{|S||\overline{S}|} = \frac{x(n-y)+y(m-x)}{(x+y)(n+m-x-y)}.$$

We denote this function by $d(x,y)$. We want to find the minimum of $d(x,y)$ over all integer values of x and y with $0 \leq x \leq m$, $0 \leq y \leq n$, and $1 \leq x+y \leq n+m-1$. Because of the symmetry, we only have to consider values of x and y with $1 \leq x+y \leq (n+m)/2$ (otherwise we switch S and $\overline{S}$). First we analyze $d(x,y)$ without considering the integrality constraints for x and y.

Consider combinations of x and y with $x+y=c$ for a constant c. For fixed values of n, m and c we will denote this function as $d_c(x) = d(x,c-x)$. The denominator of $d_c(x)$ is a constant, so $d_c(x)$ is minimum when the numerator is minimum. Substituting $y = c-x$ gives

$$x(n-c+x) + (c-x)(m-x) = 2x^2 + (n-m-2c)x + mc.$$

This is minimum when $4x+n-m-2c = 0$, so when $x = \frac{c}{2} - \frac{n-m}{4}$ and $y = \frac{c}{2} + \frac{n-m}{4}$. If $\frac{n-m}{4} \geq \frac{c}{2}$, then this value for x is negative, so within our range the minimum of $d_c(x)$ occurs at $x=0$ and $y=c$ (since $d_c(x)$ is a polynomial of degree two).

We substitute these values of x and y into $d(x,y)$, to find the value of c with $1 \leq c \leq (n+m)/2$ for which the minimum is attained. First suppose $(n-m)/2 \leq 1$, and write $d = \frac{c}{2}$ and $e = \frac{n-m}{4}$. so for all values of c with

$1 \leq c \leq (n+m)/2$, we can substitute $x = d - e$ and $y = d + e$.

$$d(x,y) = \frac{(d-e)(n-d-e) + (d+e)(m-d+e)}{2d(n+m-2d)} =$$

$$\frac{dn - d^2 - de - en + de + e^2 + dm - d^2 + de + em - de + e^2}{2d(n+m-2d)} =$$

$$\frac{d(n+m-2d) - e(n-m) + 2e^2}{2d(n+m-2d)} =$$

$$\frac{d(n+m-2d) - 4e^2 + 2e^2}{2d(n+m-2d)} = \frac{z - 2e^2}{2z},$$

with $z = d(n+m-2d)$, or alternatively, $z = c(n+m-c)/2$. Note that z is strictly positive for our choices of c. If $(n-m)/4 = e = 0$, then the minimum of $d(x,y)$ is $\frac{1}{2}$, which occurs for all $x = \frac{c}{2}$ and $y = \frac{c}{2}$, with $1 \leq c \leq (n+m)/2$.

If $e \neq 0$, then we conclude from the expression above that $d(x,y)$ attains its minimum when $c(n+m-c)/2$ is as small as possible, so $c = 1$. We conclude that if $0 < n - m \leq 2$, $d(x,y)$ is minimum for $x = \frac{1}{2} - \frac{n-m}{4}$ and $y = \frac{1}{2} + \frac{n-m}{4}$.

Now suppose $(n-m)/2 \geq 1$. The same reasoning as above shows that when $c \geq (n-m)/2$, $d(x,y)$ is minimum when c is minimum, so $c = (n-m)/2$, and $x = 0$ and $y = c$. When $1 \leq c \leq (n-m)/2$, we showed that $d_c(x)$ is minimum when $x = 0$ and $y = c$. For all such pairs,

$$d(x,y) = \frac{cm}{c(n+m-c)} = \frac{m}{n+m-c}.$$

This function is minimum when c is minimum, so $c = 1$. We conclude that when $(n-m) \geq 2$, $d(x,y)$ is minimum for $x = 0$, $y = 1$. Now we will consider the integrality constraints for x and y.

If $n \geq m + 2$, the minimum occurs at integer values of x and y ($x = 0$, $y = 1$), so the best density is $m/(n+m-1)$.

If $n = m$, then the minimum occurs at all $x = y$, and the best density is $\frac{1}{2}$. (Here we need our assumption that $n \geq 2$, otherwise no integral solutions with $x = y$ exist within our range.)

Now consider the case $n = m + 1$. For fixed $c = x + y$, we know that $d_c(x) = (2x^2 + (n - m - 2c)x + mc)/f$ for some constant f, and that this function is minimum at $x = \frac{c}{2} - \frac{1}{4}$. Rounding to the closest values of x and y with $x + y = c$ gives $x = \frac{c}{2}$ and $y = \frac{c}{2}$ when c is even, and $x = \frac{c}{2} - \frac{1}{2}$ and $y = \frac{c}{2} + \frac{1}{2}$ when c is odd. Since $d_c(x)$ is a polynomial of degree two, this way of rounding gives the best integer values of $d_c(x)$. Now we calculate the densities for these integer values. If $x = y$, then

$$d(x,y) = \frac{x(n-x) + x(n-1-x)}{2x(2n-1-2x)} = \frac{2n-1-2x}{2(2n-1-2x)} = \frac{1}{2}.$$

If $y = x + 1$, then

$$d(x,y) = \frac{x(n-x-1)+(x+1)(n-1-x)}{(2x+1)(2n-1-2x-1)} = \frac{(2x+1)(n-x-1)}{2(2x+1)(n-x-1)} = \frac{1}{2}.$$

We conclude that $\frac{1}{2}$ is the minimum density, and that it is attained by many combinations of x and y, one for every value of $x + y$. □

In the following proof we use the notations from section 2.2.1.

Corollary 2.19 *Let $m \leq n$. $K_{m,n}$ is not a bottleneck graph if and only if $m \geq 2$ and $n \geq 3$.*

Proof: If $m = 1$, then $K_{m,n}$ is a star. If $m = n = 2$, then $K_{m,n} = C_4$. In both cases it can easily be verified that $K_{m,n}$ is a bottleneck graph.

Now suppose $m \geq 2$ and $n \geq 3$. We uniformly assign distances t to the edges of $K_{m,n}$, so every edge e has $t(e) = 1/mn$. Now there are nm vertex pairs with distance $1/mn$, and $n(n-1)/2 + m(m-1)/2$ vertex pairs with distance $2/mn$. We have

$$D(G) = \sum_{u,v \in V(G)} d_t(u,v) = \frac{nm + n(n-1) + m(m-1)}{mn}.$$

If $m = n$, then $d^{-1}(G) = 2$ (Theorem 2.18), and $D(G) = \frac{3n^2-2n}{n^2}$. Since $n \geq 3$, $n^2 > 2n$, so $D(G) > \frac{2n^2}{n^2} = d^{-1}(G)$, and $K_{m,n}$ is not a bottleneck graph.

If $m < n$, then $d^{-1}(G) = \frac{m+n-1}{m}$ (Theorem 2.18), and

$$D(G) = \frac{nm + n(n-1) + m(m-1)}{mn} > \frac{nm + n(n-1)}{nm} = d^{-1}(G),$$

so $K_{m,n}$ is again not a bottleneck graph. □

Chapter 3

The complexity of the Matching-Cut problem for planar graphs and other graph classes

3.1 Introduction

A *matching-cut* is an edge cut that is also a matching. The *Matching-Cut problem* is the problem to decide whether a given graph has a matching-cut:

Matching-Cut:
Instance: A graph $G = (V, E)$.
Question: Does G have a matching-cut?

Previous results Chvátal [17] studied the Matching-Cut problem under the name of the *Decomposable Graph Recognition problem*, and showed that the problem is $\mathcal{NP}$-complete for graphs with maximum degree four (using the 3-uniform Hypergraph Bicolorability problem), and gave a polynomial time algorithm to solve the problem for graphs with maximum degree at most three. Moshi [46] gave polynomial time algorithms for this problem for line graphs and quadrangulated graphs. Unaware of Chvátal's result, Patrignani and Pizzonia [48] also proved the $\mathcal{NP}$-completeness of the problem for graphs with maximum degree four using a different reduction, though from nearly the same problem (Not-All-Equal-3-Satisfiability). In addition they presented a linear time algorithm for series-parallel graphs. They also posed the question whether the problem is $\mathcal{NP}$-complete for the class of planar graphs.

A problem closely related to Matching-Cut is the problem of deciding whether a graph has a *stable cutset*. This is a vertex cut that contains no adjacent ver-

tices. The *line graph* $L(G)$ of $G = (V, E)$ has E as the vertex set, where $e \in E$ and $f \in E$ are adjacent in $L(G)$ if they are adjacent in G. Brandstädt et al. [14] observed the following.

Observation 3.1 (Brandstädt et al.) *If $L(G)$ has a stable cutset, then G has a matching-cut. If G has a matching-cut and has minimum degree at least two, then $L(G)$ has a stable cutset.*

In order to prove that the stable cutset problem is $\mathcal{NP}$-complete for line graphs with maximum degree five, Le and Randerath [39] adapted Chvátal's construction to prove the $\mathcal{NP}$-completeness of Matching-Cut for bipartite graphs in which all vertices in one class of the bipartition have degree three and all vertices in the other class have degree four.

Other results on matching-cuts appear in [25] and [26]: in [25] matching-cuts are studied in the context of network applications, and [26] contains an extremal result regarding matching cuts (see Section 3.5 and Chapter 4 for more details). In [50] a generalization of matching-cuts is studied: a matching-cut can alternatively be defined as a set $M = [S, \overline{S}]$ such that $G[M]$ contains no P_3. In [50], the complexity of the problem is studied when P_3 is replaced by a number of other (bipartite) graphs.

New results In this chapter the Matching-Cut problem is shown to be $\mathcal{NP}$-complete for planar graphs, using a reduction from Planar Graph 3-Colorability. This answers the question from Patrignani and Pizzonia [48]. This is done in Section 3.3. An observation by Moshi [46] then immediately shows that the problem is also $\mathcal{NP}$-complete for planar bipartite graphs, where one class of the bipartition contains only vertices of degree two. By changing the components used in the transformation, in Section 3.4 the $\mathcal{NP}$-completeness of the problem is proved for the classes of planar graphs with maximum degree four and planar graphs with girth five. The girth cannot be increased in this result; all planar graphs with girth six have a matching-cut.

Next, in Section 3.5, for some graph classes positive results for Matching-Cut are described. First for claw-free graphs a polynomial time algorithm for Matching-Cut is given. This generalizes the algorithm by Moshi [46] for line graphs. In addition, we give a short characterization of co-graphs that have a matching-cut, even though this is a subset of the quadrangulated graphs studied in [46]. We also observe that for any fixed k, a linear time algorithm can be constructed for Matching-Cut for graphs with treewidth bounded by k. Such an algorithm generalizes the algorithm for series-parallel graphs from [48] (series-parallel graphs have treewidth at most two [7]). Because of the focus of this chapter on planar graphs, for the class of outerplanar graphs an additional simple linear time algorithm is described, even though this is also a graph class with treewidth at most two [7].

3.2 Preliminaries

For an explanation of the notions related to planarity, we refer to [20]. In this chapter we will briefly use *directed graphs*: in a directed graph the edges are 2-tuples of vertices, instead of sets. For an edge (u, v), u is the *tail* of the edge, and v is the *head*. This is also called an edge *from u to v*. A directed graph D is an *orientation* of (multi) graph G if D and G have the same vertex set, and if there exists a bijection f from the edge set of D to the edge set of G, such that $f((u, v)) = \{u, v\}$. A (vertex) *k-coloring* of graph G is a function $f : V(G) \to \{1, \ldots, k\}$ such that adjacent vertices receive different values. These values are also called *colors*. A graph for which a k-coloring exists is called *k-colorable*. Cycles will also be denoted by their edge set.

A graph without a matching-cut is called *immune* (which is short for matching immune). The following two trivial observations will be used to analyze the possible matching-cuts of the graphs we use in our constructions.

Observation 3.2 *If edge e is part of a triangle of G, then there is no matching-cut $[S, \overline{S}]$ in G with $e \in [S, \overline{S}]$.*

Observation 3.3 *If e and f are two edges of an induced C_4 of G that do not share an end vertex and $[S, \overline{S}]$ is a matching-cut in G, then $e \in [S, \overline{S}] \iff f \in [S, \overline{S}]$.*

3.3 The $\mathcal{NP}$-completeness of Matching-Cut for planar graphs

In this section, the $\mathcal{NP}$-completeness of the Matching-Cut problem is proved when instances are restricted to the class of planar graphs. This problem will be called *Planar Matching-Cut*. The $\mathcal{NP}$-completeness proof is by a polynomial transformation from the following graph coloring problem:

Planar Graph 3-Colorability
Instance: A planar graph $G = (V, E)$
Question: Is G 3-colorable?

Theorem 3.4 (Garey, Johnson and Stockmeyer) *Planar Graph 3-Colorability is $\mathcal{NP}$-complete.*

This has been shown in [32]. In order to prove the $\mathcal{NP}$-completeness of Planar Matching-Cut, two intermediate problems will be used which are shown to be $\mathcal{NP}$-complete. For all problems we consider, membership of the class $\mathcal{NP}$ is obvious. First we show that Graph 3-Colorability is still $\mathcal{NP}$-complete when restricted to a smaller set of instances:

Planar Hamiltonian Graph 3-Colorability:
Instance: A planar graph $G = (V, E)$ with a Hamilton cycle $H \subseteq E$.
Question: Is G 3-colorable?

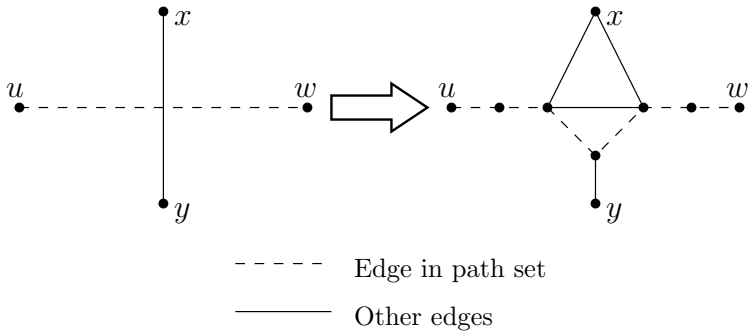


Figure 3.1: Removing a crossing

Lemma 3.5 *Planar Hamiltonian Graph 3-Colorability is $\mathcal{NP}$-complete.*

Proof: Let $G = (V, E)$ be a planar graph. We use G to construct a planar Hamiltonian graph $G' = (V', E')$ such that G is 3-colorable if and only if G' is 3-colorable. Using Theorem 3.4, this proves the $\mathcal{NP}$-completeness.

First observe that a graph G with a vertex v with $d(v) < 3$ is 3-colorable if and only if $G - v$ is 3-colorable. To construct $G' = (V', E')$, we will start with an embedding of $G = (V, E)$ (it is well-known that such an embedding can be found in polynomial time). First, edges will be added to make the graph Hamiltonian. If this introduces crossings, these are replaced by the crossing components that are defined below, restoring planarity. This replacement does not remove vertices, so $V \subseteq V'$. We then proceed by showing that any 3-coloring of G can be extended to a 3-coloring of G' and any 3-coloring of G' restricted to V is a 3-coloring of G.

Let $Q_1, \ldots, Q_k$ be a set of paths of G such that $\{V(Q_1), \ldots, V(Q_k)\}$ is a partition of $V(G)$. Note that such a path set always exists (paths may consist of one vertex), and can easily be found in polynomial time. Now choose an end vertex u of Q_1, and an end vertex v of Q_k. Add a new vertex w, and edges uw and vw. Connect the paths Q_1 and Q_k using w. If $k > 1$, then this decreases the number of paths in the path set. In that case, repeat this procedure (with new vertices in the role of w) until $k = 1$. If $k = 1$, then the procedure gives a Hamilton cycle. So this procedure constructs a graph with a given Hamilton cycle in polynomial time. Because $d(w) = 2$, in each step the addition of a vertex w has no influence on the 3-colorability of the graph.

Adding the edges uw and vw may not be possible without destroying planarity. Therefore, each addition is done as follows: in the embedding of G, the edges of the paths $Q_1, \ldots, Q_k$ do not divide the plane in more than one region. Therefore an edge uv can be drawn such that it does not cross any edges used in the paths, and crosses each of the other edges at most once. Subdivide uv using w, creating uw and vw. This subdivision can be done in such a way that on vw there are no crossings. Iteratively remove crossings on uw as follows: if

uw crosses an edge xy, replace this crossing with the crossing component shown in Figure 3.1 (without introducing new crossings). This crossing component has the following two properties: in a 3-coloring, u and w can have any color assigned to them, independent of the color assigned to the other vertex and the vertices x and y. Furthermore, x and y will be colored differently. Note that all *new* vertices in the crossing component are part of the new path from u to w and that xy was not part of the path set, so we still have a vertex disjoint set of paths containing all vertices. Repeating this procedure removes all crossings.

Let G'' be the graph obtained after adding edges uw and vw and removing all crossings in the described way. Then any 3-coloring of G can be extended to a 3-coloring of G'', because the new neighbors of u and v have degree two. Any 3-coloring of G'' is a 3-coloring of G when restricted to V, because in a 3-coloring of a crossing component different colors are assigned to x and y.

So if this (polynomial time) procedure of adding edges and removing crossings is repeated until a planar graph G' with a Hamilton cycle is constructed, G' is 3-colorable if and only if G is 3-colorable. This completes the $\mathcal{NP}$-completeness proof. □

Before we can use Graph 3-Colorability to prove the $\mathcal{NP}$-completeness of Planar Matching-Cut, we transform the problem into an artificial, but more suitable vertex coloring problem in which every vertex is incident with only one vertex that has to be colored with a different color, and at most two that have to be colored with the same color:

Segment 3-Colorability
Instance: A set of vertices V and three disjoint edge sets A, B and C such that $G = (V, A \cup B \cup C)$ is a 3-regular planar multigraph with Hamilton cycle $A \cup B$.
Question: Can we find a color function $f : V \to \{1, 2, 3\}$ such that if $uv \in A$ then $f(u) = f(v)$ and if $uv \in C$ then $f(u) \neq f(v)$?

An instance of Segment 3-Colorability will be denoted by the tuple $G = (V, A, B, C)$. Note that $G = (V, A \cup B \cup C)$ is a multigraph, so if for instance B and C both contain an edge between vertices u and v, in G these are considered to be different edges. Still, we will denote such an edge as uv if no confusion is possible. Edges in the sets A, B and C will be called respectively A-edges, B-edges and C-edges. If $G = (V, A, B, C)$ is an instance of Segment 3-Colorability, the components of the graph (V, A) are called *segments*. All vertices of one segment receive the same color in a solution of this problem, and therefore the problem is called segment coloring. Note that C is a perfect matching in G.

Using lemma 3.5, we can prove the $\mathcal{NP}$-completeness of Segment 3-Colorability:

Lemma 3.6 *Segment 3-Colorability is $\mathcal{NP}$-complete.*

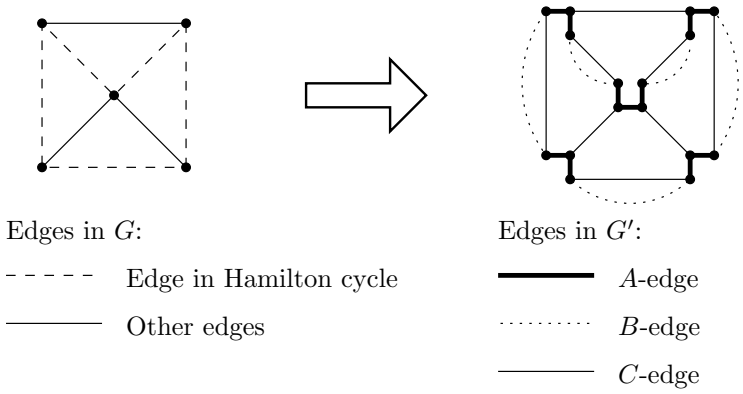


Figure 3.2: An example of the transformation in the proof of Lemma 3.6

Proof: We will construct an instance G' of Segment 3-Colorability from an instance G of Planar Hamiltonian Graph 3-Colorability such that:

- the vertices of G correspond to the segments of G',
- the edges of G correspond to the edge set C of G': G' has a C-edge between vertices of segments x and y if and only if there is an edge between the vertices of G corresponding to x and y, and
- the vertex ordering given by the Hamilton cycle H in G is the same as the segment ordering given by the Hamilton cycle $A \cup B$ in G'.

Using the first two properties above, it is easy to see that Segment 3-Colorability on G' is equivalent with Planar Hamiltonian Graph 3-Colorability on G. An example of this transformation is shown in Figure 3.2.

Consider a planar embedding of G (this can be found in polynomial time). The transformation uses the following steps, which are shown in Figure 3.3:

1. For each vertex i in G, we can use the embedding to number the incident edges clockwise from 1 to $d(i)$ (such that edge j and edge $j+1$ are on the boundary of the same face). For every incident edge j, subdivide it using vertex v_{ij}. Add edges $v_{ij}v_{i(j+1)}$ $(j = i, \ldots, d(i) - 1)$ and $v_{id}v_{i1}$, where $d = d(i)$. Delete vertex i. The edges corresponding to the edges of G form the edge set C, and the new edges form the edge set A.
2. Double every edge in C that corresponds to an edge in G that is used in the Hamilton cycle H. The new edges form the edge set B.
3. For every i, suppose w.l.o.g. that v_{i1} and v_{ij} are the two vertices on the cycle that are incident with edges in B. Let $d = d_G(i)$. Now delete the edges $v_{i(j-1)}v_{ij}$ and $v_{i1}v_{id}$ and add an edge $v_{i(j-1)}v_{id}$. The new edge belongs to the edge set A. Now the edges of A form a Hamilton path from v_{i1} to v_{ij} in the subgraph induced by $v_{i1}, \ldots, v_{id}$.

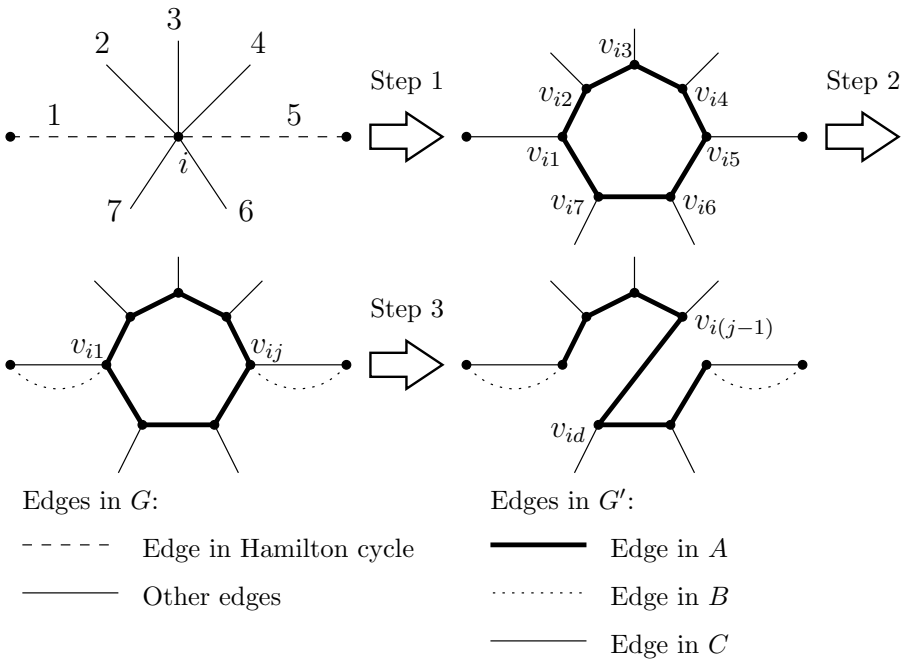


Figure 3.3: The transformation steps in the proof of Lemma 3.6

This transformation yields a 3-regular graph G'. All steps used in the transformation preserve planarity so G' is planar. (These steps are: subdivision of edges, doubling edges, adding edges between vertices that are on the boundary of the same face and deleting vertices and edges.)

For every i, vertices v_{ij} $(j = 1, \ldots, d_G(i))$ induce a component of $(V(G'), A)$ and therefore receive the same color in a feasible solution of Segment 3-Colorability. If (and only if) an edge ij is present in G, a C-edge $v_{ix}v_{jy}$ is present in G' for some x and y, so segments i and j receive different colors in a feasible Segment 3-Colorability solution for G'. We conclude that a vertex 3-coloring of G can be transformed into a segment 3-coloring of G' and vice versa. So Segment 3-Colorability is $\mathcal{NP}$-complete. □

Now, using Lemma 3.6 we can prove the $\mathcal{NP}$-completeness of Planar Matching-Cut.

Theorem 3.7 *Planar Matching-Cut is $\mathcal{NP}$-complete*

Proof: Let (V, A, B, C) be an instance of Segment 3-Colorability with $G = (V, A \cup B \cup C)$. We will transform this instance into an instance G' of Planar Matching-Cut. For every vertex in V we will introduce a *vertex component* in G'. (Throughout this proof, we do not use the graph theoretical meaning of the word component.) If two vertices in V are joined by an A-edge, the corresponding vertex components will be connected using a *vertex connection component*. If two vertices in V are joined by a B-edge, the corresponding vertex components will be connected using a *segment connection component*. In order to properly label the components that will be introduced, first label the *vertices* of G with labels $1, \ldots, k_1$. Label the *edges* in A with labels $k_1 + 1, \ldots, k_2$. Label the *edges* in B with $k_2 + 1, \ldots, k_3$. Note that for C-edges no components will be introduced.

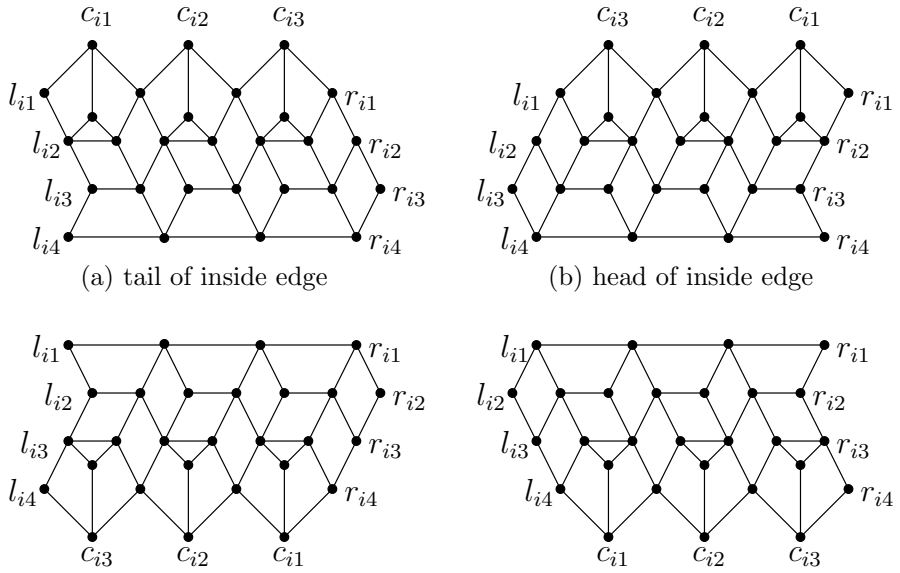
Consider an embedding of G. Orient all edges of the Hamilton cycle $A \cup B$ anticlockwise with regard to the embedding. Orient all edges of C arbitrarily. Since the edge set $A \cup B$ gives a Hamilton cycle, in the embedding of G, this Hamilton cycle divides the plane into two regions. So we can divide the edges of C into two categories using this Hamilton cycle: *inside edges* and *outside edges*. Because every vertex i in G $(i = 1, \ldots, k_1)$ is incident with exactly one edge $e \in C$, we can use this to define four variants of vertex components:

1. If e is an inside edge and i is incident with the tail of e, introduce a vertex component as shown in Figure 3.4(a).

2. If e is an inside edge and i is incident with the head of e, introduce a vertex component as shown in Figure 3.4(b).

3. If e is an outside edge and i is incident with the tail of e, introduce a vertex component as shown in Figure 3.4(c).

4. If e is an outside edge and i is incident with the head of e, introduce a vertex component as shown in Figure 3.4(d).



(a) tail of inside edge (b) head of inside edge

(c) tail of outside edge (d) head of outside edge

Figure 3.4: Four variants of vertex components for G'

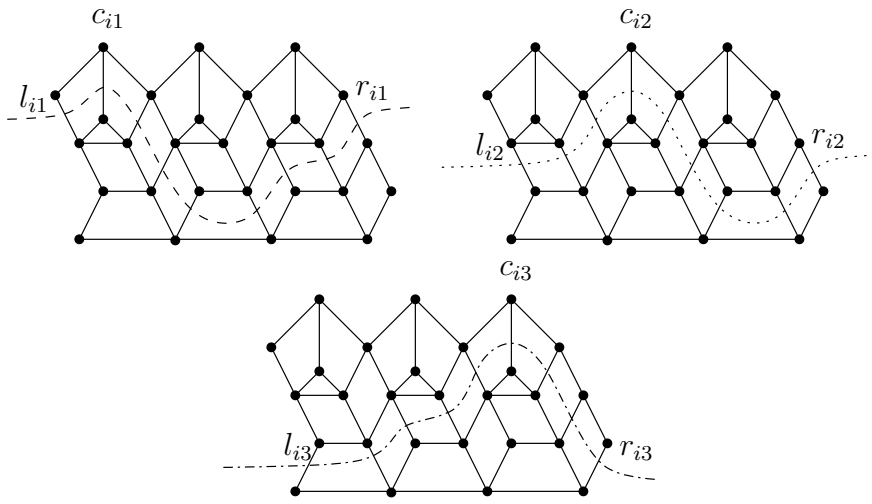


Figure 3.5: The three possible matching-cuts through a vertex component

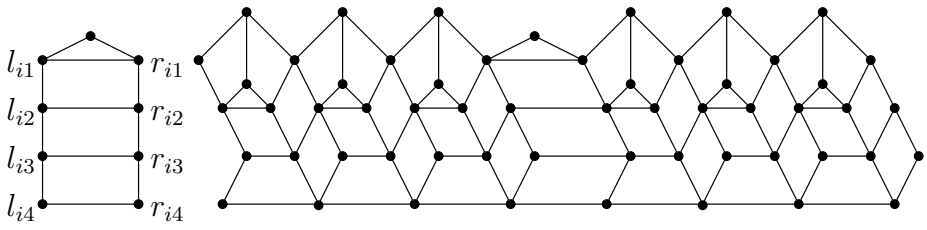


Figure 3.6: A vertex connection component and vertex connection

There are at least three possible matching-cuts through these components, which are shown in Figure 3.5. We use the two observations from Section 3.2 to show that these are the only possible matching-cuts: if one edge of a matching-cut through a vertex component is given, the other edges in the matching-cut can be derived, because all edges are on a (facial) 4-cycle. Observe that if one of the edges that is not part of the three given matching-cuts is in a matching-cut, this will imply that an edge of one of the triangles is part of the matching-cut, a contradiction. Therefore these are the only three possible matching-cuts through a vertex component. Note that no two of these sets can be part of the same matching-cut.

If edge $l_{ij}l_{i(j+1)}$ is in the matching-cut, we will say that *this vertex component is cut by cut j ($j = 1, 2, 3$)*. Below, this will correspond to a coloring of vertex i in G with color j. Observe that in this case, $r_{ij}r_{i(j+1)}$ is also in the matching-cut, and vertex c_{ij} is incident with one of the edges in the matching-cut, whereas c_{ik} for $k \neq j$ is not.

For each edge i in A ($i = k_1 + 1, \ldots, k_2$), we introduce a vertex connection component as shown on the left in Figure 3.6. It is easy to see that there are exactly three possible matching-cuts through these components. Observe that if edge $l_{ij}l_{i(j+1)}$ is in the matching-cut, then edge $r_{ij}r_{i(j+1)}$ also is.

If in G, there is an A-edge k directed from i to j, the two vertex components i and j will be connected by vertex connection component k as follows (see Figure 3.6 for an example): identify r_{il} with l_{kl} for $l = 1, 2, 3, 4$. This new vertex is called r_{il} again. Now, for $l = 1, 2, 3$ there are two edges from r_{il} to $r_{i(l+1)}$. Delete one of these. Next, identify r_{kl} with l_{jl} for $l = 1, 2, 3, 4$ and call the new vertices l_{jl}. Also delete one of all the double edges introduced here. Such a *vertex connection* has the following property:

Property 3.8 *If two vertex components i and j are connected by a vertex connection component, then for any matching-cut: i is cut by cut l if and only if j is cut by cut l ($l = 1, 2, 3$).*

Clearly, to achieve this property, vertex components can also be connected without using a vertex connection component, but for the alternative constructions in the next section, vertex connection components are useful.

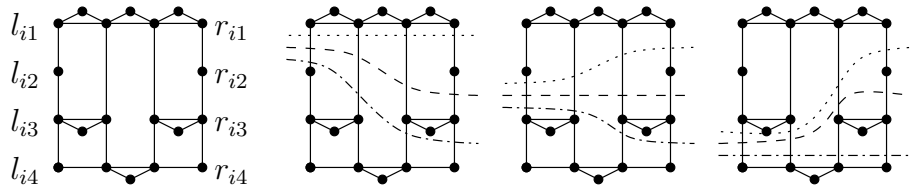


Figure 3.7: A segment connection component and the nine possible matching-cuts

For each edge i in B ($i = k_2 + 1, \ldots, k_3$), we introduce a segment connection component as shown in Figure 3.7.

Because the edges in triangles cannot be part of a matching-cut, there are nine possible matching-cuts through these components, all of which cut exactly one edge $l_{ij}l_{i(j+1)}$ and one edge $r_{ik}r_{i(k+1)}$. There is one cut for every combination of j and k ($j = 1, 2, 3$, $k = 1, 2, 3$).

If in G there is a B-edge from i to j, we can connect the vertex components i and j using a segment connection component in the same way as described above for vertex connection components. This *segment connection* has the following property:

Property 3.9 *If two vertex components i and j are connected by a segment connection component, then for any matching-cut: i is cut by this matching-cut if and only if j is cut by this matching-cut. Any combination of cuts through i and j is possible.*

The C-edges of G determine the last type of connection between vertex components: if $ij \in C$, identify c_{il} with c_{jl} ($l = 1, 2, 3$). This connection is called an *edge connection*. We know that if vertex component i is cut by cut l, then c_{il} is incident with an edge of this matching-cut. So because a matching-cut is a matching, this connection has the following property:

Property 3.10 *If two vertex components i and j are connected by an edge connection, then for any matching-cut and $l = 1, 2, 3$: it is not possible that i is cut by cut l and j is cut by cut l.*

Let G' be the graph that is constructed by introducing vertex components for every vertex in G and connecting them with vertex connection components, segment connection components and edge connections for every edge in respectively A, B and C as described above. Using the observed properties of these connections, G' has the following properties:

- Because $A \cup B$ gives a Hamilton cycle in G, and the fact that the different components have no matching-cuts other than the indicated cuts, we use

Property 3.8 and Property 3.9 to conclude that if G' has a matching-cut, this matching-cut cuts every vertex component.

- Property 3.8 shows that if G' has a matching-cut, all vertex components that correspond to vertices in the same segment of G are cut by the same cut l.

- By Property 3.10, in any matching-cut, two vertex components that correspond to vertices in segments that are joined by a C-edge are cut by different cuts.

- Property 3.9 shows that segment connection components do not impose additional constraints on the possible combinations of cuts through vertex components.

Now it is easy to see how any matching-cut in G' corresponds to a proper segment 3-coloring of G and vice versa.

The only thing left to prove is that G' is indeed an instance for Planar Matching-Cut. Therefore, we show that G' is planar.

$|V|$ vertex components, $|A|$ vertex connection components and $|B|$ segment connection components can be drawn in the plane, with inner faces as shown in the figures, and all sharing outer face F. Because $A \cup B$ is a Hamilton cycle in G, we can one by one make the vertex connections and segment connections without destroying the planarity: we can maintain an embedding where all inner faces of the components remain the same. Here we use the orientation of the Hamilton cycle $A \cup B$: every vertex is the tail of one edge on this cycle, and the head of one other edge. So for every vertex component i, one vertex or segment connection is made using the vertices l_{ij}, and one vertex or segment connection is made using the vertices r_{ij}. Since all except the last of these connection operations join two disconnected parts of the graph, making these connections does not change the number of faces. Only the last connection that 'closes the cycle' splits face F into two faces: F_1 and F_2. Assume w.l.o.g. that every vertex l_{i1} and r_{i1} is on the boundary of F_1, and every vertex l_{i4} and r_{i4} is on the boundary of F_2.

If a C-edge ij is present, then vertex components i and j are either of type (a) and (b) as shown in Figure 3.4, or of type (c) and (d) as shown in this figure. Therefore either c_{ik} and c_{jk} $(k = 1, 2, 3)$ are all on the boundary of F_1, or they are all on the boundary of F_2. Also, edges of C do not cross in the embedding of G. Because of this, all such vertex pairs can be identified without destroying the planarity. This gives an embedding of G'.

It can be checked that G' is a simple graph. Since the given transformation is polynomial, and Planar Matching-Cut is within $\mathcal{NP}$, Planar Matching-Cut is $\mathcal{NP}$-complete. □

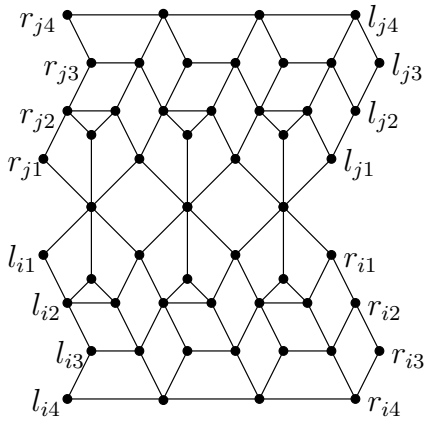


Figure 3.8: Two vertex components form an edge component

3.4 The $\mathcal{NP}$-completeness of Matching-Cut for more restricted graph classes

3.4.1 Planar graphs with maximum degree four

As Chvátal proved the Matching-Cut problem to be $\mathcal{NP}$-complete for graphs with maximum degree four, the question arises whether the problem is also $\mathcal{NP}$-complete for planar graphs with maximum degree four. To prove that this is indeed the case, we outline how the construction used in the proof of Theorem 3.7 should be altered such that the resulting graph G' is a planar graph with maximum degree four, and still has the properties needed in the proof.

For this alteration, in the graph G' from the proof of Theorem 3.7, we will replace all segment connection components by new segment connection components. The vertex components will be replaced pairwise: in G', every vertex component occurs in a pair as shown in Figure 3.8 (this pair corresponds to an inside C-edge in the Segment 3-Colorability instance. For outside edges the labeling is different, but this is not important). Call this subgraph, composed of vertex components i and j, *edge component* ij.

Theorem 3.11 *Matching-Cut is $\mathcal{NP}$-complete when restricted to planar graphs with maximum degree four.*

Proof: Consider the graph H shown in Figure 3.9. H is obviously planar and has maximum degree four. Furthermore, the vertices l_{ik}, r_{ik}, l_{jk} and r_{jk} ($k = 1, 2, 3, 4$) have smaller degree such that further connections are possible. With a little effort it can be checked (using the observations in Section 3.2) that the possible combinations of matching-cuts through H are similar to the possible combinations of matching-cuts through the edge component in Figure 3.8. There

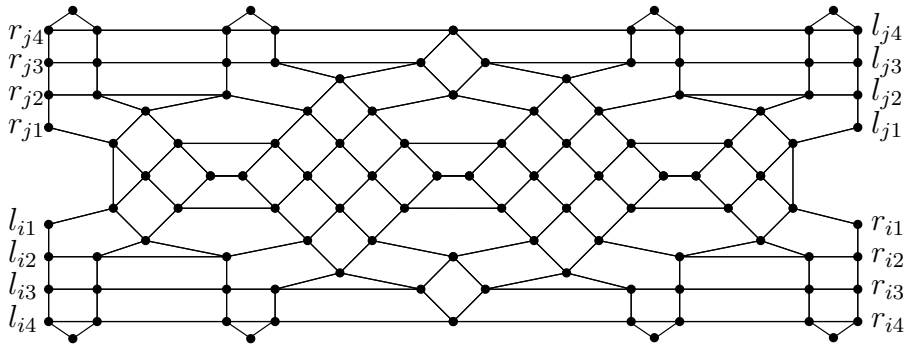


Figure 3.9: Part of an edge component with maximum degree four

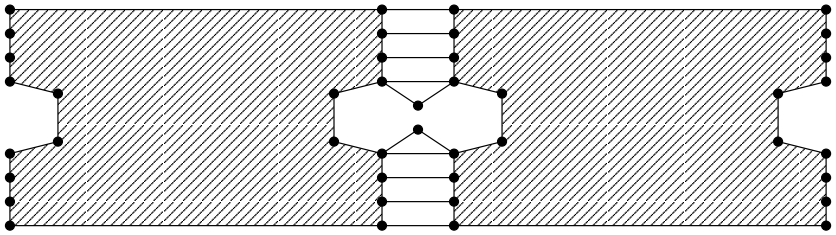


Figure 3.10: An edge component with maximum degree four

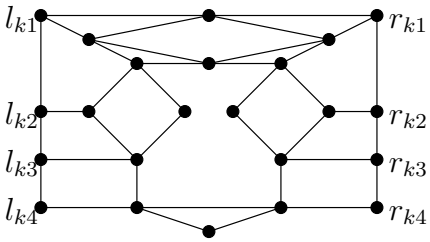


Figure 3.11: A segment connection component with maximum degree four

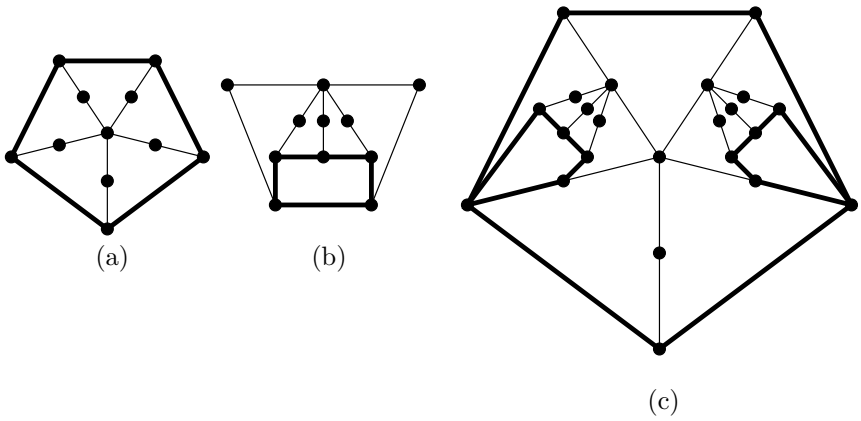


Figure 3.12: Constructing an immune graph with girth five

is one difference: the matching-cuts are 'reversed', so for instance the matching-cut that contains the edge $l_{i1}l_{i2}$ also contains the edge $r_{i3}r_{i4}$ and the matching-cut that contains the edge $l_{i3}l_{i4}$ also contains the edge $r_{i1}r_{i2}$. Therefore we replace edge components in G' by a pair of these H-graphs, connected by vertex connection components as illustrated in Figure 3.10. Note that this does not increase the maximum degree of the resulting component above four.

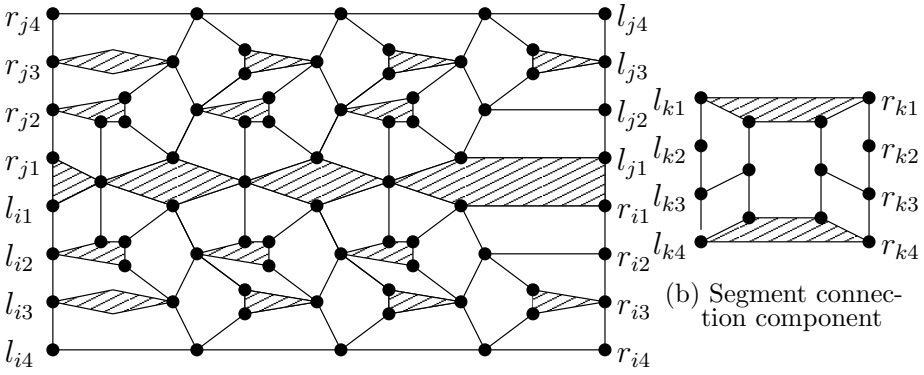
Replace each segment connection component (Figure 3.7) in G' by the component shown in Figure 3.11. This component is again planar, has maximum degree four and has a set of matching-cuts equivalent to the nine matching-cuts of the original segment connection component. Vertex connection components do not have to be replaced.

If we replace all subgraphs of G' exactly as described, then next to edge components corresponding to outside edges, vertices of degree five will occur. This problem can easily be solved by inserting extra vertex connection components (with a triangle between l_{l4} and r_{l4} instead of between l_{l1} and r_{l1}) and/or deleting vertex connection components.

This shows how the construction can be altered such that the resulting graph has maximum degree four and still has all the necessary properties. □

3.4.2 Planar graphs with large girth

After we observe that the immune graphs that we have seen all contain small cycles (triangles), the question arises whether the Matching-Cut problem is still $\mathcal{NP}$-complete for graphs with large girth. First consider simple graphs without triangles or, for a stronger result, bipartite simple graphs. Moshi [46] observed that if in a Matching-Cut instance G every edge is replaced by two parallel edges and both of those edges are subdivided with one vertex, this gives a bipartite Matching-Cut instance G' that is equivalent with G. This operation does not



(a) Edge component

(b) Segment connection component

Figure 3.13: Components for the construction with girth five

destroy planarity, so Matching-Cut is $\mathcal{NP}$-complete for planar bipartite graphs, where one side of the bipartition contains only vertices with degree two.

Before we can prove that the Matching-Cut problem is $\mathcal{NP}$-complete for planar graphs with girth five, we must find such a graph that is immune. In Figure 3.12(a), a planar graph with girth five is shown. It can be checked that none of the bold edges can be part of a matching-cut. A different embedding of this graph is shown in Figure 3.12(b). If we draw this second embedding into two of the faces of the first embedding, we obtain the planar graph with girth five in Figure 3.12(c). Again, the bold edges can not be part of a matching-cut. Using this observation it can be checked that this graph has no matching-cuts. In a similar way, for any $d \geq 5$, embeddings of immune planar graphs with girth five can be constructed such that the outer face has degree d.

Theorem 3.12 *Matching-Cut is $\mathcal{NP}$-complete when restricted to planar graphs with girth five.*

Proof: We will outline how to change the proof of Theorem 3.7 such that the resulting graph is a planar graph with girth five. Consider the components in Figure 3.13 (the role of the shaded faces will be explained below). Replace edge components and segment connection components in the graph G' from the proof of Theorem 3.7 by these new components. Recall that vertex connection components do not have to be used. Now, edges on the boundary of the shaded faces can be subdivided until all 2-cycles, 3-cycles and 4-cycles are removed (observe that in the graph obtained from G', all these small cycles contain such an edge). To ensure that none of the edges around the shaded faces can be part of a matching-cut, insert in every shaded face of degree d an embedded immune planar graph with outer face degree d, as constructed above. By this 'insert' operation we mean the operation we also used to obtain the graph in Figure 3.12(c). It can be checked that the resulting graph again has girth five. This

yields a planar graph of girth five that has the desired matching-cut properties.□

In this result, the girth cannot be increased; we will show that all planar graphs with girth at least six have a matching-cut. In [26] it is shown that for every immune graph on n vertices with m edges, $m \geq \lceil 3(n-1)/2 \rceil$, and that this bound is best possible. For an embedding of a planar graph G with girth at least six we have $m \geq 3k$, where k is the number of faces in the embedding. Euler's formula states that $n - m + k = 2$ for every embedding. This gives $n - \frac{2}{3}m \geq 2$, or $m \leq 3(n-2)/2$, so G has a matching-cut.

3.5 Graph classes for which Matching-Cut is easy

3.5.1 Overview

For some graph classes the Matching-Cut problem can be solved efficiently: Chvátal [17] described an algorithm for graphs with maximum degree at most three, and Patrignani and Pizzonia [48] described an algorithm for series-parallel graphs. Moshi [46] gave algorithms for line graphs and quadrangulated graphs. *Quadrangulated graphs* are graphs without induced cycles of length five or more. Farley and Proskurowski [26] showed that all graphs G with $|E(G)| < \lceil 3(|V(G)|-1)/2 \rceil$ have a matching-cut. In Chapter 4, we show that for graphs with $|E(G)| = \lceil 3(|V(G)|-1)/2 \rceil$, the Matching-Cut problem can be solved efficiently (Corollary 4.72). Below a few other positive results on the Matching-Cut problem are described.

3.5.2 Claw-free graphs

A *claw* is a $K_{1,3}$. A *claw-free graph* is a graph without induced claws. The following characterization of claw-free graphs that have a matching-cut leads to a polynomial time decision algorithm for Matching-Cut for these graphs. The theorem below is only formulated for non-trivial instances. A *non-trivial component* is a component that contains at least one edge.

Theorem 3.13 *Let $G = (V, E)$ be a connected claw-free graph with minimum degree at least two. Let $M \subseteq E$ be the set of edges that are not part of a triangle. G has a matching-cut if and only if*

- *One of the components of $G[M]$ is a path with length at least three or a cycle, or*
- *$G - M$ contains multiple non-trivial components.*

Proof: Since G is claw-free, $G[M]$ has maximum degree at most two, and thus all of its components are paths or cycles. Vertices with degree two in $G[M]$ also have degree two in G, again because G is claw-free.

First consider the case that $G[M]$ contains a cycle C. By definition of M, C has length at least four. All of the vertices of C have degree two in G, so

C is a component of G. Since G is connected, $G = C$. Now G clearly has a matching-cut, which proves the statement when $G[M]$ contains a cycle.

From now on we may assume that $G[M]$ contains no cycles, so every component of $G[M]$ is a path. If one of the components of $G[M]$ is a path P with length at least three, then consider two non-adjacent edges of P. Since all internal vertices of P have degree two in G, these two edges form a matching-cut in G.

If $G-M$ has multiple non-trivial components, then construct an edge set M' by choosing one edge from each component of $G[M]$. This is clearly a matching. Choose two vertices u and v that are part of different non-trivial components of $G - M$. Every (u, v)-path contains at least one edge of M. Since internal vertices of paths in $G[M]$ have degree two in G, an (u, v)-path that contains one edge from a component of $G[M]$, contains all edges of this component, and thus at least one edge of M'. We conclude that M' disconnects the graph, and therefore contains an edge cut, which is a matching-cut.

Finally we argue that if G has a matching-cut $M' = [S, \overline{S}]$, then one of the two conditions is satisfied. First observe that $M' \subseteq M$. If the matching M' contains multiple edges of one component of $G[M]$, then clearly this component is a path with length at least three (recall that $G[M]$ contains no cycles). If not, then there is a component in $G[M]$ that is a path with end vertices $u \in S$ and $v \in \overline{S}$. Since u and v both have degree at least two in G, u and v are both part of a non-trivial component of $G - M$. u and v are part of different components of $G - M'$, and therefore part of different components of $G - M$. We conclude that $G - M$ contains multiple non-trivial components. □

We remark that in the $\mathcal{NP}$-completeness proof of Chvátal [17] a $K_{1,4}$-free graph is constructed, so for $K_{1,4}$-free graphs the Matching-Cut problem is $\mathcal{NP}$-complete. Line graphs are claw-free, so this generalizes the result in [46].

3.5.3 Co-graphs

Co-graphs are graphs without an induced P_4. This is a subset of quadrangulated graphs, for which in [46] a polynomial time for Matching-Cut algorithm is given. The following characterization of immune co-graphs clearly leads to a polynomial time Matching-Cut algorithm for co-graphs.

Theorem 3.14 *Connected co-graphs with minimum degree at least two that are not equal to C_4 are immune.*

Proof: Let $G = (V, E)$ be a connected co-graph with $\delta(G) \geq 2$. Let $M \subseteq E$ be the set of edges that are not part of a triangle. If $M = \emptyset$, then G does not have a matching-cut. Otherwise consider $xy \in M$. If $d(x) \geq 3$, then x has two neighbors u and v which are not a neighbor of y. Also, y has a neighbor w which is not a neighbor of x. Because G is P_4-free, $uw \in E$ and $vw \in E$. If $[S, \overline{S}]$ is a matching-cut and $xy \in [S, \overline{S}]$, then by Observation 3.3, $\{uw, vw\} \subseteq [S, \overline{S}]$, in which case it is not a matching. So only edges xy with $d(x) = 2$ and $d(y) = 2$ that are not part of a triangle can be part of a matching-cut; in this case there

are $v, w \in V$ with $vx \in E$ and $wy \in E$ such that vw is also in the matching-cut. This shows that a connected co-graph G with $\delta(G) \geq 2$ has a matching-cut if and only if $G = C_4$. □

3.5.4 Graphs with fixed bounded treewidth

For an introduction to treewidth, we refer to [7]. In [2] it is shown that all graph properties definable in monadic second-order logic (MSOL) can be decided in linear time for classes of graphs with bounded treewidth, when a tree-decomposition is given. In addition, in [8] a linear time algorithm to find such a tree-decomposition is described. Combining these results, in order to show that Matching-Cut can be solved in linear time for graphs with fixed bounded treewidth, it suffices to show that the graph property of having a matching-cut can be expressed in MSOL.

MSOL is a logical language for graph problems in which formulas can be built using the following constituents:

- The logical connectives $\neg$, $\wedge$, $\vee$, $\leftrightarrow$, $\rightarrow$.
- Quantification over vertices, edges, sets of vertices and sets of edges (e.g. $\exists v \in V(G)$, $\forall M \subseteq E(G)$).
- Adjacency tests for vertex pairs and incidence tests for a vertex and an edge.
- Tests whether a vertex is member of a vertex set and whether an edge is member of an edge set.

Theorem 3.15 *Matching-Cut can be solved in linear time for any graph class with bounded treewidth.*

Proof: We show that the property of having a matching-cut can be expressed in MSOL. Graph $G = (V, E)$ has a matching-cut of the form $[V_1, V_2]$ if and only if:

$$\exists V_1 \subseteq V : \exists V_2 \subseteq V : (V_1 \cap V_2 = \emptyset) \wedge (V_1 \cup V_2 = V) \wedge \neg(V_1 = \emptyset) \wedge \neg(V_2 = \emptyset) \wedge$$
$$\neg(\exists u \in V_1 : \exists v \in V_2 : \exists w \in V_2 : (uv \in E) \wedge (uw \in E)) \wedge$$
$$\neg(\exists u \in V_2 : \exists v \in V_1 : \exists w \in V_1 : (uv \in E) \wedge (uw \in E)).$$

The first four predicates can be rewritten as follows:

$$V_1 \cap V_2 = \emptyset \Longleftrightarrow \neg\exists v \in V : v \in V_1 \wedge v \in V_2,$$
$$V_1 \cup V_2 = V \Longleftrightarrow \forall v \in V : v \in V_1 \vee v \in V_2,$$
$$V_1 = \emptyset \Longleftrightarrow \forall v \in V \neg(v \in V_1).$$

Quantified formulas like $\exists v \in V_1 : P(v)$ can be rewritten as $\exists v \in V : v \in V_1 \wedge P(v)$. Formulas of the form $\exists u \in V_1 : \exists v \in V_2 : (uv \in E)$ can also be

rewritten in MSOL. Therefore the property of having a matching-cut can be expressed in MSOL. From the aforementioned results in [2] and [8], it now follows that Matching-Cut is decidable in linear time for graphs with fixed bounded treewidth. □

Examples of graph classes with bounded treewidth are outerplanar graphs, Halin graphs and series-parallel graphs, which have treewidth at most two, three, resp. two [7]. So this contains the result of Patrignani and Pizzonia on series-parallel graphs [48], though the algorithm that follows from this MSOL problem description is not nearly as practical. However, it is probably tedious but not hard to generalize the method described in [48] to graphs with fixed bounded treewidth, which gives a more practical algorithm.

3.5.5 Outerplanar graphs

In the previous section, it was shown that a linear time algorithm for Matching-Cut for outerplanar graphs exists, though the method used in the previous section will probably not give a practical algorithm. In this section a practical algorithm is described.

A graph is *outerplanar* if it is planar and has an embedding such that every vertex is on the boundary of the outer face (an *outerplanar embedding*). Let *boundary edges* be the edges on the boundary of the outer face. Let *internal edges* be all other edges. A graph without bridges has a matching-cut if and only if one of the maximal 2-connected subgraphs has a matching-cut, so we will only consider 2-connected graphs. Checking whether a graph has bridges and finding all maximal 2-connected subgraphs can be done in linear time [54], so the linear time algorithm below for 2-connected outerplanar graphs translates to a linear time algorithm for all outerplanar graphs.

Theorem 3.16 *Algorithm 1 is a linear time algorithm for Matching-Cut for 2-connected outerplanar graphs.*

Proof: We will first show that the steps of the algorithm can be implemented (e.g. in Step 2, such a face F can be chosen), and then deduce the time complexity of the algorithm. Finally, we will prove the correctness; we will show that the conclusions drawn in Step 3 and 4 are correct, and that the algorithm terminates with an answer.

For outerplanar graphs G it can be checked that if D is the dual graph of an outerplanar embedding of G, and $v \in V(D)$ corresponds to the outer face of G, then $D - v$ is a tree. Therefore in step 2, a face F can always be chosen. (F corresponds to an end vertex of $D - v$.) Observe that after step 6, the resulting graph is again 2-connected, outerplanar and has all boundary edges marked and all internal edges unmarked.

We will now prove that the algorithm is a linear time algorithm. In linear time, outerplanar graphs can be recognized, and an outerplanar embedding can be found [45]. We need a data structure where for each face, the facial cycle is

Algorithm 1 Outerplanar Matching-Cut

Input: A 2-connected outerplanar graph G.

1. Start with an outerplanar embedding of G. Mark all boundary edges with 1.
2. Choose a face F incident with at most one internal (unmarked) edge, that is not equal to the outer face.
3. **If** F is incident with two non-adjacent edges marked with 1, **then return** YES. Stop.
4. **If** there are no unmarked edges around F, **then return** NO. Stop.
5. Let e be the unique unmarked edge around F.
   If there is an edge f around F, non-adjacent to e, and marked with 1, **then** mark e with 1, **else** mark e with 0.
6. Delete all vertices on the boundary of F not incident with e, and **goto** step 2.

stored, and for each edge, the two incident faces are returned in constant time. In addition, the number of internal edges around a face has to be stored, and a list of faces incident with at most one internal edge has to be maintained. Observe that this data structure can be built in linear time. This shows that Step 1 can be implemented in linear time.

Let k be the number of faces of the input graph, and let m be the number of edges. In every iteration of the algorithm (Steps 2–6), the number of faces decreases by one, so at most k iterations are done. Using the list of faces incident with at most one internal edge, Step 2 can be implemented such that it takes constant time. The same holds for Step 6; the vertices are only deleted to facilitate the correctness proof below. Steps 3–5 cannot be implemented in constant time. However, using the facial cycles, an $O(d)$ implementation exists, where d is the degree of the face F. In addition, every edge is considered at most twice during these steps: boundary edges of the input graph are considered at most once, and internal edges are considered at most twice. We conclude that the time complexity of the algorithm is $O(k + m)$, hence it is a linear time algorithm.

Now we will prove that if the algorithm terminates in step 3, then a matching-cut exists. When the algorithm terminates here, a face F incident with two non-adjacent edges g and h, both marked with 1, is considered. Start with $M = \{g, h\}$. Now for every edge $e \in M$ marked with 1, it is either on the boundary of the outer face of G, or it is marked with 1 in step 5. In the second case, also add the edge f referred to in step 5 to M. Repeat this for every edge in M. Now we have constructed $M = \{e_1, \ldots, e_k\}$, such that e_1 and e_k are the only edges on the boundary of the outer face, and e_i and e_{i+1} are non-adjacent

and on the boundary of the same face F_i, for every $i = 1, \ldots, k-1$. ($g = e_i$ and $h = e_{i+1}$ for some i.) So we can draw a closed curve in the plane that crosses only these edges, and thus M is an edge cut.

Now we will argue that M is a matching. Clearly, for every i, e_i and e_{i+1} are not adjacent. Observe that for every $2 \leq i \leq k-1$, $e_i = uv$ is an internal edge, incident with the faces F_i and F_{i+1}, but both u and v are incident with the outer face. F_i and F_{i-1} are in turn incident with e_{i+1} and e_{i-1}, which are not adjacent to e_i. It follows that a curve from u to v through the outer face can be combined with edge uv such that a closed curve is obtained. This curve has edges $e_1, \ldots, e_{i-1}$ in one of its two regions, and the edges $e_{i+1}, \ldots, e_k$ in the other region (note that this does not yet rule out edges touching the curve in u or v). Edge e_{i-1} does not touch the curve, so e_{i-1} is not adjacent to any of the edges $e_{i+1}, \ldots, e_k$. This can be done for every $2 \leq i \leq k-1$, so M is a matching.

Now we will prove that if a matching-cut exists, the algorithm will terminate in step 3: if M is a matching cut in G, and $e \in M$, then we argue that at every step of the algorithm, e is unmarked or marked with 1. This is clearly true after step 1. The next time an edge $e \in M$ is marked in the algorithm (in step 5), a face F is considered. Since M is a matching-cut, this face is incident with one other edge $f \in M$ such that e and f are not adjacent. By induction, f is marked with 1 (since it is marked at this stage), so e will be marked with 1.

So whenever a face is considered that is incident with (at least) two edges from M, either a new edge of M will be marked with 1, or in step 3 it will be concluded that a matching-cut is found. Since at every step the graph is 2-connected with the internal edges unmarked, an internal face only has all incident edges marked when there is only one internal face. We conclude that the matching-cut will be found before in step 4 it is concluded that no matching-cut exists. If no matching-cut is found, the algorithm will end in step 4 when the last face is considered. □

Chapter 4

A characterization of extremal graphs without matching-cuts

4.1 Introduction

Throughout this chapter we allow graphs to have multi-edges. An edge set is called a *matching-cut* if it is both an edge cut and a matching. If a graph has no matching-cut, it is called *(matching) immune*. Motivated by a question on economic, reliable network design, Farley and Proskurowski studied immune graphs with minimum number of edges, for any given number of vertices [25]. Immune graphs represent networks that are more reliable than graphs with matching-cuts. Therefore we want to find an immune (connected) graph on a given set of vertices, such that the the number of edges (the cost of the network) is minimized. A natural question is how many edges are needed to construct an immune graph on a given number of vertices, and what is the structure of a graph that minimizes this number?

Farley and Proskurowski [26] proved the following extremal result on immune graphs.

Theorem 4.1 (Farley and Proskurowski) *If $G = (V, E)$ is immune, then*

$$|E| \geq 3(|V| - 1)/2.$$

In addition, they constructed a large class of multi-graphs which we will call *ABC graphs*, that have the following properties:

- ABC graphs are immune.
- If $G = (V, E)$ is an ABC graph, then $|E| = \lceil 3(|V| - 1)/2 \rceil$.

For every integer $n \geq 1$, an ABC graph exists. For every integer $n \geq 1$, $n \neq 2$ a simple ABC graph exists. This shows that the lower bound from Theorem 4.1 is tight. It also inspires the following definition.

Definition 4.2 *An immune graph $G = (V, E)$ is called* extremal immune *if* $|E| = \lceil 3(|V| - 1)/2 \rceil$.

Instead of 'G is an ABC graph' we will often say 'G is ABC'. Farley and Proskurowski stated the following conjecture.

Conjecture 4.3 (Farley and Proskurowski) *Every extremal immune graph is ABC.*

We present a proof of Conjecture 4.3. The rest of the chapter is organized as follows. We will start with some general definitions and results from [26] in Section 4.2. In Section 4.3 we give an overview of the proof. Definitions related to ABC graphs are stated in Section 4.4. Then in Section 4.5 the structure of ABC graphs is studied, and some properties are stated. In Section 4.6 a few types of matching-cuts that we will often use in the proof are introduced. In Section 4.7 the first part of the proof of the conjecture is given. In Section 4.8 and Section 4.9 two important cases of the proof are considered, and in Section 4.10 the proof of the conjecture is completed. In Section 4.11, a fast recognition algorithm for ABC graphs is presented.

A similar problem is studied in [49]. In this paper, Rose establishes a lower bound on the size of connected graphs of given order that cannot be disconnected by removing a stable vertex set, instead of a matching. Matching-cuts have been studied under many different names (e.g. simple cuts, stable cutsets in line graphs, disconnecting matchings). In [13] and [33], *primitive graphs* are studied, which are immune graphs that have no immune subgraphs. For results regarding the complexity of finding matching-cuts for different graph classes, see Chapter 3.

4.2 Preliminaries

4.2.1 Blocks

A connected graph is *2-connected* if it has no cut vertices and is not a K_1 or K_2. A *block* of a graph is a maximal connected subgraph without cut vertices. So a block is 2-connected unless it is a K_1 or K_2. The edge sets of all blocks of a graph partition the edge set of the graph. It is well-known that another way to characterize the edge partition given by the blocks of a graph is the following:

Observation 4.4 *The edges e and f are part of the same block of G if and only if there is a cycle containing both e and f.*

In this chapter observations denote statements that are well-known or easy to prove, and will be stated without proof.

Observation 4.5 *If $M \subseteq E(H)$ is an edge cut for a graph H, and H is a block of G, then M is an edge cut for G.*

Definition 4.6 *Let H be a subgraph of G, and $v \in V(H)$. If $d_H(v) = d_G(v)$ then v is called an* internal vertex *of H, otherwise v is called a* connection vertex *of H. H is called an i-*connection subgraph *of G if H has at most i connection vertices.*

This definition will be used often for induced triangles and 4-cycles. Note that if G is immune and an induced C_4 is a 2-connection subgraph of G with connection vertices u and v, then u and v cannot be neighbors. In this case the C_4 is called a 2-connection 4-cycle *between* u and v.

Observation 4.7 *If $G \neq K_1, K_2$ is connected but not 2-connected, then G has at least two 1-connection blocks.*

4.2.2 Contraction and expansion operations

In this chapter we make extensive use of contraction operations and their inverse, edge expansion operations. Recall that these are defined for edge-labeled graphs (Chapter 1) that consist of a vertex set, edge set, and incidence function on the edges. Therefore if a graph G' can be obtained from G by a series of edge contractions and edge deletions, then all edges of G' correspond to edges of G, even though they may have different end vertices in both graphs (the incidence function has changed). We will often consider one particular edge set M and study its properties both in G' and in G. For instance, M can be a matching in G but not in G'.

However, to improve the readability of our proofs, we use the usual notations throughout: we denote edges by their end vertices, and graphs with just their vertex and edge sets and omit the incidence function. It is important to keep in mind that if for instance G' is obtained from G by the contraction of u_1u_2 into u, edges $u_1v \in E(G)$ and $uv \in E(G')$ are considered to be the same edge.

Suppose G' can be obtained from G by a series of edge contractions and edge deletions. So G can be constructed from G' by a series of edge expansions and edge additions. In this case the following definition is useful:

Definition 4.8 *Suppose G can be obtained from G' be a series of edge expansions and edge additions. Then we say that an edge set $M \subseteq E(G')$ is* split *if $G'[M]$ and $G[M]$ are not isomorphic. In this case, the subgraph $G'[M]$ is also said to be split. Similarly, we say that two edges $e, f \in E(G')$ are split if $G'[\{e, f\}]$ and $G[\{e, f\}]$ are not isomorphic.*

In Figure 4.1, an example is shown: G' is obtained from G by an edge expansion of u into u_1u_2; $G[M_1]$ is split but $G[M_2]$ is not.

The following two observations are essential for our matching-cut constructions.

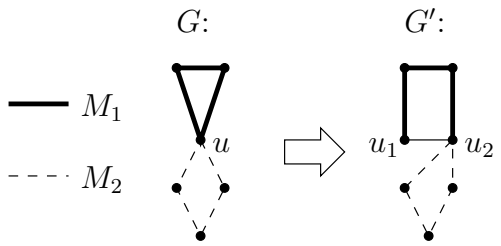


Figure 4.1: An example of splitting

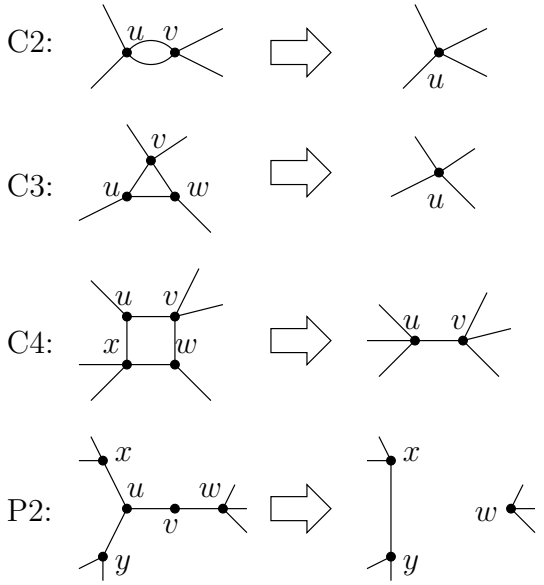


Figure 4.2: The four reduction operations

Observation 4.9 *Let G be obtained from G' by a series of edge expansions and edge additions. If M is a matching in G', then M is a matching in G. If $e, f \in M$ share one end vertex v in G', no other edge pairs in M are adjacent in G', and e and f are split, then M is a matching in G.*

Observation 4.10 *Let G be obtained from G' by a series of edge expansions, and additions of edges parallel to existing edges. If M is an edge cut in G', and no edges parallel to edges in M are added, then M is an edge cut in G.*

For their proof of Theorem 4.1, Farley and Proskurowski [26] introduced four graph operations, named after the structure they reduce. The four operations are illustrated in Figure 4.2. Below are formal definitions, which show that all of these operations can be expressed by edge deletions and contractions.

C2 Let the vertices u and v induce a C_2 in the graph G. The C2 operation

consists of deleting one of the edges of this C_2 and contracting the other.

C3 Let the vertices u, v and w induce a C_3 in G. The C3 operation consists of deleting uv and contracting vw and wu.

C4 Let subgraph C of G be a 4-cycle with edge set $\{uv, vw, wx, ux\}$. The C4 operation consists of deleting uv and contracting ux and vw. Note that for one C_4 subgraph the C4 operation can have two different results.

Originally this operation was only defined for *induced* 4-cycles, but we remark that Lemma 4.11 below also holds when C is not induced. In this case, the resulting graph will have a multi-edge between the two resulting vertices.

P2 Let the vertices u and v be neighbors in G with $d(u) = 3$ and $d(v) = 2$. Let v have another neighbor $w \neq u$, and let u have another neighbor $x \neq v$. The P2 operation consists of deleting uv and contracting ux and vw.

It is clear that these operations consist of series of edge deletions and contractions, so Definition 4.8 of split subgraphs can be used for these operations. For instance we will say 'G' is obtained from G with a C4 operation, such that the subgraph H of G' is split (when reconstructing G from G')'.

It can be checked that these four operations have the following properties.

Lemma 4.11 (Farley and Proskurowski) *Suppose G' can be obtained from G by a C2, C3, C4 or P2 operation. Then the following statements hold:*

- *If G is immune, then G' is immune.*
- *If $|E(G)| = \lceil 3(|V(G)| - 1)/2 \rceil$, then $|E(G')| \leq \lceil 3(|V(G')| - 1)/2 \rceil$. This inequality is always an equality for the C3, C4 and P2 operation.*

To prove Theorem 4.1, the following lemma was used.

Lemma 4.12 (Farley and Proskurowski) *If $G \neq K_1$ is an extremal immune graph, then one of the operations C2, C3, C4 or P2 can be applied to G.*

Our proof of Conjecture 4.3 is based on this lemma.

4.3 An overview of the proof

The proof of Conjecture 4.3 is by contradiction, so first we assume an extremal immune graph exists that is not an ABC graph. Then we consider a graph G with minimum number of vertices among all such graphs. This is called a *minimum counterexample*. We first consider the case that a minimum counterexample G contains a C_2, so a C2 operation can be applied, resulting in vertex v. After applying this C2 operation, another extremal immune graph G' is obtained (Lemma 4.11, Theorem 4.1). By our choice of G, G' must be an

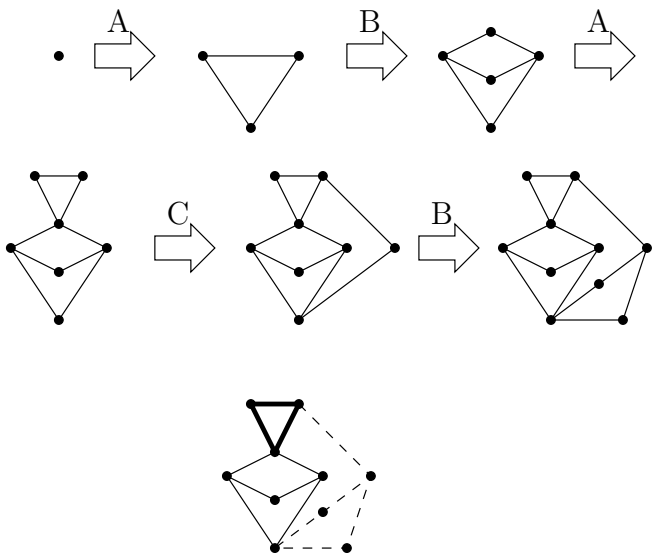


Figure 4.3: The construction of an ABC graph and corresponding edge partition

ABC graph. We consider a number of cases for G', for the choice of $v \in V(G')$ and for the possible graphs G that can correspond to this, and in every case we obtain one of the following contradictions: G has a matching-cut, G is also an ABC graph, or a smaller counterexample exists.

If a minimum counterexample G contains a triangle, we can apply operation C3 and find a contradiction in a similar way. If G contains a C_4, applying a C4 operation leads to a contradiction. Finally, we can show that if a P2 operation can be applied resulting in ABC graph G', then there always is a triangle or C_4 in G' that corresponds to a triangle or C_4 in G, so the previous cases can be applied. Since every operation leads to a contradiction, Lemma 4.12 shows that no counterexamples for the conjecture can exist.

Before we can state the proof, we need to study the structure of ABC graphs, which is done in the following three sections.

4.4 ABC Graphs: preliminaries

4.4.1 Definition and basic properties

In [26] a set of graphs is defined which we will call ABC graphs. ABC graphs are named after the three graph operations that can be used to construct them, which are defined below. See Figure 4.3 for an example of these operations.

Definition 4.13 *An* A operation *on a vertex u introduces two new vertices v and w and the edges uv, uw and vw. $G' = A(G, u, v, w)$ is used to denote that G' is obtained by an A operation on a vertex u of G, introducing v and w.*

Definition 4.14 *A* B operation *on the edge uv introduces two new vertices w and x and the edges uw, vw, ux and vx, and removes the edge uv. $G' = B(G, uv, w, x)$ is used to denote that G' is obtained by a B operation on the edge uv of G, introducing w and x.*

Definition 4.15 *A* C operation *on the vertices u and v ($u = v$ is allowed) introduces a new vertex w and the edges uw and vw. $G' = C(G, u, v, w)$ is used to denote that G' is obtained by a C operation on the vertices u and v of G, introducing w.*

We say that a vertex v is *used in operation x* if x is an A or C operation on v, or a B operation on an edge incident with v. Note that the C operation is the only operation that can introduce parallel edges.

Definition 4.16 *An* AB graph *is a graph that can be obtained from a K_1 by a sequence of A and B operations. An* ABC graph *is a graph that can be obtained from a K_1 by a sequence of A and B operations and* at most one *C operation.*

If G is an AB(C) graph, a sequence of operations that constructs G is called a *decomposition* of G. Formally, a decomposition is a list of the form $G_0 = (\{u\}, \emptyset)$, $G_1 = A(G_0, u, v, w)$, $G_2 = B(G_1, vw, x, y)$ etc. (In this example, G_2 is a $K_{2,3}$.) $G_0, \ldots, G_{i-1}$ are called *intermediate graphs* in this particular decomposition of G_i. In general, an ABC graph can have different decompositions, even decompositions where the intermediate graphs are not isomorphic. In the top part of Figure 4.3 an example of a decomposition of an ABC graph is shown.

Observation 4.17 *Suppose that in a decomposition of the ABC graph G, operation x and operation y are applied consecutively, x first. Now unless operation x introduces vertices that are used in operation y, the order in which x and y are applied can be reversed, giving another decomposition of G.*

The above observation will be used implicitly in a lot of proofs, just like the next observations.

Observation 4.18 *In a decomposition of the ABC graph G that does not start with vertex v, $d(v) = 2$ if and only if v is not used in any operation in the decomposition.*

For applying Observation 4.18 it is useful to note that for every vertex v in an ABC graph $G \neq K_1$, G has a decomposition that does not start with v.

Observation 4.19 *AB graphs are simple. In an ABC graph, only between one pair of vertices u and v parallel edges can exist. In this case, in every decomposition, the C operation is on u introducing v or on v introducing u, and no B operations are applied to these edges between u and v. There are at most two parallel edges between u and v.*

It can be checked that for ABC graphs the following two properties hold [26].

Theorem 4.20 (Farley and Proskurowski) *If G is an AB graph on n vertices with m edges, then n is odd, $m = 3(n-1)/2$ and G is immune.*

Theorem 4.21 (Farley and Proskurowski) *If G is an ABC graph on n vertices with m edges but G is not an AB graph, then n is even, $m = (3n-2)/2$ and G is immune.*

So ABC graphs are extremal immune graphs.

Observation 4.22 *Let $f : n \to \lceil 3(n-1)/2 \rceil$. $f(n+1) = f(n)+1$ if n is even, and $f(n+1) = f(n)+2$ if n is odd.*

4.4.2 Partitions of ABC graphs into H-components

In the next definition G represents a (vertex) labeled graph, but H is an unlabeled graph.

Definition 4.23 *A graph G that can be obtained from a graph H by assigning vertex labels and applying B operations is called an* H-component.

For an H-component G, the sequence of B operations that constructs G from a labeled graph isomorphic to H is called a *decomposition of G from H* (or *starting with H*). Note that G does not have to be an ABC graph, but it is an ABC graph if H is an ABC graph.

The following graphs will often be used for H in the context of H-components:

K_2: a K_2-component is also called an *edge component*. If we consider a decomposition starting with a K_2 on vertices u and v, then this is called an edge component *between u and v*.

K_3: a K_3-component is also called a *triangle component*. If we consider a decomposition starting with a specific labeled copy of K_3, then the vertices of this K_3 are called the *triangle vertices* of this triangle component.

P_3: the end vertices of the P_3 are called the *end vertices* of the P_3-component.

C_2: both C_2 and P_3-components are associated with the C operation in a decomposition.

Next we prove the useful property that we can partition the edges of ABC graphs into edge induced subgraphs that are all H-components for $H = K_3$, C_2 or P_3.

Claim 4.24 *For every decomposition of an ABC graph G, we can partition the edges of G into sets $A_1, \ldots, A_k$ and at most one set C such that for every i, $G[A_i]$ is a triangle component, and $G[C]$ is a C_2 or P_3-component.*

Proof: By induction on the number of operations. Observe that if G can be made from the ABC graph G' by an A or C operation, and the edge set M induces a graph H in G', then M induces the same graph H in G. The edges

introduced by an A operation induce a K_3 (a triangle component), and the edges introduced by a C operation induce a P_3 or a C_2 (a P_3 or C_2-component). If G can be made from the ABC graph G' by a B operation, the statement is trivial. □

Triangle components that correspond to a decomposition of G in this way are called *A-components*, and the C_2 or P_3-component that corresponds to the C operation is called the *C-component*. See the bottom part of Figure 4.3 for an example of a partition of edges into two A-components and one C-component, corresponding to the given decomposition. For this graph it can be checked that any decomposition will give the same partition into A and C-components, but in general such a partition depends on the chosen decomposition.

Observation 4.25 *For any decomposition of an H-component G from a labeled copy of H, we can partition $E(G)$ into sets $\{E_{uv} : uv \in E(H)\}$ such that $G[E_{uv}]$ is an edge component between u and v. u and v are the only vertices of $G[E_{uv}]$ that can be connection vertices in G.*

The edge components $G[E_{uv}]$ from this observation will be denoted as $F(uv)$. For instance, for a triangle component with triangle vertices a, b and c (so a particular decomposition is chosen), we will often consider the subgraphs $F(ab)$, $F(ac)$ and $F(bc)$. $F(ab)$ is a 2-connection edge component with connection vertices a and b. Similarly, C_2 and P_3-components will be partitioned into two edge components.

Now that the main terminology and notations are defined, we will proceed by stating a large number of properties of ABC graphs, which we need in order to prove Conjecture 4.3 in Sections 4.7-4.10.

4.5 The structure of ABC graphs

4.5.1 Blocks in ABC graphs

We characterize the block structure of ABC graphs. In Figure 4.4, this block structure is illustrated by three graphs from a decomposition. G_1 is an AB graph, in which the blocks are exactly the A-components (this holds for every decomposition). All blocks of G_1 have odd order. Then a C operation is applied on the vertices x and y. In G_2, the two new edges become part of an even order new block, that contains exactly those blocks from G_1 that contain edges from an (x, y)-path P (the choice of P does not matter). Then further A and B operations are applied to obtain G_3: A operations introduce new blocks and B operations change blocks, but the essence of the block structure is not changed by these operations. The following lemma describes the block structure of ABC graphs formally.

Lemma 4.26 *An ABC graph G is connected and consists of odd order blocks and at most one even order block. For every decomposition of G, the odd order*

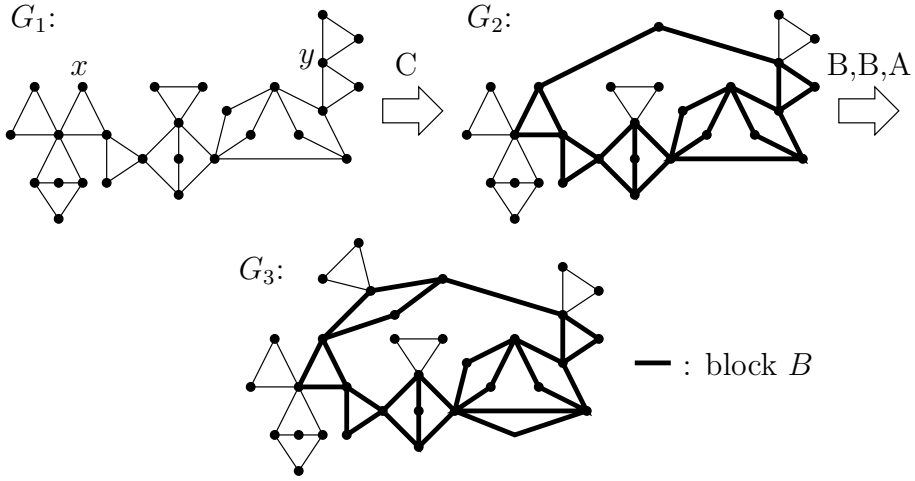


Figure 4.4: The block structure of ABC graphs

blocks are A-components, and the even order block B has the following structure: it contains the C-component C, and either $B = C$ and C is a C_2-component, or C is a P_3-component with end vertices x and y such that

1. *There are $k \geq 1$ A-components $T_1, \ldots, T_k$ such that $E(B) = E(C) \cup E(T_1) \cup \ldots \cup E(T_k)$.*
2. *There are $k-1$ vertices $v_1, \ldots, v_{k-1}$ such that $V(T_i) \cap V(T_{i+1}) = \{v_i\}$, and no other A-components in B share vertices.*
3. *If $k = 1$ then $V(T_1) \cap V(C) = \{x, y\}$. If $k \geq 2$ then $V(T_1) \cap V(C) = \{x\}$ and $V(T_k) \cap V(C) = \{y\}$. No other A-components in B share vertices with C.*

Proof: Let G be an ABC graph. G is obviously connected. The proof is by induction on the number of operations in a decomposition of G. Consider a decomposition of G.

Suppose G is obtained by a B operation from an ABC graph G'. Observe that B operations do not change the block structure. This means that if $E' \subseteq E(G)$ induces a block and $e \in E'$, then a B operation on e will result in four edges that together with $E' - e$ form a block in the new graph, and all other blocks remain the same. Also all other properties in the lemma are maintained when B operations are applied: by definition an H-component is still an H-component after applying a B operation on one of its edges, all other H-components remain the same, the parity of the order of blocks does not change, and the vertices that two H-components have in common are unchanged by B operations.

Suppose G is obtained by an A operation from an ABC graph G'. Every A operation introduces a new block, consisting of the three new edges, and does not change the block structure of the rest of the graph. Again all other

properties in the lemma are maintained by A operations. This already proves the statement for AB graphs.

Now we only have to prove the lemma for the case $G = C(G', x, y, z)$. For G' we know that the blocks correspond to A-components. Let $C = G[\{xz, yz\}]$. If $x = y$, then vertex x is a cut vertex in G, and the two new edges induce a new block in G. All other blocks remain the same. It is easy to check that the lemma holds.

If $x \neq y$, then consider a path P from x to y in G'. P contains edges from A-components $T_1, \ldots, T_k$, numbered along P. Note that P does not visit the same A-component twice, since in that case a cycle can be constructed in G' containing edges of multiple blocks (A-components) of G', which is a contradiction.

The edges of P together with xz and yz form a cycle in G, so these edges are part of the same block B in G. Existing cycles in G' are not changed by the C operation, so $E(T_1) \cup \ldots \cup E(T_k) \cup \{xz, yz\} \subseteq E(B)$ (Observation 4.4). Suppose B contains more edges, so there is a cycle K in G containing at least one edge from $E(T_1) \cup \ldots \cup E(T_k) \cup \{xz, yz\}$ and at least one edge from another A-component of G'. K then contains a path Q with two end vertices in $V(T_1) \cup \ldots \cup V(T_k)$, but no internal vertices in this set. Using P, Q can be extended to a cycle that does not contain xz or yz, but contains edges from different A-components, which is a contradiction with the block structure from G'. We conclude that the block containing xz and yz is induced by $E(T_1) \cup \ldots \cup E(T_k) \cup E(C)$.

T_i and T_{i+1} clearly share a vertex ($1 \leq i \leq k-1$). Since they are blocks in G', they share at most one vertex. We call this vertex v_i. If T_i and T_j with $j > i+1$ share a vertex, then P can be used to construct a cycle through $T_i, T_{i+1}, \ldots, T_j$ in G', a contradiction.

If T_i with $1 < i < k$ contains x or y, then we can use P to construct a cycle through multiple A-components in G', a contradiction.

Finally, it can be verified that the number of vertices in the new block B is even, and that the C operation does not change parity of other blocks, or the fact that the other blocks are A-components. This concludes the proof. □

Note that there is an even order block in an ABC graph G if and only if G has even order.

We remark that a C_2-component that is not equal to a C_2 can be viewed as a combination of a triangle component and a P_3-component (consider the first B operation), so in this case the characterization of the even order block in Lemma 4.26 can be applied in two ways (see also Claim 4.34). But other than this case, the number of A-components is always the same in any decomposition of an ABC graph.

We state some immediate corollaries of this lemma (without proofs). These will be used later, while referring to the above lemma:

- If an AB graph is 2-connected, it contains exactly one A-component.
- If the C operation is applied to two distinct vertices x and y of the same A-component, then the even order block has only one A-component.

- If the C operation is applied to x and y, then x is incident with only one A-component that is part of the even order block, and the same holds for y.
- If G is a 2-connected ABC graph such that the C operation is applied on x and y, then x and y are the only connection vertices of the C-component.

Corollary 4.27 *Let G be an even order ABC graph with a decomposition such that the C operation is applied on x and y. If M_1 is an edge cut for the C-component that separates x from y, and M_2 is an edge cut for an A-component T, then either M_2 or $M_2 \cup M_1$ is an edge cut for G.*

Proof: We use the notation from Lemma 4.26, but use the notation v_0 and v_k for x resp. y. If T is a block, the result follows directly from Observation 4.5. Otherwise, T is part of the even order block B, so $T = T_i$ for some i. If M_2 is not an edge cut for G, then $M_2 = [S_2, V(T_i) \backslash S_2]$ for some $S_2 \subset V(T_i)$, with $v_{i-1} \in S_2$, $v_i \notin S_2$ (Lemma 4.26). We know that $M_1 = [S_1, V(C) \backslash S_1]$ for some $S_1 \subset V(C)$, with $v_0 \in S_1$, $v_k \notin S_1$. Consider $S = S_1 \cup V(T_1) \cup \ldots \cup V(T_{i-1}) \cup S_2$. By Lemma 4.26, $[S, V(B) \backslash S] = M_1 \cup M_2$, and therefore $M_1 \cup M_2$ is an edge cut for G, since B is a block. □

The following corollary follows directly from Lemma 4.26 in combination with Observation 4.7.

Corollary 4.28 *Let $G \neq K_1$ be an ABC graph that is not 2-connected. Then G contains a 1-connection A-component. If G is an AB graph, then it contains at least two 1-connection A-components.*

We will also use the following observation about 2-connected ABC graphs.

Observation 4.29 *If G is a 2-connected ABC graph, then in any decomposition no A operation is applied after the C operation.*

4.5.2 Decompositions of triangle components

In this section we show that triangle components have many different decompositions. In particular, if edge uv is an edge in triangle component T, then T has a decomposition that starts with a triangle that contains edge uv. This is shown in Corollary 4.32. A similar statement appears in Claim 4.33.

Claim 4.30 *If an ABC graph G contains a 2-connection 4-cycle C, then a C4 operation on C yields another ABC graph G'. Moreover, if G is a triangle component, then G' is a triangle component.*

Proof: Consider a 2-connection 4-cycle C. As the main step in the proof, we first determine a decomposition of G such that all edges of C are introduced by the same B operation.

Consider a decomposition of G. If G is a triangle component, this decomposition starts with G_0 which is a triangle. Let G_i be the first intermediate graph

that contains all edges of C (note that $i > 0$). Let operation a be the operation that is used to obtain G_i from G_{i-1}. So some of the edges of C are introduced by operation a.

Clearly, operation a is not an A operation.

Now suppose operation a is a B operation, say $G_i = B(G_{i-1}, uv, w, x)$. If $E(C) = \{uw, vw, ux, vx\}$, then the desired decomposition is found. Otherwise, C contains exactly two edges introduced by this operation, and w.l.o.g. $E(C) = \{uw, vw, uy, vy\}$ for some vertex y. Since $N_{G_i}(x) = N_{G_i}(y) = \{u, v\}$, we can switch the labels x and y throughout the decomposition of G_i, to get a decomposition of G_i where all edges of C are introduced by the B operation. Then continue with the rest of the decomposition (without changing any labels). Apart from possibly the vertex labels, G_0 is still the same in the new decomposition.

If operation a is a C operation, then w.l.o.g. it is a C operation on u and v that introduces w, and $E(C) = \{uw, vw, ux, vx\}$. If x is introduced by a B operation that also introduces y, then this must be a B operation on uv since $d(x) = 2$ (Observation 4.18). Now instead let this B operation introduce w and x and immediately apply a C operation on u and v introducing y. Now proceed with the rest of the decomposition, which gives the desired decomposition. If x is introduced by an A operation, then w.l.o.g. this must be an A operation on u introducing x and v. Instead apply a C operation only on u introducing v, and immediately apply a B operation on one of the edges between u and v, introducing w and x. Observe that we always can find a decomposition where x is not the starting vertex, so this covers all cases for x.

In all cases we have determined a decomposition of G such that all edges of C are introduced by a B operation that introduces w and x. Since $d(w) = d(x) = 2$, by Observation 4.17 and Observation 4.18 we can assume that this is the last operation applied. So the graph G' obtained by a C4 operation on C is one of the intermediate graphs in this decomposition, and therefore is an ABC graph. In addition, if all operations in the original decomposition are B operations, then the initial graph G_0 is the same in our new decomposition, so if G is a triangle component, then G' is a triangle component. □

Claim 4.31 *If T is a triangle component with at least 5 vertices, then for every edge $e \in E(T)$, T contains a 2-connection 4-cycle that does not contain e.*

Proof: Observe that every triangle component on 5 vertices is a $K_{2,3}$, and for this graph the property holds. Now consider a triangle component T with at least 7 vertices, and a decomposition of T. For an edge e that is not one of the edges introduced by the last B operation, the statement is obvious. Otherwise, consider the triangle component T' from which T was constructed by a B operation on an edge e' (so T' does not contain e). By induction, T' contains a 2-connection 4-cycle C that does not contain e'. After the B operation on e', C still is a 2-connection 4-cycle in T, which proves the statement for T. Since triangle components have odd order, this proves the claim by induction. □

By combining the previous two claims, we obtain a useful corollary.

Corollary 4.32 *If u is a vertex in a triangle component T, then a decomposition of T exists that starts with the single vertex u. If uv is an edge in a triangle component T, then a decomposition of T exists where u and v are triangle vertices.*

Proof: We first prove the statement for the edge uv by induction. For $T = K_3$, the statement is clearly true. Otherwise, T contains a 2-connection 4-cycle K that does not contain edge uv (Claim 4.31). A C4 operation on K gives another triangle component T' (Claim 4.30), from which T can be constructed (assuming proper vertex labeling in T'). By induction, T' has a decomposition where u and v are triangle vertices.

The statement for the single vertex u follows immediately. □

Claim 4.33 *For a triangle component T and any two vertices $u, v \in V(T)$, a decomposition exists with triangle vertices u, a and b (and edge component $F(ab)$) such that $v \in V(F(ab))$ ($v = a$ or $v = b$ is possible).*

Proof: If $T = K_3$, then the statement is obvious. Now suppose $T \neq K_3$.

If a 2-connection 4-cycle K exists such that a C4 operation on K does not remove u or v, then apply this C4 operation. This gives another triangle component (Claim 4.30) with vertices u and v, and the statement follows by induction.

If no such 2-connection 4-cycle exists, then either u or v, say u, is a degree two vertex on a 2-connection 4-cycle. Choose an edge ua incident with u. By Corollary 4.32, a decomposition exists such that u, a and another vertex b are triangle vertices. Since $d(u) = 2$, both edge component $F(ua)$ and $F(ub)$ are single edges, so $v \in V(F(ab))$. □

4.5.3 Decompositions of ABC graphs

Claim 4.34 *If an ABC graph G is simple, then a decomposition of G exists such that every intermediate graph is simple.*

Proof: Consider a decomposition of a simple ABC graph G. The only way that parallel edges can be introduced is with a C operation on $x = y$, introducing z (Observation 4.19). Since G is simple, one of the edges between x and z must be used in a B operation, that introduces v and w. Now instead of the C operation, use an A operation on x that introduces v and z. Instead of the B operation, use a C operation on x and z that introduces w. This gives the desired decomposition of G. □

Claim 4.34 implies that for every ABC graph G there is a decomposition where the C-component is a P_3-component, unless the C-component consists of two parallel edges.

Claim 4.35 *If an ABC graph G contains a 1-connection A-component T with connection vertex u, then G has a decomposition in which $G[E(G) \setminus E(T)]$ is an intermediate graph.*

Proof: Consider a decomposition of G and a 1-connection A-component T with connection vertex u.

If T is introduced by an A operation on a vertex other than u, this is an isolated vertex, so this A operation is the first operation in the decomposition. Therefore w.l.o.g. we can consider a decomposition in which T is introduced by an A operation on u.

Since u is the only connection vertex of T in G, there is a decomposition of G that ends with this A operation and a number of B operations on this A-component (Observation 4.17). In this decomposition, $G[E(G)\backslash E(T)]$ is an intermediate graph. □

Claim 4.36 *If T is an A-component of an AB graph G, then a decomposition of G exists in which T is an intermediate graph or $T = G$.*

Proof: We use induction on the number of A-components. If G has only one A-component, then the statement is trivial. Otherwise, G has a cut vertex (Lemma 4.26), and therefore at least two 1-connection A-components (Corollary 4.28). So there is a 1-connection A-component $T' = G[E']$ that is not equal to T. By Claim 4.35, G can be constructed from the AB graph $G' = G[E(G)\backslash E(T')]$. G' has fewer A-components than G (in any decomposition). By induction G' has a decomposition in which T is an intermediate graph, which proves the claim. □

Note that a similar statement is not true for ABC graphs: consider the ABC graph from Figure 4.3. Apply an A operation on the vertex introduced by the C operation. It can be checked that there is no decomposition of the resulting ABC graph in which the A-component introduced by the last A operation is an intermediate graph.

From Claim 4.36 and Corollary 4.32, the following corollary follows immediately.

Corollary 4.37 *For an AB graph G and $v \in V(G)$, a decomposition exists that starts with the single vertex v.*

Claim 4.38 *If an ABC graph G contains a C-component P between x and y such that x and y are the only connection vertices of P in G, then G has a decomposition in which $G[E(G)\backslash E(P)]$ is an intermediate graph.*

Proof: In the decomposition we consider, the C-component P is introduced by a C operation on vertices x and y. Since x and y are the only connection vertices of P in G, there is a decomposition of G that ends with this C operation and a number of B operations on this C-component (Observation 4.17). In this decomposition, $G[E(G)\backslash E(P)]$ is an intermediate graph. □

Claim 4.39 *If an ABC graph G contains a triangle T, then a decomposition exists in which T is an A-component.*

Proof: We consider a decomposition of G such that G is obtained from ABC graph G' with a single operation a, such that G' is simple if G is simple (Claim 4.34). Clearly, for every triangle in G that is also a triangle in G' the statement is true by induction. So if suffices to consider triangles in G that use edges introduced by operation a.

If operation a is an A operation, then for the new triangle the statement is true. (Note that no new edges combine with old edges to form a triangle.)

If $G = B(G', uv, w, x)$, and there is a triangle in G that contains edges from $C = G[u, v, w, x]$, then w.l.o.g. this triangle is equal to $G[u, v, w]$. This means that in G', two parallel edges exist between u and v, but G is simple (by Observation 4.19 there are at most two parallel edges between u and v and no other parallel edges exist in G'), a contradiction with our choice of the decomposition.

Finally consider the case that $G = C(G', x, y, z)$, and there is a triangle K in G that contains edge xz or yz. Then $K = G[x, y, z]$, and $xy \in E(G')$. xy is part of an A-component T of G' (since G' is an AB graph). By Claim 4.36, there is a decomposition of G' that starts with the construction of T. By Corollary 4.32, there is a decomposition of T that has x and y as triangle vertices. Let z' be the third triangle vertex in this decomposition. So a decomposition of G' exists that starts with a triangle with vertices x, y and z'. Now instead start with a triangle with vertices x, y and z, and apply a C operation on x and y introducing z'. Proceed with the rest of the decomposition of G'. This is a decomposition of G such that the triangle K is introduced by an A operation. □

4.6 Edge components and matching-cuts

In this section, we show various ways to find matching-cuts for edge components, and for graphs deduced from edge components, triangle components and P_3-components by an expansion operation.

Throughout this section, we use G_{uv} to denote an edge component with which we associate a decomposition from a copy of K_2 with vertex labels u and v. In the following proofs we use the fact that every edge component G_{uv} not equal to a single edge has at least one 2-connection 4-cycle C such that a C4 operation on C yields again an edge component between u and v. In a decomposition of G_{uv}, the edges introduced by the last B operation correspond to such a 2-connection 4-cycle. A C4 operation on this 2-connection 4-cycle allows us to use induction.

Claim 4.40 *For every edge component G_{uv} and every edge $e \in E(G_{uv})$, there is a matching-cut M that separates u from v with $e \in M$.*

Proof: If $E(G_{uv}) = \{uv\}$ then the statement is true. Otherwise, consider a 2-connection 4-cycle C between x and y. A C4 operation on C gives another edge component G'_{uv}. If $e \in E(G'_{uv})$, then consider a matching-cut M for G'_{uv} that contains e and separates u from v (induction). M can be turned into a matching-cut in G_{uv} with the desired properties, also if $xy \in M$. If $e \notin E(G'_{uv})$,

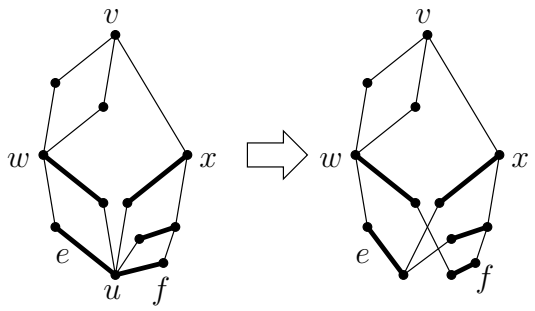


Figure 4.5: A matching-cut in a split edge component that is not incident with v

then consider a matching-cut M for G'_{uv} that separates u from v with $xy \in M$ (induction). M can be turned into a matching-cut M' in G_{uv} that separates u from v with $e \in M'$. □

Claim 4.41 *For every edge component G_{uv} and every vertex $w \in V(G_{uv})\backslash\{v\}$, there is a matching-cut M that separates $\{u, w\}$ from $\{v\}$.*

Proof: If $E(G_{uv}) = \{uv\}$ then the statement is true. Otherwise, a 2-connection 4-cycle C between x and y exists such that a C4 operation on C yields another edge component G'_{uv}. If $w \in V(G'_{uv})$, then start with a matching-cut M for G'_{uv} that separates $\{v\}$ from $\{u, w\}$ (induction). M can be turned into a matching-cut for G_{uv} with the desired properties, also if $xy \in M$. If $w \notin V(G'_{uv})$, consider a matching-cut M for G'_{uv} that separates u from v with $xy \in M$ (Claim 4.40). This can be made into a matching-cut in G_{uv} with the desired properties. □

The proof of the following claim is illustrated in Figure 4.5. In the claim we consider the same edge set M in two different graphs, that can be obtained from each other by contractions resp. edge expansions (see Section 4.2.2).

Claim 4.42 *If a graph G' can be made from an edge component G_{uv} by a non-trivial edge expansion of u, then there is an edge cut M in G_{uv} that separates u from v, is not incident with v, and is a matching-cut in G'.*

Proof: Since G' is obtained by a non-trivial edge expansion, in G_{uv} we have $d(u) \geq 2$. So in a decomposition of G_{uv} at least one B operation is applied. Let the first B operation introduce two vertices w and x. So the edges of G_{uv} can be partitioned into 2-connection edge components $F(uw)$, $F(ux)$, $F(vw)$ and $F(vx)$. Since the edge expansion is non-trivial, we can find $e \in E(F(uw))$ and $f \in E(F(ux))$ such that e and f are split by the edge expansion. Let $M_1 = [S_1, T_1]$ be a matching-cut for $F(uw)$ with $u \in S_1$, $w \in T_1$ and $e \in M_1$ (Claim 4.40). Let $M_2 = [S_2, T_2]$ be a matching-cut for $F(ux)$ with $u \in S_2$, $x \in T_2$ and $f \in M_2$. The only adjacent edges in $M_1 \cup M_2$ are e and f, so $M_1 \cup M_2$ becomes a matching in G' (Observation 4.9). Considering the vertices

that the four edge components have in common, we see that $M_1 \cup M_2 = [S_1 \cup S_2, T_1 \cup T_2 \cup V(F(vw) \cup V(F(vx))]$ is an edge cut in G_{uv} and therefore also an edge cut in G' (Observation 4.10). □

Claim 4.43 *If the graph G' can be made from the edge component G_{uv} by a non-trivial edge expansion of u, then for any two vertices $w, x \in V(G_{uv})\backslash\{u, v\}$ an edge cut for G_{uv} exists that separates u from v, does not separate w from x and is a matching-cut in G'.*

Proof: Consider a decomposition of G_{uv}. If the first B operation in this decomposition introduces both w and x, then we can construct the same edge cut as in the proof of the previous claim (see Figure 4.5). This edge cut separates $\{u\}$ from $\{v, w, x\}$.

Otherwise, we can actually construct a matching-cut in G_{uv} (instead of in G') that separates u and v but does not separate w and x. The proof is again by induction. Let C be the 2-connection 4-cycle in G_{uv} that corresponds to the last B operation in the decomposition. A C4 operation on C yields an edge e in an edge component G'_{uv}. Since the first B operation does not introduce both w and x, we know that G_{uv} is not equal to a 4-cycle and therefore G'_{uv} does not consist of a single edge. W.l.o.g. we can consider three cases:

1. The C4 operation removes w, but not x. Consider a matching-cut $M = [S, \overline{S}]$ for G'_{uv} that separates u from v and includes e (Claim 4.40). M can be made into a matching-cut $[S', \overline{S'}]$ of G_{uv} by replacing e by two edges from C. This can be done such that $w \in S'$ or such that $w \in \overline{S'}$, so we can construct the desired matching-cut regardless of x.

2. The C4 operation removes both w and x. In G'_{uv}, let f be an edge that is adjacent to the edge e. Consider a matching-cut M for G'_{uv} such that $f \in M$ (Claim 4.40). Now $e \notin M$, so M is also a matching-cut for G_{uv}, and has the desired properties.

3. The C4 operation neither removes w nor x. Recall that in the decomposition of G'_{uv} we consider, w and x are not introduced by the first B operation. So by induction, G'_{uv} has a matching-cut M that separates u from v but does not separate w from x. M is easily turned into a matching-cut in G_{uv} with the same properties.

Note that if G'_{uv} is a C_4, then case 1 or 2 applies since we assume that w and x are not introduced by the first B operation. This proves the induction base.

All matching-cuts constructed above are also matching-cuts in G' (Observation 4.9, Observation 4.10). □

The following two lemmas are useful for determining matching-cuts in graphs made from ABC graphs by an expansion operation. Lemma 4.44 is illustrated in Figure 4.6.

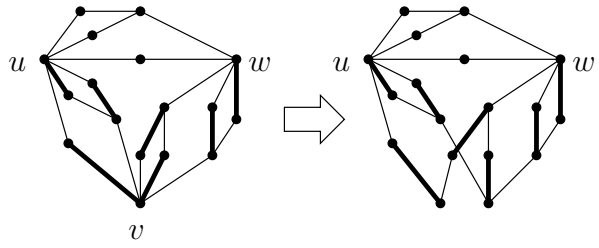


Figure 4.6: A matching-cut in a split triangle component

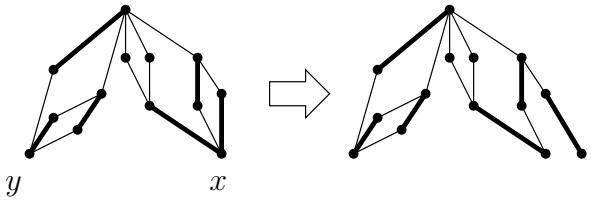


Figure 4.7: A matching-cut in a split P_3-component

Lemma 4.44 *Let T' be a graph that can be made from a triangle component T with triangle vertices u, v and w by a non-trivial edge expansion of v. There is an edge cut M in T that is not incident with any vertex in $V(F(uw))\backslash\{u, w\}$ and that is a matching-cut in T'.*

Proof: The edges of T can be partitioned into 2-connection edge components $F(uv)$, $F(uw)$ and $F(vw)$ (Observation 4.25). Since the edge expansion is non-trivial, we can find $e \in E(F(uv))$ and $f \in E(F(vw))$ such that e and f are split by the expansion. Let $M_1 = [S_1, T_1]$ be a matching-cut for $F(uv)$ with $v \in S_1$, $u \in T_1$ and $e \in M_1$ (Claim 4.40). Similarly, let $M_2 = [S_2, T_2]$ be a matching-cut for $F(vw)$ with $v \in S_2$, $w \in T_2$ and $f \in M_2$. The only adjacent edges in $M_1 \cup M_2$ are e and f, so the edges of $M_1 \cup M_2$ form a matching in T'. It can be checked that $M_1 \cup M_2 = [S_1 \cup S_2, T_1 \cup T_2 \cup V(F(uw))]$ is an edge cut in T and therefore also an edge cut in T'. Since M_1 and M_2 contain only edges from $F(uv)$ and $F(vw)$, for every $a \in V(F(uw))\backslash\{u, w\}$, a is not incident with edges from $M_1 \cup M_2$. □

Lemma 4.45 is illustrated in Figure 4.7. Recall that if $\{x, y\} \in S$, we say that the edge cut $M = [S, \overline{S}]$ does not separate x and y, even though there may not be an (x, y)-path in $G - M$.

Lemma 4.45 *Let P' be a graph that can be made from a P_3-component P with end vertices x and y by a non-trivial edge expansion of x, and possibly an edge expansion of y. In P an edge cut M exists that does not separate x and y and that is a matching-cut in P'.*

Proof: The edges of P can be partitioned into 1-connection edge components

$F(xz)$ and $F(yz)$ that only have z in common. Since x is only incident with edges from $F(xz)$, the edge expansion of P corresponds to a non-trivial edge expansion of $F(xz)$ into F'. Now let M_1 be an edge cut for $F(xz)$ that separates x from z, contains no edges incident with z, and that is a matching in F' (Claim 4.42). Let M_2 be any matching-cut for $F(yz)$ that separates y from z. $M_1 \cup M_2$ forms the desired edge cut. Since M_1 is a matching in F' and contains no edges incident with z, this is a matching-cut in P'. Clearly, it stays a matching-cut if in addition y is expanded. □

4.7 A proof by contradiction: properties of minimum counterexamples

We want to prove that every extremal immune graph is an ABC graph. Our proof is by contradiction, so first we assume an extremal immune graph exists that is not an ABC graph. Then we consider a graph with minimum size among all such graphs, and derive a contradiction by exploring the properties of this possible counterexample. This explains the following definition.

Definition 4.46 *A graph G is a* minimum counterexample *if it is extremal immune, it is not an ABC graph, and has minimum size among all such graphs.*

Claim 4.47 *A minimum counterexample G contains no 2-connection 4-cycle.*

Proof: If there is a 2-connection 4-cycle C, apply a C4 operation on C. This yields an extremal immune graph G' (Lemma 4.11). If G' is ABC, then G is ABC (use a B operation). Otherwise, G is not a minimum counterexample. □

In Figure 4.8 the following claim and proof are illustrated.

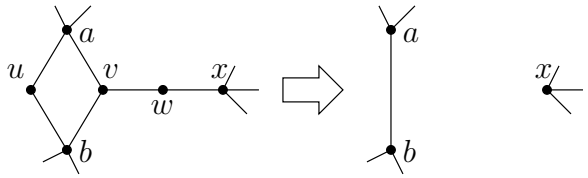


Figure 4.8: The operation from the proof of Claim 4.48

Claim 4.48 *A minimum counterexample G contains no vertices u, v and w with $N(u) = \{a, b\}$, $N(v) = \{a, b, w\}$ and $d(w) = 2$.*

Proof: Consider $G' = G - u - v - w + ab$. If $ab \in E(G)$ then G' will have two parallel edges between a and b. $|V(G')| = |V(G)| - 3$ and $|E(G')| \leq |E(G)| - 5$. Therefore $|E(G')| \leq 3(|V(G')| - 1)/2$ and this inequality is strict if $|V(G')|$ is even (Observation 4.22). It can be checked that G' is immune again, so $|E(G')| \geq 3(|V(G')| - 1)/2$ (Theorem 4.1). We conclude that $|V(G')|$ is odd

and G' is an extremal immune graph. Since G is a minimum counterexample, G' is an ABC graph, and since $|V(G')|$ is odd it is an AB graph. In G, let $N(w) = \{v, x\}$ ($x = a$ or $x = b$ is possible). In G', apply a B operation on the new edge between a and b, introducing u and v. Now a C operation can be applied on v and x introducing w. This way G is obtained, so G is ABC, a contradiction. □

Claim 4.49 *A minimum counterexample G contains no i-connection subgraph H that is an AB graph with $|V(H)| > 3$ and $i \leq 3$.*

Proof: Suppose G contains such a subgraph H.

Choose three vertices u, v and w in $V(H)$ such that $y \in V(H)\backslash\{u, v, w\}$ implies that y is only incident with edges in H (y is not a connection vertex of H).

Replace H with a triangle T with vertices u, v and w, such that none of the edges outside of H are destroyed. Call the new graph G'. (Formally $G' = ((V(G)\backslash V(H)) \cup \{u, v, w\}, (E(G)\backslash E(H)) \cup \{uv, vw, uw\})$.) It can be checked that G' is again extremal immune ($|V(G)| - |V(G')| = 2k$ for some k, and $|E(G)| - |E(G')| = 3k$). Since G is a minimum counterexample, G' is an ABC graph. By Claim 4.39, a decomposition of G' exists such that all edges of T are introduced by the same A operation. W.l.o.g., this is an A operation on u introducing v and w. By Corollary 4.37, H has a decomposition that starts with u (without C operations). In the decomposition of G', use this decomposition of H instead of the A operation introducing T. This is a decomposition of G, so G is an ABC graph, a contradiction. □

Claim 4.50 *A minimum counterexample G is simple.*

Proof: If G has vertices u and v with at least three parallel edges between them, then one of these edges can be deleted and the resulting graph is still immune, contradicting Theorem 4.1.

Now suppose there are two parallel edges between u and v. Suppose that a C2 operation on u and v gives an ABC graph G' with vertex u resulting from the contraction. Since G' is immune (Lemma 4.11), G' has odd order (Observation 4.22, Theorem 4.1) and thus is an AB graph.

First assume that two edges e and f that are part of the same A-component T are split in the construction of G from G'. Then a matching-cut for G can be obtained: consider an edge set $M \subset E(T)$ that is an edge cut in T and a matching in G (Lemma 4.44). Since T is a block in G' (Lemma 4.26), M is also an edge cut in G' (Observation 4.5), and therefore an edge cut in G.

So now we may assume that no A-component is split. We complete the proof by showing how an ABC decomposition of G can be obtained: by Corollary 4.37, G' has a decomposition that starts with u. Now first apply a C operation on u introducing v, and proceed with the rest of the decomposition. If an A-component T is introduced by an A operation on u, then $G[E(T)]$ is a triangle component that is incident with only one of u or v in G, since T is not split. Apply an A operation on either u or v instead in this new decomposition. Note

that every A-component T that is incident with u in G' is introduced by an A operation on u in the chosen decomposition of G', so by changing the operations this way, we obtain an ABC decomposition of G. □

Claim 4.51 *A minimum counterexample G is 2-connected.*

Proof: Suppose G has a cut vertex v. Let $Q_1, \ldots, Q_k$ be the components of $G - v$. Define $G_1 = G[V(Q_1) \cup \{v\}]$ and $G_2 = G[V(Q_2) \cup \ldots \cup V(Q_k) \cup \{v\}]$. Let $n = |V(G)|$, $n_1 = |V(G_1)|$, $n_2 = |V(G_2)|$, $m = |E(G)|$, $m_1 = |E(G_1)|$ and $m_2 = |E(G_2)|$. Observe that $n = n_1 + n_2 - 1$, $m = m_1 + m_2$. Let $f : n \to \lceil 3(n-1)/2 \rceil$. Then $m = f(n)$, and clearly $m_1 \geq f(n_1)$ and $m_2 \geq f(n_2)$ (Theorem 4.1).

$$m = m_1 + m_2 \geq \lceil 3(n_1 - 1)/2 \rceil + \lceil 3(n_2 - 1)/2 \rceil \geq$$

$$\lceil 3(n_1 - 1)/2 + 3(n_2 - 1)/2 \rceil = \lceil 3(n-1)/2 \rceil = m.$$

We conclude that both inequalities are equalities, and therefore $m_1 = f(n_1)$ and $m_2 = f(n_2)$, so both G_1 and G_2 are smaller extremal immune graphs. Hence both are ABC. Since the second inequality above is also an equality, we know that at least one of n_1 and n_2 is odd (otherwise both terms are rounded up). W.l.o.g. n_1 is odd, so G_1 is an AB graph. By Corollary 4.37, a decomposition of G_1 can start with any vertex. Now consider a decomposition of G_2, and add a decomposition of G_1 that starts with v. Together this is a decomposition of G, a contradiction. □

4.8 A minimum counterexample contains no C_4

In this section, we prove the following lemma:

Lemma 4.52 *A minimum counterexample G contains no C_4.*

The proof is by contradiction. Suppose subgraph C of G is a 4-cycle with edge set $\{u_1u_2, u_2v_1, v_1v_2, v_2u_1\}$. W.l.o.g., assume that $d(u_1) \geq 3$ and $d(v_1) \geq 3$, for if two neighbors on C have degree two then a matching-cut is immediate. Applying operation C4 on C gives a new graph G'. The resulting edge in G' will be called uv (vertex u results from the contraction of u_1u_2, and v results from the contraction of v_1v_2). G' is extremal immune (Lemma 4.11), and therefore G' is an ABC graph.

Consider a decomposition of G' such that edge sets $E_1, \ldots, E_k$ induce the A-components $T_1, \ldots, T_k$, and if the order of G' is even, let the edge set F induce the C-component P. These edge sets correspond to edge sets $E'_1, \ldots, E'_k$ and F' of G such that if $uv \in E_i$ (or F), then $E(C) \subseteq E'_i$ (resp. F'). $\{E_1, \ldots, E_k, F\}$ partitions $E(G')$ (Claim 4.24), and $\{E'_1, \ldots, E'_k, F'\}$ partitions $E(G)$. These edge sets induce subgraphs $T'_1, \ldots, T'_k$ and P' of G.

At first it may be confusing that parts of G' are denoted without primes and parts of G are denoted with primes, but note that G' is deduced from G,

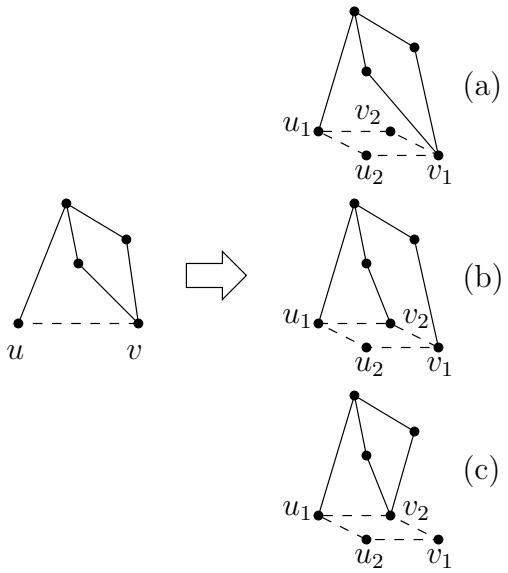


Figure 4.9: Three different ways to reverse a C4 operation

$E_1, \ldots, E_k$ and F are a natural partition of the edges of ABC graph G', and $E'_1, \ldots, E'_k$ and F' are deduced from these sets.

Since the C4 operation consists of two edge contractions and one edge deletion, we can use the notion of splitting from Section 4.2.2 for edge induced subgraphs of G'. However, note that the edge in G that corresponds to uv in G' can be u_1v_2 or u_2v_1, and on this arbitrary choice it can depend whether a subgraph $G'[M]$ with $uv \in M$ is split. Therefore we will slightly abuse the definition in this section, and say that $G'[M]$ *is split if* $G'[M-uv]$ *is not isomorphic to* $G[M-uv]$.

In Figure 4.9, some examples are shown. Only in case (b) the triangle component is split. Observe that it is possible that a triangle component $G'[M]$ is not split, while $G[M]$ is not isomorphic to a triangle component (Figure 4.9(c)). We argue that in this case we may assume w.l.o.g. that the case in Figure 4.9(c) holds: if a triangle component T_i with triangle vertices u, v and w is not split, then in G no edges of edge component $F(uw)$ are incident with u_1 or none are incident with u_2, and a similar statement is true for $F(vw)$. Suppose no edges of $F(uw)$ are incident with u_2 (u_1) in G, and no edges of $F(vw)$ are incident with v_2 (v_1) in G (Figure 4.9(a)). Then T'_i can be obtained from T_i by a B operation on uv, and renaming the resulting vertices. Thus if T_i is not split and T'_i is not a triangle component, then edges of $F(uw)$ are not incident with u_2 (u_1) in G, and edges of $F(vw)$ are not incident with v_1 (v_2) in G, so w.l.o.g. the case in Figure 4.9(c)) holds.

If the order of G' is even, one C operation is used in every decomposition of G'. In the decomposition we consider, x and y will denote the vertices in G'

on which the C operation is applied, and z will denote the vertex introduced by the C operation. The notation $F(xz)$ and $F(yz)$ will be used to denote the edge components between x and z resp. between y and z of which P consists.

Note that vertices in G' directly correspond to vertices in G, except for u and v. So if for instance $u = x$, then there is not necessarily a unique vertex in G that corresponds to x. However, if $uv \notin F$ and no edges of $F(xz)$ are incident with u_2 (u_1) in G (so $F(xz)$ is not split), then u_1 (u_2) is called vertex x in G. For the cases $x = v$, $y = u$ and $y = v$ the notation is similar.

With G, G', u, v etc. defined as above, we first state a number of claims before Lemma 4.52 can be proved.

Claim 4.53 *G' has a decomposition with at least one A-component.*

Proof: We consider a decomposition of G' such that every intermediate graph is simple if G' is simple (Claim 4.34). Suppose no A operations are applied in the decomposition. So a decomposition of G' starts with a C operation on $x = y$. If at least one B operation is applied, then G' is simple (Observation 4.19), a contradiction with our choice of the decomposition. We conclude that $G' = C_2$. Now it is easy to check that G either is ABC, or has a matching-cut, a contradiction. □

Claim 4.54 *If an A-component T_i of G' is split then the order of G' is even, and T_i contains both x and y and $x \neq y$. In addition, if G' is 2-connected then T_i is the only A-component of G'.*

Proof: Let T_i be an A-component that is split. W.l.o.g. $u \in V(T_i)$. First we construct a matching-cut M in T_i'.

If $v \notin V(T_i)$, then T_i' can be obtained from T_i by a non-trivial edge expansion of u into u_1u_2, and deleting u_1u_2. By Lemma 4.44, an edge cut M can be constructed in T_i that is a matching-cut in T_i'.

If $v \in V(T_i)$ then $uv \in E(T_i)$. T_i has a decomposition such that u, v and another vertex w are triangle vertices (Corollary 4.32). So $T_i - uv$ can be seen as a P_3-component. This P_3-component is split, so an edge cut M can be constructed in T_i that is a matching-cut in T_i' (Lemma 4.45).

In both cases an edge cut M in T_i is constructed that is a matching-cut in T_i', and that does not contain uv. If T_i is a block of G', then M is an edge cut in G' (Observation 4.5). Note that an edge cut in G' that does not contain uv is an edge cut in G, since the C4 operation can be reversed by adding a parallel edge to uv and expanding u and v, and edge expansions do not destroy edge cuts (Observation 4.10). So of T_i is a block of G', M is a matching-cut in G. Thus T_i is part of an even order block, and therefore the order of G' is even and $x \neq y$ (Lemma 4.26).

If $x \notin V(T_i)$, then consider any matching-cut M' for $F(xz)$ that separates x from z. Because of the block structure of G', either M or $M \cup M'$ is an edge cut in G' (Corollary 4.27). This edge cut in G' is also an edge cut in G. Since $x \notin V(T_i)$, we know that $V(T_i) \cap V(F(xz)) = \emptyset$ (Lemma 4.26), so $M \cup M'$ is a matching in G, and we have found a matching-cut in G.

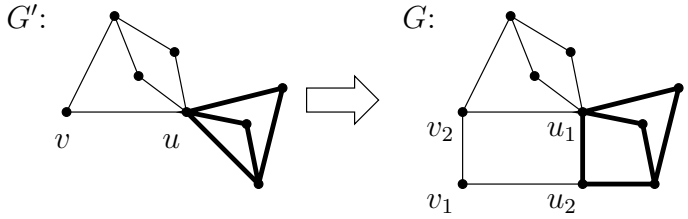


Figure 4.10: An example where G' has a cut vertex

If $x \in V(T_i)$ but $y \notin V(T_i)$, the matching-cut construction is similar. So since G is immune, the order of G' is even, $x \in V(T_i)$ and $y \in V(T_i)$. If G' is 2-connected, then T_i is the only A-component in G' (Lemma 4.26). □

Claim 4.55 *G' is 2-connected, has even order and $x \neq y$. G' has a decomposition where $G'[E_1 \cup \ldots \cup E_k]$ is an intermediate AB graph.*

Proof: Suppose G' has a cut vertex. Since G does not have a cut vertex (Claim 4.51), this cut vertex is u or v. W.l.o.g., u is a cut vertex in G'. G' contains a 1-connection A-component (Corollary 4.28). If G' contains a 1-connection A-component that does not contain uv, let T_i be this A-component, otherwise let T_i be the only 1-connection A-component.

If T_i is split, then it contains x and y, and $x \neq y$ (Claim 4.54). In this case x and y are two different connection vertices of T_i, a contradiction. So T_i is not split. Then if $uv \notin E(T_i)$, then the connection vertex of T_i in G' also corresponds to a cut vertex in G, a contradiction (Claim 4.51).

So $uv \in E(T_i)$, and because of our choice of T_i, we conclude that T_i is the only 1-connection A-component. Since u is the only cut vertex in G' in this case, G' consist of two blocks: T_i and an even order block B (Lemma 4.26). Note that B is an ABC graph (Claim 4.35), and therefore extremal immune.

If T_i' is again a triangle component (see Figure 4.9(a)), then G contains a triangle component subgraph on at least five vertices with at most two connection vertices, a contradiction with Claim 4.49. So since T_i is not split, w.l.o.g. u_2 and v_1 have degree two in T_i' (see Figure 4.9(c)). B is split, otherwise u_1 or u_2 is a cut vertex in G. Consider $B' = G[E(B) \cup \{u_1u_2\}]$. This case is illustrated in Figure 4.10, where B and B' are shown by the bold edges. If B' has a matching-cut M, then M or $M \cup \{v_1v_2\}$ is a matching-cut in G, so B' is immune. B' has one more vertex and one more edge than B, and the order of B is even, so B' is also extremal immune. Since B' has odd order and is smaller than G, B' is an AB graph. If $|V(B')| > 3$, then Claim 4.49 leads to a contradiction. So B' is a triangle on vertices u_1, u_2 and another vertex w.

Now we can show that G is ABC. Start with a decomposition of T_i. Rename u as u_1, and v as v_2. Apply an A operation on u_1 introducing w and u_2. Apply a C operation on u_2 and v_2 introducing v_1. Now graph G is obtained.

In every case a contradiction is obtained, so we conclude that G' is 2-connected. Suppose that the order of G' is odd. Then G' consists of only

one triangle component T_1 (Lemma 4.26). If T_1' is again a triangle component, then G is an ABC graph, a contradiction. But T_1 is not split (Claim 4.54), so w.l.o.g. $d(u_1) = d(v_2) = 2$, and G has a matching-cut (see Figure 4.9(c)). We conclude that the order of G' is even, and since there is at least one A-component (Claim 4.53), it follows that $x \neq y$ (Lemma 4.26).

It follows that a decomposition of G' exists where $G'[E_1 \cup \ldots \cup E_k]$ is an intermediate AB graph: since G' is 2-connected, x and y are the only connection vertices of the C-component (Lemma 4.26), and Claim 4.38 can be applied. □

Note that since $x \neq y$, G' must be simple (Observation 4.19), so we can conclude that C is an *induced* 4-cycle in G.

For the following claim recall that P is the C-component of G'.

Claim 4.56 *If $uv \notin E(P)$ then P consists only of the edges xz and yz.*

Proof: Assume the C-component P is split. We know that x and y are the only connection vertices of P (Claim 4.55, Lemma 4.26). Since we assume that uv is not part of P, w.l.o.g. $u = x$, and the C-component is split at x. So P' can be obtained from a P_3-component with end vertices x and y by a non-trivial edge expansion of x (and deletion of the resulting edge), and possibly also an edge expansion of y if $y = v$. By Lemma 4.45, an edge cut M for P exists that does not separate x and y and that is a matching-cut in P'. Since x and y are the only connection vertices of P, M is also an edge cut in G' and a matching-cut in G.

So the C-component is not split. Now if at least one B operation is applied to P, then G' contains a 2-connection 4-cycle which is also a 2-connection 4-cycle in G, a contradiction with Claim 4.47. □

So if $uv \notin E(P)$, the C-component is not split, and therefore we can unambiguously identify two vertices x and y in G corresponding to x and y in G'.

Claim 4.57 *If an A-component T_i of G' is not split, then T_i' in G is also a triangle component.*

Proof: Suppose this is not true. Note that since at least one A-component is not split, no A-component is split (Claim 4.54, Claim 4.55). If $uv \notin E(T_i)$ and T_i is not split then T_i' is isomorphic to T_i by definition, and therefore a triangle component. So $uv \in E(T_i)$. Consider a decomposition of T_i with triangle vertices u, v and w (Corollary 4.32). Suppose T_i is not split and T_i' is not a triangle component. Then w.l.o.g. in G no edges of edge component $F(uw)$ are incident with u_2 in G, and no edges of $F(vw)$ are incident with v_1 in G (see Figure 4.9(c)).

Observe that in T_i', $M = \{u_1u_2, v_1v_2\}$ is an edge cut that separates $S_1 = \{u_2, v_1\}$ from $S_2 = V(T_i')\backslash\{u_2, v_1\}$. We will show that M is also an edge cut in $G'' = G[E_1', \ldots, E_k']$, using the fact that no A-components are split. Suppose this is not true, so in $G'' - M$ there is a path R with one end vertex in S_1, one end vertex in S_2, and no internal vertices in $V(T_i')$. Since M is an edge cut for

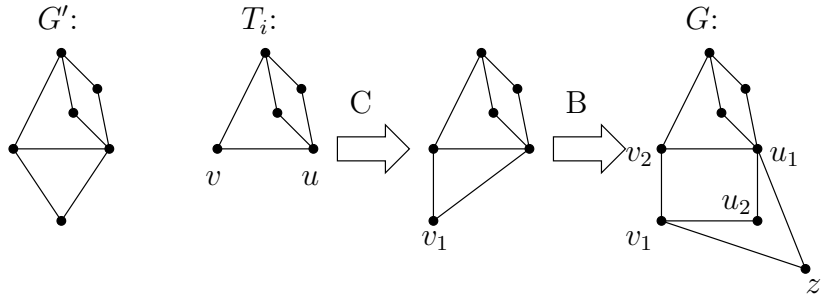


Figure 4.11: A decomposition of G from T_i

T_i', R only uses edges from A-components other than T_i. T_i shares at most one vertex with every T_j, $j \neq i$ (Lemma 4.26), and no A-components are split, so T_i' shares at most one vertex with T_j'. Therefore R contains edges from at least two A-components. In G', the edges from R are a cycle, or can be extended to a cycle using edges from T_i. This is a cycle through multiple A-components of G', that does not use edges of the C-component. Since $G'[E_1 \cup \ldots \cup E_k]$ is an AB graph (Claim 4.55), this is a contradiction with the block structure (Lemma 4.26). We conclude that such a path does not exist, and therefore $\{u_1u_2, v_1v_2\}$ is an edge cut in $G[E_1', \ldots, E_k']$.

Let $\{S, T\}$ be the partition of the vertices of $G[E_1', \ldots, E_k']$ such that $[S, T] = \{u_1u_2, v_1v_2\}$. Since G is immune, these edges are not part of a matching-cut in G. There is a decomposition of G' that ends with a C operation on vertices x and y in the AB graph $G'[E_1, \ldots, E_k]$ (Claim 4.56). We conclude that in G, w.l.o.g. $x \in S$ and $y \in T$ and x and y are incident with $[S, T]$. Since $x \neq y$ in G', w.l.o.g. $x = u_1$ and $y = v_1$ in G. This implies that $x = u$ and $y = v$ in G', and therefore T_i is the only A-component of G' (Lemma 4.26).

Consider the following decomposition (see Figure 4.11). Start with a decomposition of triangle component T_i, and rename u as u_1 and v as v_2. Apply a C operation on u_1 and v_2 introducing v_1. Apply a B operation on u_1v_1, introducing z and u_2. This is an ABC decomposition of G. This final contradiction proves the claim. □

Claim 4.58 *There is a decomposition of G' such that uv is part of an A-component.*

Proof: Suppose G' only has decompositions for which uv is part of the C-component.

Consider the case where there is an A-component T_1 that has a decomposition such that x and y are triangle vertices. Because G' is 2-connected (Claim 4.55), T_1 is the only A-component (Lemma 4.26). Let z' be the third triangle vertex in this decomposition of T_1. Now there is a decomposition of G' that starts with a triangle on x, y and z', then applies a C operation on x and y introducing z, and ends with a number of B operations. Now if instead we start

with a triangle on x, y and z and apply a C operation on x and y introducing z', a decomposition of G' is obtained such that uv is part of an A-component, a contradiction. So there is no A-component that has a decomposition where x and y are triangle vertices.

Now suppose that an A-component T_i is split. In this case, w.l.o.g. $u = x$ and $u \in V(T_i)$ since x and y are the only connection vertices of P. By Claim 4.54, $\{x, y\} \in V(T_i)$. Now T_i' can be obtained with a non-trivial edge expansion of u and deleting the resulting edge. Consider a decomposition of T_i with u, a and b as triangle vertices such that $y \in V(F(ab))$ (Claim 4.33). We have shown that $y \neq a, b$. By Lemma 4.44, an edge cut M_1 exists in T_i that is a matching-cut in T_i' such that y is not incident with edges from M_1. If y is separated from u by M_1 (in fact this is always true), M_1 can be made into an edge cut for G' and G by adding the edges of a matching-cut M_2 for $F(yz)$ that separates y from z. $M_1 \cup M_2$ is a matching in G since none of the vertices in $F(yz)$ are incident with edges of M_1.

This contradiction shows that we may assume that no A-component is split. At least one A-component exists (Claim 4.53). If G' has a single A-component and this is a triangle, then x and y are triangle vertices of the same A-component, a contradiction. Therefore, $G'' = G'[E_1 \cup \ldots \cup E_k]$ has at least five vertices, and is an AB graph (Claim 4.55). In G', the subgraph G'' has two connection vertices (x and y). No A-component is split, and x and y are both incident with exactly one A-component (Lemma 4.26). It follows that G'' is also a 2-connection AB subgraph of G, a contradiction (Claim 4.49). □

Claim 4.59 *At least one A-component of G' is split.*

Proof: Assume no A-components are split. Then if T_i is the A-component containing uv (such an A-component exists by Claim 4.58), T_i' can be obtained from T_i with a B operation (Claim 4.57). The C-component consists only of edges xz and yz (Claim 4.56), so $G' - z$ is an AB graph (Claim 4.55). uv is part of the A-component T_i in $G' - z$. By Claim 4.36, there is a decomposition of $G' - z$ that starts with the construction of T_i. There is a decomposition of T_i that starts with a triangle on u, v and another vertex w (Corollary 4.32). Instead of starting only with this triangle, start with this triangle and apply a B operation on edge uv, introducing u_2 and v_2, and rename u and v as u_1 resp. v_1. Continue with the rest of the decomposition. If an A-component T_j is introduced by an A operation on u, then T_j' is only incident with u_1 or u_2 in G, since T_j is not split. So apply an A operation on u_1 resp. u_2 instead. This is similar for v and for the C-component. This gives an ABC decomposition of G. □

We summarize the above results in the following Corollary.

Corollary 4.60 *G' consists of a single split A-component containing uv and a C-component consisting only of the edges xz and yz.*

Proof: Consider a decomposition such that uv is part of an A-component (Claim 4.58). At least one A-component T_i must be split (Claim 4.59). A split

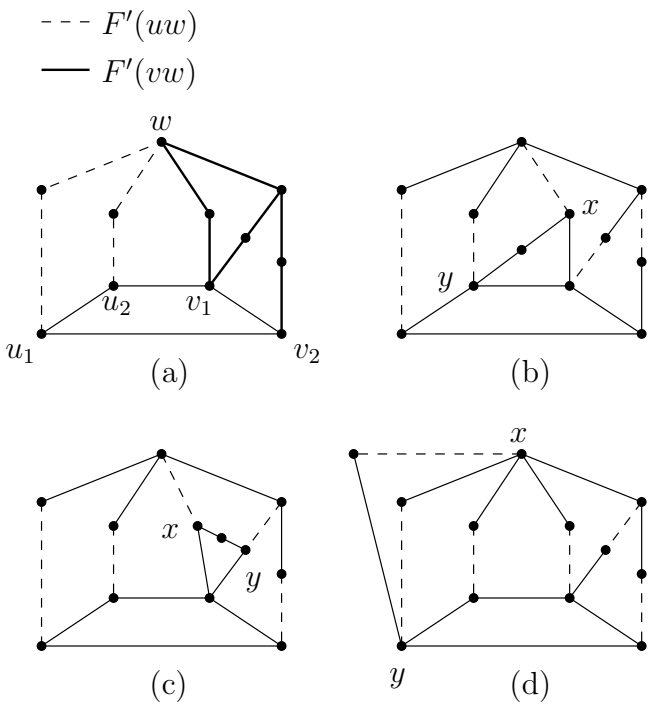


Figure 4.12: An example of $G - z$ and matching-cuts for three different choices of x and y

A-component T_i contains both x and y and $x \neq y$ (Claim 4.54). So because of the block structure and the 2-connectedness of G' (Lemma 4.26, Claim 4.55), T_i is the only A-component, so uv is part of T_i. By Claim 4.56, the C-component consists only of edge xz and yz. □

Claim 4.61 *In G, either $d(v_2) = 2$ or $d(u_2) = 2$.*

Proof: Let T be the only A-component of the decomposition of G' we consider, and let $uv \in E(T)$ (Corollary 4.60). Consider a decomposition of T with u and v as triangle vertices (Corollary 4.32), and let the third triangle vertex be w. Let $F'(uw)$ and $F'(vw)$ denote the subgraphs of G induced by the edge sets of $F(uw)$ resp. $F(vw)$. Let $T' = G - z$. See Figure 4.12(a) for an example of T'.

The outline of the proof is as follows: we will first assume that both $F(uw)$ and $F(vw)$ are split. Then, for every choice of x and y we will point out a matching-cut M in T' such that either M does not separate x from y, or x and y are not both incident with M. In the first case M is also a matching-cut in G, and in the second case $M \cup \{xz\}$ or $M \cup \{yz\}$ is a matching-cut in G. Hence we may conclude that $F(uw)$ or $F(vw)$ is not split and the statement will follow.

Assume that both $F(uw)$ and $F(vw)$ are split. Now we construct matching-cuts for $F'(uw)$ and $F'(vw)$ similar to the previous matching-cuts for a P_3-

component (see Lemma 4.45), but this time taking the position of x and y into consideration.

First, suppose that $x \in V(F(vw))\backslash\{v,w\}$, $y \notin V(F(vw))\backslash\{v,w\}$. See Figure 4.12(b) for an example. Since $F(uw)$ is split at u, we can find an edge cut $[S_1, S_2]$ for $F(uw)$ with $u \in S_1$ and $w \in S_2$, that is not incident with w and is a matching-cut in $F'(uw)$ (Claim 4.42). Also add v to S_1. Since $y \notin V(F(vw))\backslash\{v,w\}$, either $y \in S_1$ or $y \in S_2$. Suppose $y \in S_1$. Now for $F(vw)$ a matching-cut $[S_1', S_2']$ exists with $\{x,v\} \subseteq S_1'$ and $w \in S_2'$ (Claim 4.41). Since none of the edges in $[S_1, S_2]$ are incident with vertices in $F(vw)$, together these edge sets form a matching in G. These edge sets form an edge cut in T, since $[S_1', S_2'] \cup [S_1, S_2] = [S_1' \cup S_1, S_2' \cup S_2]$. Since $y \in S_1$ and $x \in S_1'$ this is also an edge cut in G' that does not contain uv, and therefore an edge cut in G. If $y \in S_2$, the matching-cut construction is similar.

This construction can easily be generalized to prove that if one of x and y is an internal vertex of one of the edge components and the other vertex is not an internal vertex of the same edge component, then a matching-cut in G can be found.

Now consider the case that both x and y are internal vertices of the same edge component, w.l.o.g. $\{x,y\} \subseteq V(F(vw))\backslash\{v,w\}$. See Figure 4.12(c). Since $F(vw)$ is split at v, for $F'(vw)$ a matching-cut $[S_1, S_2]$ exists with $\{v_1, v_2\} \subseteq S_1$ and $w \in S_2$, and either $\{x,y\} \subseteq S_1$ or $\{x,y\} \subseteq S_2$ (Claim 4.43). Similar to the previous case, we can combine this with a matching-cut $[S_1', S_2']$ for $F'(uw)$ such that w is not incident with edges of $[S_1', S_2']$ (Claim 4.42), and a matching-cut in G is found.

Finally, if x and y are both not internal vertices of any of the two edge components, then in G, $\{x,y\} \subset \{u_1, u_2, v_1, v_2, w\}$. Considering the previously constructed matching-cuts, it is clear that w.l.o.g. $x = w$. See Figure 4.12(d). Now let $M_1 = [S_1, S_2]$ be a matching-cut for $F'(uw)$ such that $\{u_1, u_2\} \in S_1$, $w \in S_2$ and w is not incident with edges in M_1 (Claim 4.42). Let $M_2 = [S_1', S_2']$ be a matching-cut for $F'(vw)$ such that $\{v_1, v_2\} \in S_1'$, $w \in S_2'$ and w is not incident with edges in M_2. Let $S = S_1 \cup S_1' \cup \{z\}$. Now $[S, \overline{S}] = M_1 \cup M_2 \cup \{xz\}$ is a matching-cut in G.

We have found matching-cuts for G in all cases, so this shows that not both $F(uw)$ and $F(vw)$ can be split. W.l.o.g. $F(uw)$ is split but $F(vw)$ is not, and edges of $F'(vw)$ are not incident with v_2, only with v_1. In this case we prove $v_2 \neq x, y$. Consider a matching-cut $[S_1, S_2]$ for the split P_3-component $G[E(F'(uw)) \cup E(F'(vw))]$ with $\{u_1, u_2, v_1\} \subseteq S_1$ (Lemma 4.45). v_2 is not part of $F'(uw)$ or $F'(vw)$, and therefore not incident with M. If $y = v_2$ ($x = v_2$) then either M or $M \cup \{yz\}$ ($M \cup \{xz\}$) is a matching-cut in G. Therefore $d_G(v_2) = 2$. □

Now we are ready to finish the proof of Lemma 4.52.

Proof of Lemma 4.52: Corollary 4.60 shows that in the decomposition of G' we consider, there is only one A-component T, $uv \in E(T)$, and T is split. We also know that $G' - z = T$. By Claim 4.61, w.l.o.g. $d(v_2) = 2$ in G. T' is

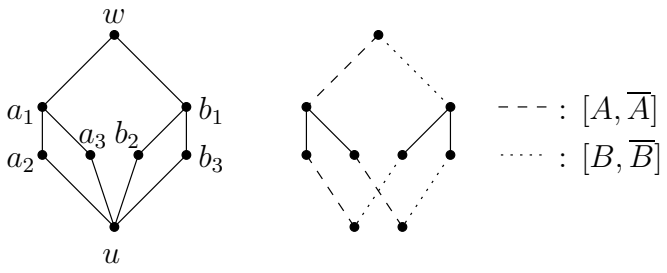


Figure 4.13: Disjoint 2-connection 4-cycles lead to disjoint matching-cuts

the subgraph of G that corresponds to T ($T' = G - z$). Throughout this proof we use the fact that if $[S, \overline{S}]$ is a matching-cut in T', it is easily extended to a matching-cut in G unless w.l.o.g. $x \in S$, $y \in \overline{S}$ and x and y are both incident with edges from $[S, \overline{S}]$.

Consider a decomposition of T such that u, v and another vertex w are triangle vertices (Corollary 4.32). We make the following observations:

1. Every 2-connection 4-cycle C in T that does not contain uv is split. If not, then in G there is also a 2-connection 4-cycle (a contradiction with Claim 4.47), unless w.l.o.g. x is equal to one of the degree 2 vertices of C. If this is the case, and y is not equal to the other degree 2 vertex of C, then the forbidden structure from Claim 4.48 is present, a contradiction. If y is the other degree 2 vertex, then in G a 2-connection $K_{2,3}$ is present, which is a 2-connection AB graph on 5 vertices, again a contradiction (Claim 4.49).

2. Since every 2-connection 4-cycle in T that does not contain uv is split, every 2-connection 4-cycle is incident with u. This is because $d_G(v_2) = 2$ (Claim 4.61). Therefore edge component $F(vw)$ consists of a single edge.

3. In a decomposition of edge component $F(uw)$ (this decomposition follows from the decomposition of T), every B operation is applied on an edge incident with u, otherwise in $F(uw)$ 2-connection 4-cycles will exist that are not incident with u (consider the last such B operation).

4. Therefore, if in a decomposition of T a B operation is applied to ua, then $d_T(a) = 3$.

5. In an edge component, no triangles or multi-edges are present. Therefore, if C is a 2-connection 4-cycle in $F(uw)$, then in any decomposition of $F(uw)$, all edges of C are introduced by the same B operation, so 2-connection 4-cycles correspond to B operations. (Observe that this is not necessarily true for 2-connection 4-cycles in T.) So a C4 operation on any 2-connection 4-cycle in $F(uw)$ yields again an edge component.

6. In $F(uw)$, no two edge disjoint 2-connection 4-cycles exist. The following proof of this statement is illustrated in Figure 4.13. Suppose in $F(uw)$ two edge disjoint 2-connection 4-cycles C_1 and C_2 exist. In $F(uw)$, if we apply C4 operations on C_1 and C_2, the resulting graph has no multi-edges since it is again an edge component. Therefore C_1 and C_2 have at most one vertex in common, and $V(C_1) \cap V(C_2) = \{u\}$. Let $V(C_1) = \{u, a_1, a_2, a_3\}$ and $V(C_2) = \{u, b_1, b_2, b_3\}$, such that $ua_1 \notin E(T)$ and $ub_1 \notin E(T)$. When we consider again the edge component resulting from the C4 operations on these two cycles, and the fact that this edge component does not have triangles, we can also deduce that a_1 and b_1 are not neighbors. Since $d_T(a_1) = d_T(b_1) = 3$ and both 2-connection 4-cycles are split, vertex sets $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$ both yield a matching-cut in T'. Since $[A, \overline{A}]$ does not extend to a matching-cut in G, w.l.o.g. $x \in A$. In this case, x is not incident with edges from $M = [B, \overline{B}]$ (since a_1 and b_1 are not adjacent), and therefore either M or $M \cup \{xz\}$ is a matching-cut in G, a contradiction.

7. Consider a decomposition of $F(uw)$. If in any intermediate graph two edge disjoint 2-connection 4-cycles exist, then in $F(uw)$ at least two edge disjoint 2-connection 4-cycles exist. So in this decomposition, a B operation is always applied to an edge introduced by the previous B operation.

These observations narrow down the possibilities for T considerably: T is obtained from the triangle on u, v and w by first applying a B operation on uw (at least one B operation is used since T is split) and then, for an arbitrary number of steps, applying a B operation on one of the two edges that is part of the single 2-connection 4-cycle in edge component $F(uw)$ and that is incident with u. So, because of the symmetry, T is completely characterized by the number of B operations that are applied. See Figure 4.14(a). We use the vertex labeling as shown in this figure. w is given the label a_0. B operations are applied on edge ua_i, introducing vertices a'_i and a_{i+1}. So ua_n is an edge if and only if exactly n B operations are applied. Let n be the number of B operations that are applied. In T, $N(u) = \{v, a_n, a'_{n-1}, a'_{n-2}, .., a'_0\}$. W.l.o.g., in T' vertex a'_{n-1} has u_1 as neighbor and vertex a_n has u_2 as neighbor. See Figure 4.14(b) for an example of a graph T' that corresponds to T.

The first matching-cut we consider in T' is $M_1 = [S_1, \overline{S}_1]$ with $S_1 = \{a_n, a_{n-1}, a'_{n-1}\}$ (See Figure 4.15(a)). The second matching-cut we consider in T' is $M_2 = [S_2, \overline{S}_2]$ with $S_2 = \{v_2, u_1\} \cup \{a'_i : u_1a'_i \in E(T')\}$ (See Figure 4.15(b)). Observe that $M_1 \cap M_2 = \emptyset$. Furthermore, $M_1 \cup M_2$ is a set of isolated edges plus the edge set of a path P of length five or six (depending on whether a'_{n-2} is adjacent to u_1 or u_2 in T') (see Figure 4.15(c)). Since x and y are incident with both M_1 and M_2, x and y are internal vertices of P. Also, if P' is the subpath of P from x to y, P' contains an odd number of edges from M_1 and an odd number of edges from M_2. Since the edges alternate and P has length at most six and x and y are internal vertices of P, P' has exactly two edges. Now if $x = u_1$ and $y = a_{n-1}$, then G has a 2-connection 4-cycle. Therefore, only two

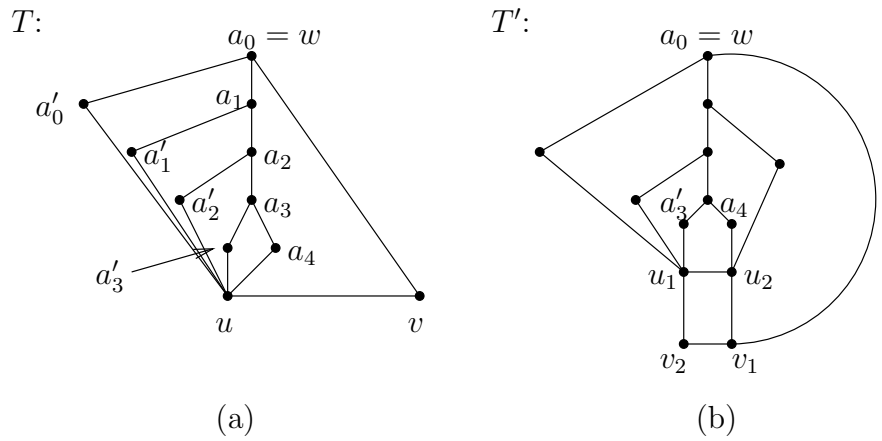


Figure 4.14: Examples of T and a corresponding T'

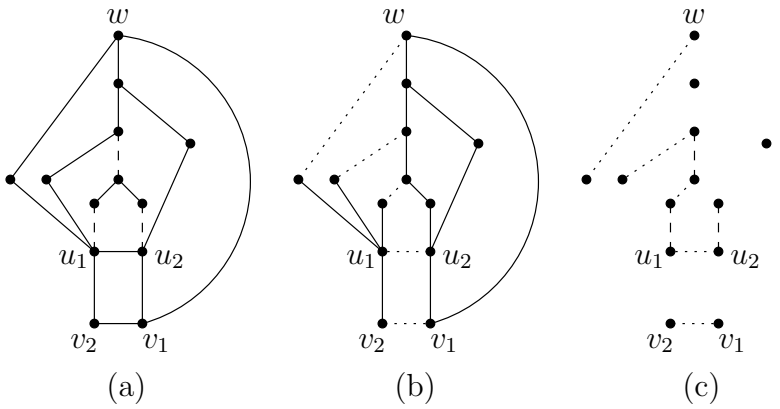


Figure 4.15: Two matching-cuts in T'

possibilities for x and y remain: either $x = u_2$ and $y = a'_{n-1}$ or $x = a'_{n-1}$ and $y = a_{n-2}$ (or $y = v$ if $n = 1$). Consider these two cases:

- Suppose $x = u_2$ and $y = a'_{n-1}$. If at least one vertex a'_i is adjacent to u_1 in T' ($i \neq n-1$), then $[S, \overline{S}]$ with $S = \{a'_i, a_i, a_{i+1}, \ldots, a_n\}$ is a matching-cut in G (see Figure 4.16(a)). Therefore, in T' all vertices a'_i are adjacent to u_2 (except for a'_{n-1}). See Figure 4.16(b). So $d_G(u_1) = 3$. In this case, the forbidden structure from Claim 4.48 is present in G, which is shown by the bold edges in Figure 4.16(b), a contradiction.

- Suppose $x = a'_{n-1}$ and $y = a_{n-2}$ (or $y = v$ if $n = 1$). If at least one vertex a'_i is adjacent to u_2 in T', then $[S, \overline{S}]$ with $S = \{a'_i, a_i, a_{i+1}, .., a_{n-1}, a'_{n-1}, z\}$ is a matching-cut in G (see Figure 4.17(a)). Therefore, every vertex a'_i is adjacent to u_1 in T'. Now $d_G(u_2) = 3$, and the forbidden structure

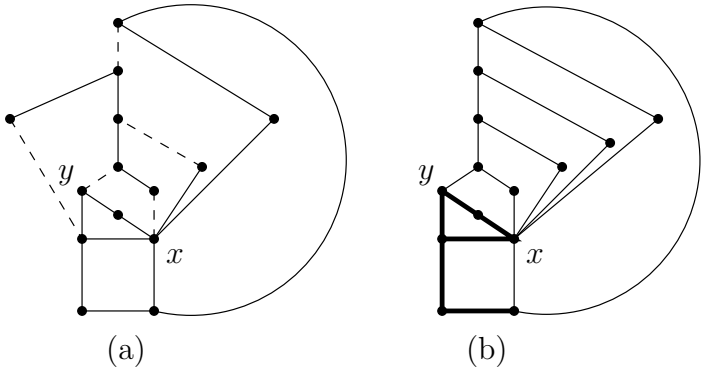


Figure 4.16: Two examples of G if $x = u'$ and $y = a'_{n-1}$

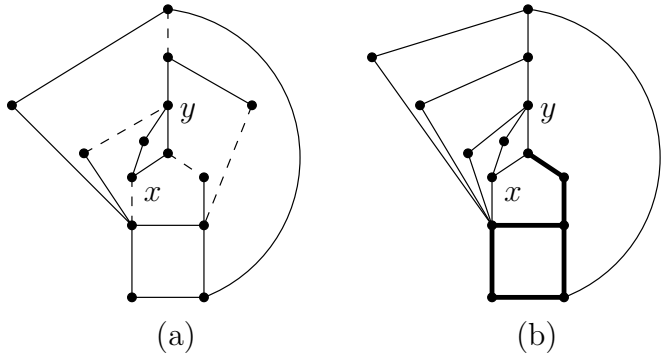


Figure 4.17: Two examples of G if $x = a'_{n-1}$ and $y = a_{n-2}$

from Claim 4.48 is again present, as indicated by the bold edges in Figure 4.17(b), a contradiction (recall that $d_G(v_2) = 2$, also if $y = v$ in G').

So in every case a contradiction is obtained, which shows that G cannot contain a C_4. □

4.9 A minimum counterexample contains no C_3

In this section, we prove the following lemma:

Lemma 4.62 *If G is a minimum counterexample, G does not contain a C_3.*

The proof is by contradiction. Suppose C is a triangle in G on vertices v_1, v_2 and v_3. We may assume w.l.o.g. that $d(v_1) \geq 3$ and $d(v_2) \geq 3$, otherwise a 1-connection triangle is present, which leads to a contradiction with Claim 4.51.

Applying operation C3 on C such that the resulting vertex is vertex v gives a new graph G'. This graph is again extremal immune (Lemma 4.11), and by our assumption, must be an ABC graph.

Consider a decomposition of G' with A-components $T_1, \ldots, T_k$, and if the order of G' is even, C-component P. The edge sets of these components induce the components $T'_1, \ldots, T'_k$ resp. P' in G. Observe that the edges of these components together with the edges $\{v_1v_2, v_2v_3, v_1v_3\}$ of C give a partition of the edges of G.

G can be constructed from G' with two edge expansions and an edge addition. So in this section we can use the terminology of Section 4.2.2 and consider whether components of G' are split or not by these operations: an edge induced component $G'[M]$ is split if it is not isomorphic to $G[M]$.

If the order of G' is even, one C operation is used in every decomposition of G'. In the decomposition we consider, x and y will denote the vertices in G' on which the C operation is applied, and z will denote the vertex introduced by the C operation. So the C-component P consists of edge components $F(xz)$ and $F(yz)$.

With G, G', v etc. defined as above, we first state a number of claims before Lemma 4.62 can be proved.

Claim 4.63 *G' has a decomposition with at least one A-component.*

Proof: The proof is very similar to the proof of Claim 4.53. □

Claim 4.64 *In G', if an A-component T_i is split then the order of G' is even and T_i contains both x and y and $x \neq y$.*

Proof: Let T_i be an A-component that is split, so $v \in V(T_i)$. First we construct a matching-cut M in T'_i.

W.l.o.g. T'_i can be obtained from T_i by a non-trivial edge expansion of v into v_1v_2 and deletion of v_1v_2, followed by an edge expansion of v_2 into v_2v_3, and deleting v_2v_3. Since a decomposition of T_i exists such that v is a triangle

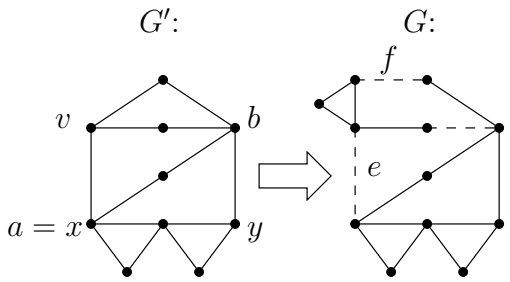


Figure 4.18: P is split

vertex (Corollary 4.32), Lemma 4.44 shows that an edge cut M exists in T_i that becomes a matching after the first edge expansion, and therefore is a matching-cut in T_i'. Observe that M is a matching-cut in G if and only if it is an edge cut in G' (the C3 operation can be reversed by adding a loop and applying two edge expansions. These operations do not destroy edge cuts). If G' is odd, then T_i is a block of G' (Lemma 4.26), so M is also an edge cut in G' (Observation 4.5). So G' is even, and T_i is not a block of G' and therefore $x \neq y$ in G'.

Suppose that in G', x is not incident with edges from M. Consider any matching-cut M' in $F(xz)$ that separates x from z. Since edges from M and M' share no end vertices (Lemma 4.26), either M or $M \cup M'$ is a matching-cut in G (Corollary 4.27). A similar matching-cut can be constructed if y is not incident with edges from M. We conclude that x and y are both incident with edges from M and therefore part of T_i. □

Claim 4.65 *G' is 2-connected.*

Proof: If G' is not 2-connected, then it contains a 1-connection A-component T_i (Corollary 4.28). If T_i is split, then it contains x and y, and $x \neq y$ (Claim 4.64). In this case x and y are two different connection vertices of T_i, a contradiction. So T_i is not split. In that case, the connection vertex of T_i in G' also corresponds to a cut vertex in G, a contradiction (Claim 4.51). □

Claim 4.66 *The order of G' is even, and $x \neq y$.*

Proof: Suppose the order of G' is odd. Then by Lemma 4.26 and Claim 4.65, G' has only one A-component. By Claim 4.64, it cannot be split. It follows that v_1 is a cut vertex in G, a contradiction (Claim 4.51).

So the order of G' is even. Since G' is 2-connected (Claim 4.65) and there is at least one A-component (Claim 4.63), $x \neq y$. □

Claim 4.67 *There is a decomposition of G' such that v is part of an A-component.*

Proof: Suppose $v \in V(P) \backslash \{x, y\}$. Since G' is 2-connected (Claim 4.65), x and y are the only connection vertices of P (Lemma 4.26), and v is only incident

with edges from P. So if P is not split, v_1 is a cut vertex in G, a contradiction (Claim 4.51). We conclude that P is split. The rest of the proof is illustrated in Figure 4.18. Consider a decomposition of P, and consider the operation that introduces v. If $v = z$ this is the C operation, otherwise a B operation. In both cases, after this operation, v is incident with exactly two edges av and bv. In P we consider the edge components $F(av)$ and $F(bv)$, which only have connection vertices v and a resp. b, and the P_3-component Q consisting of both of these. Since P is split and v is only incident with edges from Q, we can find edges $e \in F(av)$ and $f \in F(bv)$ both incident with v, that are not adjacent in G. Let M_1 be a matching-cut in $F(av)$ that contains e and separates a and v, and let M_2 be a matching-cut in $F(bv)$ that contains f and separates b and v (Claim 4.40). Together these form an edge cut $M_1 \cup M_2$ for Q that does not separate a and b. Since a and b are the only connection vertices of Q in G', this is also an edge cut in G'. So by choice of the edges e and f, this is a matching-cut in G. (Note that since $x \neq y$, this is also a matching when $Q = P$.) □

Claim 4.68 *The C-component of G' consists only of edges xz and yz.*

Proof: The proof is the same as the proof of Claim 4.56: if the C-component P is split w.l.o.g. it is split at x (Claim 4.67), which leads to a matching-cut (using Lemma 4.45, since $x \neq y$). So P is not split and if at least one B operation is applied in the decomposition of P, it leads to a 2-connection 4-cycle in G, a contradiction (Claim 4.47). □

Claim 4.69 *At least one A-component of G' is split.*

Proof: Assume no A-components are split. Since G' is 2-connected, in every decomposition of G' no A operation is applied after the C operation. So using Claim 4.68, $G' - z$ is an AB graph. v is part of an A-component (Claim 4.67). By Corollary 4.37, there is a decomposition of $G' - z$ that starts with v. Instead of starting only with v, start with the triangle on v_1, v_2 and v_3. Continue with the rest of the decomposition. If an A-component T_j is introduced by an A operation on v, then T'_j is only incident with one of the vertices v_1, v_2 or v_3 in G, since T_j is not split. So apply an A operation on v_1, v_2 resp. v_3 instead. This is similar for the C-component. This yields an ABC decomposition of G. □

Proof of Lemma 4.62: The above claims show that at least one A-component T is split (Claim 4.69), so T contains x and y (Claim 4.64), and therefore this is the only A-component (Claim 4.65, Lemma 4.26). The C-component consists only of edges xz and yz (Claim 4.68). By Lemma 4.52 we know that G cannot contain a C_4, so every C_4 in G' is split and therefore incident with v. In addition, it follows that T is not a triangle, since then the three edges of T together with one edge from $\{v_1v_2, v_1v_3, v_2v_3\}$ would form a C_4 in G.

Consider $T' = G - z$. Note that even though $T = T_1$, $T' \neq T'_1$ since the edges of cycle C are included in T'. We will describe a number of matching-cuts for T', and show that no matter how x and y are chosen, one of these matching-cuts is also a matching-cut in G. Throughout this proof we use the fact that if M

is a matching-cut in T', it is easily extended to a matching-cut in G unless M separates x from y and x and y are both incident with edges from M.

For every $u \in V(T)$ that is adjacent to v, we know that a decomposition of G' exists where u and v are triangle vertices of T (Corollary 4.32). Let w be the third triangle vertex of T in this decomposition. $F'(vw)$ is the subgraph of T' induced by the edges of edge component $F(vw)$. $F(uw)$ is a single edge, otherwise T contains a C_4 that is not incident with v. Since T is not a triangle, $F(vw)$ is not a single edge. Then $F(vw)$ is split, otherwise $F'(vw)$ contains a C_4. Now we can find a matching-cut in T' that does not contain uv.

Recall that w.l.o.g. $F'(vw)$ can be obtained from $F(vw)$ by a non-trivial edge expansion of v into v_1v_2 and deleting v_1v_2, followed by an edge expansion of v_2 into v_2v_3 and deleting v_2v_3.

Claim 4.42 shows that an edge cut M for $F(vw)$ exists that separates w from v, such that w is not incident with edges from M, and M becomes a matching after the first edge expansion. So M is a matching-cut in $F'(vw)$ that is not incident with w. $M \cup \{uw\}$ is a matching-cut for T' that does not include uv (See Figure 4.19(b)). We conclude that for every neighbor u of v, we can find a matching-cut in T' that does not contain uv. As a corollary, we find that it is not possible that $x = v$ and y is a neighbor of v in G'.

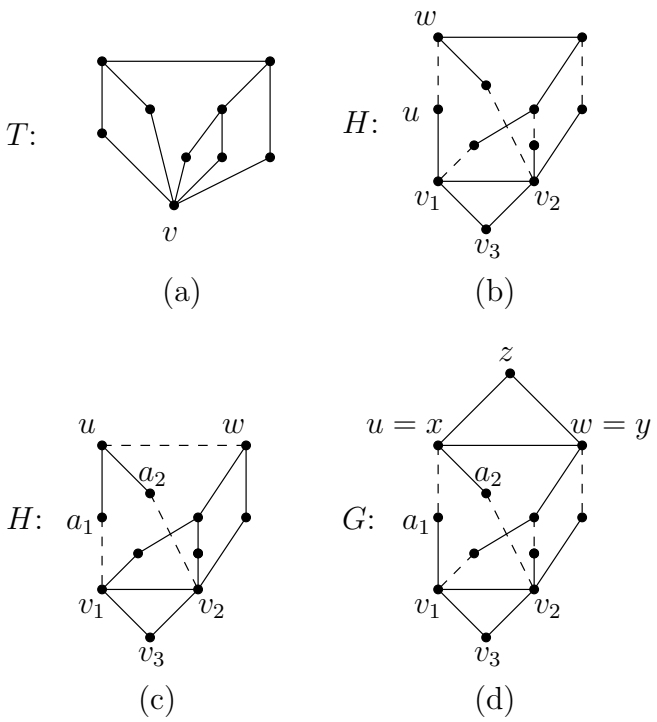


Figure 4.19: Two matching-cuts in T' and one in G

Since T is not a triangle, in a decomposition of T, at least one B operation is used. Consider the last B operation. The 2-connection 4-cycle corresponding to this operation is split so it has connection vertices v and another vertex u. Let a_1 and a_2 be the other vertices of this 2-connection 4-cycle. If we consider the intermediate graph in the decomposition from which T is obtained, we know it has a decomposition with triangle vertices u and v (Corollary 4.32), so T also has a decomposition with triangle vertices u, v and another vertex w. W.l.o.g. we assume that a_1 is adjacent to v_1 in T', and a_2 is adjacent to v_2 in T'. Now we can find a matching-cut in G: $F(uw)$ is a single edge again. In T', edge set $M = \{uw, a_1v_1, a_2v_2\}$ is a matching-cut (see Figure 4.19(c)). If the distance from x to y in T' is two, then G contains a C_4. So since M is not part of a matching-cut in G, the distance from x to y is one. Above we showed that it is not possible that $x = v$ and y is a neighbor of v in G', so the only remaining possibility is $x = u$ and $y = w$.

Finally consider the following matching-cut: take any matching-cut M for $F(vw)$ that separates v from w. If M is incident with v_1 in T', $M \cup \{a_2v_2, a_1u\}$ is a matching-cut in G, otherwise $M \cup \{a_1v_1, a_2u\}$ is a matching-cut (see Figure 4.19(d)).

Now in every case we can find a matching-cut in G, a contradiction. □

4.10 The proof of the conjecture

In this section, we prove Conjecture 4.3:

Theorem 4.70 *If graph $G = (V, E)$ is extremal immune, then G is ABC.*

Proof: Suppose this is not true, so there exist counterexamples, which are extremal immune graphs that are not ABC. Let G be a graph with minimum size among these counterexamples. Claim 4.50 shows that G is simple, so no C2 operation can be applied to it. Lemma 4.52 shows that G cannot contain a 4-cycle, so the C4 operation cannot be applied to it. Lemma 4.62 shows that G cannot contain a triangle, so no C3 operation can be applied.

Suppose a P2 operation can be applied to G. Then in G there are neighbors u and v, with $d(u) = 3$ and $d(v) = 2$. x and y are the other neighbors of u, and z is the other neighbor of v. If $z = x$ or $z = y$ then G contains a C_3, a contradiction. If x and z are neighbors or y and z are neighbors, then G contains a C_4, also a contradiction. So after the P2 operation is applied, graph G' is obtained that contains edge xy and vertex z, and z is not equal to or adjacent to x or y. G' is an ABC graph since it is again extremal immune (Lemma 4.11). Clearly, G' cannot be a K_1, a C_2 or a C_3. For every ABC graph G' other than these three graphs and every edge $e \in E(G')$, we can show that there is a 4-cycle or triangle that does not contain e:

If $G' \neq C_2$, we may assume there is at least one A-component (Claim 4.34). If G' contains an A-component of order at least 5, then the statement follows from Claim 4.31. Otherwise, if G' has at least two A-components, then two

disjoint triangles are easily found. So G' must consist of a triangle and a C-component. If on this C-component at least one B operation is applied, then we have a C_4 and a C_3 which are disjoint. So G' is a diamond (a K_4 minus one edge). For this graph the statement is also true.

So in G' there is a 4-cycle or triangle that does not contain xy. This corresponds to a 4-cycle or triangle in G, a contradiction.

We conclude that no C2, C3, C4 or P2 operation can be applied to G, which is a contradiction with Lemma 4.12. Therefore a minimum counterexample does not exist. □

4.11 Recognizing ABC graphs

Algorithm 2 is an algorithm that recognizes AB graphs, and Algorithm 3 is an algorithm that recognizes ABC graphs. A *B operation 4-cycle* of G is a 4-cycle in G that has no adjacent connection vertices. Note that B operations always introduce B operation 4-cycles. Recall that a 1-connection triangle has *at most* one connection vertex: if the input for Algorithm 2 is AB, then the last C3 operation in the algorithm is applied when $G = K_3$.

Algorithm 2 Recognition of AB graphs

INPUT: A graph G.

```
while G contains a 1-connection triangle or B operation 4-cycle C do
  Apply a C3 resp. C4 operation on C in G
end while
if |V(G)| = 1 then
  return YES, stop
else
  return NO, stop
end if
```

Theorem 4.71 *Algorithm 2 recognizes AB graphs in polynomial time, and Algorithm 3 recognizes ABC graphs in polynomial time.*

Proof: We first prove the correctness of Algorithm 2. We prove by induction over (odd values of) $n = |V(G)|$ that Algorithm 2 returns YES when the input G is AB. If $G = K_1$ then YES is returned. Otherwise, G contains at least one 1-connection triangle or B operation 4-cycle C (consider the last operation in a decomposition). A C3 resp. C4 operation on any such C results in an AB graph G' (Claim 4.35 resp. Claim 4.30). By induction, Algorithm 2 now returns YES.

If Algorithm 2 returns YES for graph G, then reversing the C3 and C4 operations one by one yields an AB decomposition of G: the decomposition starts with a K_1, and since only on 1-connection triangles and B operation 4-cycles C3 resp. C4 operations were applied, the reverse operations correspond to A resp. B operations.

Algorithm 3 Recognition of ABC graphs

INPUT: A graph G.
  if $|V(G)|$ is odd **then**
    use Algorithm 2
  end if
  while G contains a 1-connection triangle or B operation 4-cycle C **do**
    Apply a C3 resp. C4 operation on C in G
  end while
  for all vertices $v \in V(G)$ with $d(v) = 2$ **do**
    if $G - v$ is AB **then**
      return YES, stop
    end if
  end for
  return NO, stop

In every iteration of Algorithm 2, graph G loses two vertices, so at most $(n-1)/2$ iterations are needed. All steps can be done in polynomial time, hence Algorithm 2 is a polynomial time algorithm.

We now prove the correctness of Algorithm 3. Let input graph G be ABC. If the order of G is odd then we showed above that YES is returned. Otherwise, we prove by induction over (even values of) $n = |V(G)|$ that YES is returned. If a C4 operation is applied on a B operation 4-cycle of G to yield G', then G' is again ABC (Claim 4.30), and we can use induction. If G contains a 1-connection triangle C with connection vertex v, then C is a block of G, and therefore it is an A-component in any decomposition (Lemma 4.26). Then there is a decomposition in which an A operation on v that introduces C is the last operation. The graph obtained by a C3 operation on C is an intermediate graph in this decomposition, and thus is ABC. By induction, the algorithm returns YES. If G contains no 1-connection triangle or B operation 4-cycle, then every decomposition of G ends with a C operation. Consider a decomposition where the C operation introduces a vertex z. $G - z$ is an AB graph, so Algorithm 3 returns YES. To evaluate whether $G - z$ is an AB graph, Algorithm 2 is used.

Similar to the previous case, if Algorithm 3 returns YES, then reversing all operations applied on G by both algorithms yields a decomposition of G: reversing the vertex deletion in the for-loop corresponds to the C operation.

Note that all steps of Algorithm 3 can be done in polynomial time, including the recognition of AB graphs. The number of iterations in the for-loop is at most n, and the number of iterations of the while-loop is at most $n/2$, so Algorithm 3 has polynomial time complexity. □

Since we have established that extremal immune graphs are exactly ABC graphs, we have the following corollary.

Corollary 4.72 *For graphs $G = (V, E)$ with $|E| = \lceil 3(|V| - 1)/2 \rceil$, we can decide in polynomial time whether G is immune.*

Chapter 5

Extremal results for spanning trees with many leaves

5.1 Introduction

In this chapter we study spanning trees with many leaves, or equivalently, small connected dominating sets (See Section 5.2.2). We are interested in statements of this type: every graph on n vertices that satisfies a certain set of properties, has a spanning tree with at least $\alpha n + \beta$ leaves ($0 < \alpha < 1$). One statement of this type was proved independently by Linial and Sturtevant [41] and Kleitman and West [38]:

Theorem 5.1 (Linial & Sturtevant, Kleitman & West) *Every connected graph on n vertices with $\delta \geq 3$ has a spanning tree with at least $\lceil \frac{1}{4}n + 2 \rceil$ leaves.*

This bound is best possible for every n: for $n \bmod 4 \in \{0, 1\}$, in [38] examples are given of connected graphs with $\delta = 3$ that do not have spanning trees with more than $\lceil \frac{1}{4}n + 2 \rceil$ leaves. For $n \bmod 4 \in \{2, 3\}$, such examples can also be found: Figure 5.1 shows some examples for various values of n, which should make clear how to construct examples for every n. The encircled vertices show how leaves can be chosen to construct a spanning tree with maximum number of leaves. One can see that these examples contain many *diamonds*: a diamond is a K_4 minus one edge. An immediate question is whether the bound can be improved when diamonds are forbidden. One result in this direction is by Griggs, Kleitman and Shastri [34].

Theorem 5.2 (Griggs, Kleitman & Shastri) *Every connected cubic graph without diamonds on n vertices has a spanning tree with at least $\lceil (n + 4)/3 \rceil$ leaves.*

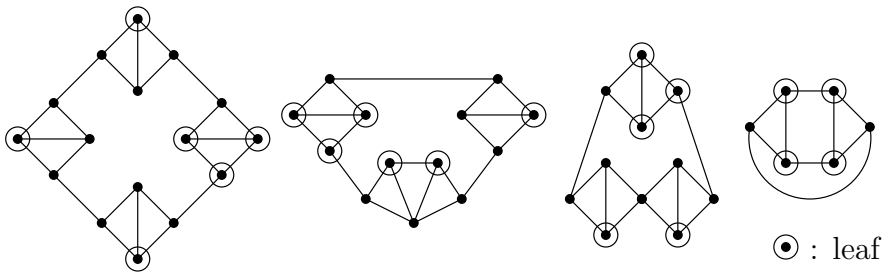


Figure 5.1: Theorem 5.1 is best possible for $n = 16, 13, 11, 6$

The bound in Theorem 5.2 is also best possible in some sense: of all the valid bounds of the form $\alpha n + \beta$ for this class, this bound maximizes α, and of all of those bounds that maximize α, this bound maximizes β. Equivalently, we can say that this is the best possible asymptotically sharp linear bound for this class. This is proved in [34] by first constructing examples of graphs with no more than $\lceil (n+4)/3 \rceil$ leaves for every n divisible by six: this shows that $\alpha = \frac{1}{3}$ is best possible. The single graph Q_3 then shows that $\beta = \frac{4}{3}$ cannot be increased (Q_3 has eight vertices and no spanning tree with more than four leaves). However, for most other values of n no examples are known that show that this bound is sharp. See Section 5.8 for a discussion on the sharpness of this bound and the other bounds in this chapter.

Our first question is what kind of bounds can be obtained when we consider graphs with $\delta \geq 3$ instead of cubic graphs, but still forbid (certain types of) diamonds. In this chapter, we present one best possible bound for these graphs. In addition, even though the proof of Kleitman and West for Theorem 5.1 is very short and elegant, the proof of Theorem 5.2 uses the same techniques, but consists mainly of a very long and intricate case study. Therefore we are also interested in new techniques for proving results of this kind.

In Section 5.4 we prove that the bound from Theorem 5.2 also holds for graphs with $\delta \geq 3$ without triangles:

Theorem 5.3 *Every connected graph on n vertices without triangles with $\delta \geq 3$, has a spanning tree with at least $\lceil (n+4)/3 \rceil$ leaves.*

Our main result is that for graphs with $\delta \geq 3$, the following result holds. Here a *cubic diamond* is a diamond induced by vertices of degree three.

Theorem 5.4 *Every connected graph on n vertices with $\delta \geq 3$, that contains D cubic diamonds, has a spanning tree with at least $\lceil (2n - D + 12)/7 \rceil$ leaves.*

Theorem 5.4 is proved in Section 5.5. See also Section 5.5 for a slightly sharper formulation of Theorem 5.4. The bounds in Theorem 5.3 and Theorem 5.4 are best possible, in the same sense as explained above. This is shown in Section 5.6.

Apart from a small difference of $\frac{2}{7}$ in the constant, Theorem 5.4 can be seen as a generalization of Theorem 5.1: since $D \leq \frac{1}{4}n$ (diamonds induced by degree

three vertices are vertex disjoint), it follows directly from Theorem 5.4 that connected graphs with $\delta \geq 3$ have a spanning tree with at least $\lceil (2n - D + 12)/7 \rceil \geq \lceil (\frac{7}{4}n + 12)/7 \rceil = \lceil \frac{1}{4}n + \frac{12}{7} \rceil$ leaves.

The proofs we will present use techniques that are different from previously used techniques. Theorem 5.3 has a relatively short proof that demonstrates these techniques, but for Theorem 5.4 a more elaborate proof is needed. In Section 5.3, the ideas behind our method are introduced.

Our proofs are constructive, and correspond to efficient and simple algorithms: Spanning trees that satisfy the bound from Theorem 5.3 can be found with an algorithm that repeatedly adds vertices or pairs of vertices to a leaf set (according to some simple rules) until no more leaves can be added, and then constructs a spanning tree with those leaves. Spanning trees that satisfy the bound from Theorem 5.4 can be found using a similar algorithm that also involves local search on these leaf sets. These algorithms are explained in Section 5.7. The formulation of the algorithms is very basic, and there is a lot of room to customize them for different purposes; whether the goal is to speed them up, find better trees or tune them to specific instances.

Finally, in Section 5.8, we discuss some possible ways to extend or improve the current results.

5.2 Preliminaries

5.2.1 An alternative definition of blocks

In this chapter, the *blocks* of a graph are maximal 2-connected subgraphs. So unlike in the usual definition, a K_2 is not a block. Therefore bridges are not contained in any block of the graph. The following lemma is an easy to prove variant of a well-known lemma.

Lemma 5.5 *A connected graph without leaves and with cut vertices has at least two blocks that contain exactly one cut vertex.*

5.2.2 Spanning trees and connected dominating sets

The following close relation between spanning trees and connected dominating sets is well-known. If T is a spanning tree for $G = (V, E)$ with leaf set L, then $V \backslash L$ is a connected dominating set, unless $G = K_1$ or $G = K_2$: $G - L$ is connected, and every vertex in L is adjacent to a vertex not in L. Similarly, for every connected dominating set S, we can easily find a spanning tree for which $V \backslash S$ is a subset of the leaf set: choose any spanning tree of $G[S]$, and for every vertex $u \notin S$ add an edge between u and one of its neighbors in S. This leads to a connected, spanning subgraph of G without cycles. So $G \neq K_1, K_2$ has a spanning tree with at least l leaves if and only if G has a connected dominating set with at most $|V| - l$ vertices. Note that this construction can be done in polynomial time, so a polynomial time algorithm for constructing

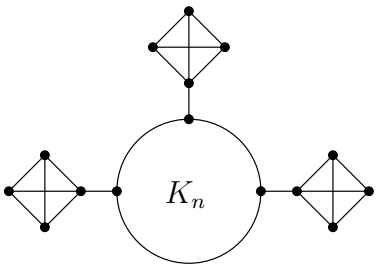


Figure 5.2: A graph with no small 2-CD-set

connected dominating sets leads to a polynomial time algorithm for finding the corresponding spanning trees.

In the remainder of the chapter, we will consider connected dominating sets instead of leaf sets of spanning trees. We feel that this is more practical for our purposes, which is illustrated by our methods, and discussed in Section 5.8. We will call connected dominating sets *CD-sets* for short. If S is a CD-set, we will still call the vertices in $V \backslash S$ the *leaves* of S. A neighbor u of v is called a leaf neighbor or CD-set neighbor of v if u is a leaf or CD-set vertex, respectively. If S is a CD-set for G and $G[S - v]$ is not connected, v is called a *connector*. If S is a CD-set for G and $S - v$ is not dominating, v is called a *dominator*. Observe that if S and $S' \subset S$ are CD-sets for G, and $v \in S$ is a dominator or connector, then $v \in S'$. A leaf v is called an *i-leaf* if it has i CD-set neighbors. A CD-set is called a *2-CD-set* if all its leaves are 1-leaves or 2-leaves.

5.3 Introduction to the method

We are interested in finding bounds of this form: every (connected) graph G, with possibly some additional properties, has a CD-set S with $|S| \leq \alpha|V(G)| - \beta$, where $0 < \alpha < 1$ and β is any constant. It is clear that when G can have many vertices of degree two, no such bounds can be obtained (consider P_n), so we only consider graphs with $\delta \geq 3$.

The first question one can ask is: how good are the bounds that we can obtain using *minimal* CD-sets? The idea is to start with a CD-set containing all vertices, and removing vertices that are not connectors or dominators, until nothing can be removed anymore, and a minimal CD-set is obtained. It is easy to see that with this method we cannot guarantee that a certain fraction of the vertices will become leaves: consider for instance the wheel W_n, where there is only one CD-set containing one vertex, and all other minimal CD-sets contain $n - 3$ vertices.

The reason that this method fails is that for some examples, if we make a wrong initial choice, we will have a leaf with many neighbors in the CD-set, all of which have degree three. This then can then again be seen as finding a CD-set in a graph with many degree two vertices.

So the next idea is to only remove vertices from the CD-set that have few neighbors in the CD-set, for instance at most two neighbors. In that case we are looking for minimal 2-CD-sets. The first problem is that some graphs do not have any small 2-CD-sets: see for instance the graph G in Figure 5.2. G has a CD-set on 6 vertices, but no 2-CD-set on fewer than $|V(G)| - 9$ vertices. In addition, for some other graphs (e.g. K_n), we need to remove arbitrarily many vertices to go from one 2-CD-set to the next, which makes it hard to find efficient algorithms and short proofs for results on minimal 2-CD-sets.

Fortunately, in order to prove a good bound it is not necessary that leaves have few neighbors in the CD-set, it is only necessary that they have few neighbors of degree three in the CD-set. To put it differently, we often may ignore edges between two vertices of high degree in our analysis. This leads to the following definitions of certain types of CD-sets.

Definition 5.6 *$S \subseteq V(G)$ is a* standard CD-set *for graph G if the following properties hold:*

1. *S is a 2-CD-set for G.*

2. *Every $v \in S$ has $d_G(v) \geq 3$.*

Definition 5.7 *$S \subseteq V(G)$ is a* potential standard CD-set *for G if there exists a spanning subgraph G' of G such that S is a standard CD-set for G'. Such a graph G' is called a* realization *of S. G' is a* maximal realization *of S if there is no other realization of S of which G' is a subgraph.*

We will show that the notion of minimal potential standard CD-sets is strong enough to obtain good bounds, and that potential standard CD-sets are easy to work with within an algorithmic context.

For a given potential standard CD-set S, every maximal realization has the same number of edges: if a leaf v has k neighbors in S, then $k - 2$ edges between v and S need to be deleted in order to obtain a realization. In a maximal realization exactly $k - 2$ of these edges are deleted. This is true for every leaf, deleted edges count towards only one leaf, and no other edges have to be deleted. So maximal realizations can alternatively be defined as having maximum number of edges.

Using minimal potential standard CD-sets, we will be able to prove the bounds from Theorem 5.3 and Theorem 5.4, expressed in terms of CD-sets instead of spanning trees. We first restrict ourselves to connected graphs without triangles with $\delta \geq 3$. In the next section we will show that in this case, for any minimal potential standard CD-set S of G and any minimal CD-set $S' \subseteq S$, $|S'| \leq \frac{2}{3}|V(G)| - \frac{4}{3}$. By imposing two additional restrictions on the minimal potential standard CD-set S, we can prove Theorem 5.4 in a similar way.

5.4 Small CD-sets for graphs without triangles

5.4.1 Construction and properties

In this section we state a number of properties for potential standard CD-sets S and their realizations G'. In Section 5.4.2 we will prove a bound for pairs of S and G' that satisfy these properties. But first we will show that such a pair S and G' exists for every connected graph G with $\delta(G) \geq 3$.

The next theorem states the essential properties of minimal potential standard CD-sets that we need to prove our bound. Property 2 is added to allow an easier formulation of Properties 4 and 5, though it is not necessary for the method. Similarly, Property 3 is only added to ensure G' is a maximal realization, which simplifies the later proofs. We remark that for every minimal potential standard CD-set S, a realization G' can be found such that G' and S satisfy the properties from Theorem 5.8. However, from now on we are not concerned with minimality, only with the properties stated in Theorem 5.8. Note that for this theorem, we do not yet need to exclude triangles.

Theorem 5.8 *A connected graph G with $\delta(G) \geq 3$ has a spanning subgraph G' and a CD-set S with the following properties. Degrees, neighbors and all other graph related notions are taken with respect to G'.*

1. *S is a standard CD-set (for G').*
2. *If $u, v \in S$, $d(u) \geq d(v) \geq 4$ and $uv \in E(G')$, then uv is a bridge of $G'[S]$.*
3. *Edges in $E(G) \backslash E(G')$ are between vertices in S, or between a vertex in S and a 2-leaf.*
4. *Every vertex $v \in S$ that is neither a dominator nor a connector has at least three neighbors in S, and all of its neighbors in S have degree three.*
5. *If $\{u, v\} \subseteq S$, $uv \in E(G')$ and u and v are both neither dominators nor connectors, then $G'[S] - u - v$ is not connected.*

Proof: We consider a CD-set S and spanning subgraph G' such that S is a standard CD-set for G', that are optimal according to the following priorities:

- Minimize $|S|$.
- Minimize $|E(G'[S])|$.
- Maximize $|E(G')|$.

By this we mean that among all pairs of S and G' that minimize $|S|$, we consider pairs that minimize $|E(G'[S])|$. Among all these pairs, we choose one of those that maximizes $|E(G')|$. Note also that V is a standard CD-set for G, so there is at least one such pair S and G'. We show that for S and G' chosen this way, the above properties hold: if not, we can find an improved S and G' according to the priorities.

Property 2: If $u, v \in S$, $d(u) \geq d(v) \geq 4$ and $uv \in E(G')$, then uv is a bridge of $G'[S]$.
Proof: Consider $u, v \in S$ with $d(u) \geq 4$, $d(v) \geq 4$ and $uv \in E(G')$. If uv is not a bridge in $G'[S]$ then S is a standard CD-set for $G' - uv$. Since this decreases $|E(G'[S])|$, this change is an improvement.

Property 3: Edges in $E(G) \backslash E(G')$ are between vertices in S, or between a vertex in S and a 2-leaf.
Proof: Suppose a 1-leaf u is incident with an edge $uv \in E(G) \backslash E(G')$. If $v \in S$, then adding edge uv to G' preserves the standard CD-set; u becomes a 2-leaf. If there is an edge $uv \in E(G) \backslash E(G')$ with $u, v \notin S$, then adding uv clearly also preserves the standard CD-set. In both cases, the values from the first two priorities are not changed, and $|E(G')|$ increases, which is an improvement. We conclude that all edges in $E(G) \backslash E(G')$ are incident with at least one vertex in S and none are incident with 1-leaves.

Property 4: Every vertex $v \in S$ that is neither a dominator nor a connector has at least three neighbors in S, and all of its neighbors in S have degree three.
Proof: Let v be a vertex in S that is not a dominator or connector. If v has at most two neighbors in S, then $S - v$ is again a CD-set, for which v is a 2-leaf, so this is an improvement. Suppose v has a neighbor $u \in S$ with $d(u) \geq 4$. Since v is not a connector, uv is not a bridge of $G'[S]$, so by Property 2, $d(v) = 3$. Then $S - v$ is a standard CD-set for $G' - uv$, an improvement.

Property 5: If $\{u, v\} \subseteq S$, $uv \in E(G')$ and u and v are both neither dominators nor connectors, then $G'[S] - u - v$ is not connected.
Proof: Suppose u and v in S are neighbors which are both not connectors or dominators. By Property 4, u and v have at least three neighbors in S, and all of those neighbors have degree three. It follows that u and v both have degree three, and therefore they have no neighbors outside of S. So $S - u - v$ is again a dominating set, and u and v are 2-leaves with respect to $S - u - v$ and G'. It follows that $S - u - v$ is an improved standard CD-set for G', unless $S - u - v$ is not a connected set. □

Now that we have proved the existence of a CD-set with these properties, we can use them to establish upper bounds on the size of the CD-set, related to the properties of the input graph.

5.4.2 An upper bound for the size of the constructed CD-set

Our basic approach behind the deduction of a good upper bound for the size of a CD-set S is based on the following idea. Let S be a potential standard CD-set for G and G' a realization, and suppose they satisfy the properties from Theorem 5.8. For showing that $|S|$ is roughly bounded from above by $\frac{2}{3}|V(G)|$,

it is sufficient to show that $|S| \leq 2|\overline{S}|$. This is easy when $G'[S]$ is a tree: vertices with degree one resp. two in $G'[S]$ have at least two resp. one neighbors in $\overline{S}$, since these vertices have degree at least three in G'. A tree has average degree less than two, so there are at least $|S|$ edges in $[S, \overline{S}]$. Combining this with the fact that vertices in $\overline{S}$ are incident with at most two of these edges, we obtain $|S| \leq 2|\overline{S}|$. This is the basic idea behind the proof of our bound, which will be refined later, also for the case when $G'[S]$ is not a tree.

Observe that when $G'[S]$ is not a tree, there may be vertices in S that are not connectors or dominators, even when S is a *minimal* potential standard CD-set. In this case we can find a minimal CD-set $S' \subset S$. It is for this set S' that we prove the bound. To improve the clarity of the exposition, we first state the theorem which yields this bound, and then prove the two lemmas that are used in its proof.

Theorem 5.9 *Let $G = (V, E)$ be a connected graph with $\delta(G) \geq 3$, and let G' and S be a spanning subgraph and CD-set for G that satisfy the properties from Theorem 5.8, such that in addition no block of $G'[S]$ contains a triangle. If $S' \subseteq S$ is a minimal CD-set, then $|S'| \leq (2|V| - 4)/3$.*

Proof: In order to prove this theorem, we assign weights w to the vertices of S such that the total weight $w(S)$ is $|V \backslash S| + \frac{3}{2}|S \backslash S'|$. We show that $w(S) \geq \frac{1}{2}|S| + 2$. Then we can combine these equations to obtain the result.

The weight assignment is as follows: all leaves of S distribute a weight of 1 equally among their neighbors in S' (vertices in $S \backslash S'$ receive no weight yet). This means we have assigned a total weight of $|V \backslash S|$. Next, we assign a weight of $\frac{3}{2}$ to each vertex in $S \backslash S'$.

In order to show that $w(S) \geq \frac{1}{2}|S| + 2$, we prove three lower bounds for this weight assignment w.

1. A vertex v with degree two in $G'[S]$ is a dominator or connector (Property 4 from Theorem 5.8), so $v \in S'$. Since v has at least one leaf neighbor $u \notin S$ (Property 1), and u has at most two neighbors in $G'[S']$, u assigns at least $\frac{1}{2}$ to v. This shows that for a degree two vertex v, $w(v) \geq \frac{1}{2}$.

2. A vertex v with degree one in $G'[S]$ is a dominator, since it cannot be a connector, and it must be a connector or dominator (Property 4). From the 1-leaf adjacent to it, it receives a weight of 1. In addition, since v has degree at least three and leaves have at most two CD-set neighbors, it gains an additional weight of at least $\frac{1}{2}$. So for a degree one vertex v, $w(v) \geq \frac{3}{2}$.

3. Let $G'[B]$ be a block of $G'[S]$, and let L denote the set of vertices in B with two neighbors in B, and $H = B \backslash L$ the set of vertices in B with at least three neighbors in B. Then $|L| \geq |H| + 1$ (this is shown below in Lemma 5.10). Let $C \subset S$ denote the set of connectors, which are the cut vertices of $G'[S]$.

A vertex in L that is not a connector also has degree two in $G'[S]$, and therefore must be a dominator (Property 4). A dominator in B has weight at least 1. We consider two cases:

(a) Suppose $|(S\backslash S') \cap B| \geq 1$. Then one of the vertices in H has an additional weight of at least $\frac{3}{2}$. We have $w(B\backslash C) \geq \frac{3}{2} + |L\backslash C| \geq \frac{3}{2} + |L| - |C \cap B|$. Since $|L| \geq |H| + 1$, $|L| \geq \frac{1}{2}|B| + \frac{1}{2}$. Hence $w(B\backslash C) \geq 2 + \frac{1}{2}|B| - |C \cap B|$.

(b) Suppose $(S\backslash S') \cap B = \emptyset$. Suppose B contains a vertex v that is not a dominator or connector with respect to S. Since $v \in S'$, it is a connector or dominator with respect to S'. But $(S\backslash S') \cap B = \emptyset$, so v is also not a connector for S'. Therefore, in S, v is adjacent to a 2-leaf u, and the other CD-set neighbor of u is in $S\backslash S'$. In this case, v receives a weight of 1 from u. Recall that dominators also receive a weight of at least 1. So all vertices in $B\backslash C$ receive a weight of at least 1. It follows that $w(B\backslash C) \geq |B\backslash C| = |B| - |C \cap B|$. Since $G'[B]$ is not a triangle, $|B| \geq 4$. Hence $w(B\backslash C) \geq \frac{1}{2}|B| + 2 - |C \cap B|$.

In both cases we have $w(B\backslash C) \geq \frac{1}{2}|B| + 2 - |C \cap B|$.

If $G'[S] \neq K_1$, Lemma 5.11 below shows that $w(S) \geq \frac{1}{2}|S| + 2$. In that case we have

$$\frac{3}{2}|S\backslash S'| + |V\backslash S| = w(S) \geq \frac{1}{2}|S| + 2 \Leftrightarrow$$

$$\frac{3}{2}|S| - \frac{3}{2}|S'| + |V| - |S| \geq \frac{1}{2}|S| + 2 \Leftrightarrow$$

$$|V| - 2 \geq \frac{3}{2}|S'| \Leftrightarrow |S'| \leq (2|V| - 4)/3.$$

If $G'[S] = K_1$, then this final inequality follows easily since $|V| \geq 4$ (G' is a spanning subgraph of a graph with $\delta \geq 3$). □

We continue to prove the two lemmas that were used in the above proof. We adopt the notation from the proof. In particular, for a block $G'[B]$ of $G'[S]$, the set L is the set of vertices in B that have two neighbors in B, and $H = B\backslash L$ is the set of vertices with at least three neighbors in B. The first lemma gives a lower bound for the number of L-vertices in a block of $G'[S]$.

Lemma 5.10 *Let CD-set S and graph G' have the properties stated in Theorem 5.8. For every block $G'[B]$ in $G'[S]$, $|L| \geq |H|$.*

If $G'[B]$ contains no triangles, then $|L| \geq |H| + 1$.

Proof: Let $G'' = G'[B]$. Consider two neighbors $u, v \in H \cap B$. If $d_{G'}(u) \geq 4$, then $d_{G'}(v) = 3 = d_{G''}(v)$ since uv is not a bridge of $G'[S]$ (Property 2 from Theorem 5.8). Then v is not a dominator or connector since all its neighbors are in B. This is a contradiction with Property 4. So both u and v have degree three in G', and therefore are not dominators or connectors. So by Property 5, they form a 2-cut in $G'[S]$, and therefore also in G''.

We show that it follows that every vertex in H has at most one neighbor in H. Suppose $u \in H$ has two neighbors v and w in H. $G'' - u - v$ is disconnected, and u and v both have exactly one neighbor in both components. Therefore if u and v share a neighbor, this must be a vertex in L since G'' is 2-connected. So w is not a neighbor of v. Now it can be checked that $G'' - u - w$ is still connected, a contradiction with the statement from the previous paragraph.

So every vertex in H has at most one neighbor in H, and therefore at least two neighbors in L. Clearly, every vertex in L has at most two neighbors in H. It follows that $|L| \geq |H|$.

In addition, if $|L| = |H|$, then every vertex in H has exactly one neighbor in H, and every vertex in L has two neighbors in H. We show that it follows that G'' contains a triangle. Consider two neighbors $u, v \in H$. Let C_1 and C_2 be the two components of $G'' - u - v$, such that $|V(C_1)| \leq |V(C_2)|$. If C_1 contains vertices of H, then let u' and v' be two neighbors in $H \cap V(C_1)$. Let C_1' and C_2' be the components of $G'' - u' - v'$, such that $\{u, v\} \subseteq V(C_2')$. Now $V(C_2) \subset V(C_2')$, so $|V(C_1')| < |V(C_1)|$. We can continue to find new neighbor pairs in H like this, until the smallest component contains no vertices of H anymore. Since every vertex in L has two neighbors in H, this component consists of a single vertex, and we have found a triangle. □

In the proof of Theorem 5.9 we started with a certain weight assignment and showed that in $G'[S]$: a degree two vertex has weight at least $\frac{1}{2}$, a degree one vertex has weight at least $\frac{3}{2}$, and for a block B and cut vertex set C (with respect to $G'[S]$), $w(B \backslash C) \geq \frac{1}{2}|B| + 2 - |C \cap B|$. We anticipated in the proof of Theorem 5.9 that this yields that the total weight of S is at least $\frac{1}{2}|S| + 2$, so slightly more than $\frac{1}{2}$ on average.

The proof of the following lemma shows our counting method: leaves of $G'[S]$ contain excess weight, and blocks of $G'[S]$ have a weight deficit. The proof shows how weights are moved inwards along the branches, and how the weight deficit for blocks (given by the $|C \cap B|$ term) becomes an excess when all but one of the branches surrounding a block are dealt with this way. See Figure 5.3 for an example illustrating this reassignment of weights.

Lemma 5.11 *Let $G = (V, E) \neq K_1$ be a connected graph with non-negative weights w on the vertices such that:*

1. *If $d(v) = 2$ then $w(v) \geq \frac{1}{2}$.*
2. *If $d(v) = 1$ then then $w(v) \geq \frac{3}{2}$.*
3. *If $G[B]$ is a block of G, and C is the set of cut vertices of G, then $w(B \backslash C) \geq \frac{1}{2}|B| + 2 - |C \cap B|$.*

For this graph, $w(V) \geq \frac{1}{2}|V| + 2$.

Proof: We prove the statement by induction on $|V|$. If G is 2-connected, then the statement follows immediately from Property 3. If $G = K_2$ the statement is clearly true.

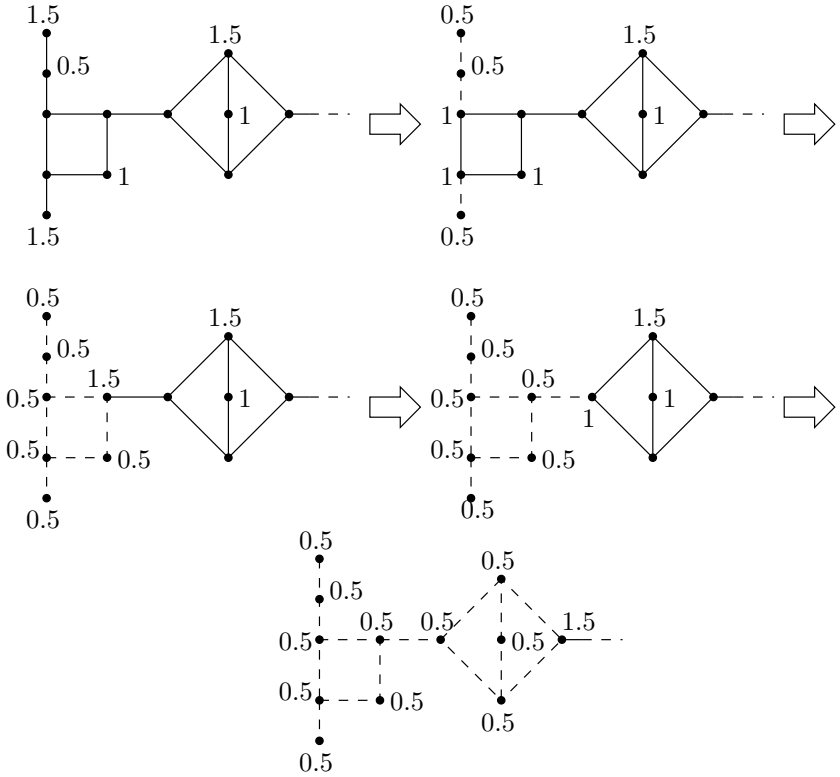


Figure 5.3: Moving the weights in $G'[S]$

In any other case, G has at least one leaf u such that $G-u$ is not K_1, or it has at least one block with exactly one cut vertex (Lemma 5.5). We consider these two cases.

Suppose first that u is a leaf of G, with neighbor v. Consider the graph $G' = G - u$ with weight function $w'(v) = w(v) + w(u) - \frac{1}{2}$, and $w'(x) = w(x)$ for all other vertices. We prove that for G' and w', the three properties hold again, so we can use induction. Note that since $w(u) \geq \frac{3}{2}$, $w'(v) \geq w(v) + 1$.

If v has degree two in G' then since $w'(v) \geq 1$, the first property holds for v. If v has degree one in G', then $w(v) \geq \frac{1}{2}$ (since v has degree two in G), so $w'(v) \geq \frac{3}{2}$, and the second property holds for v. For all other vertices, the weights and degrees do not change so these two properties hold for all vertices.

For a block $G'[B]$, it is obvious that Property 3 still holds, unless $v \in B$ and v is not a cut vertex anymore in G'. In this case, let C_G resp. $C_{G'}$ denote the set of cut vertices of G and G' ($C_G = C_{G'} + v$).

$$w'(B \backslash C_{G'}) \geq w(B \backslash C_G) + 1 \geq \frac{1}{2}|B| - |C_G \cap B| + 3 = \frac{1}{2}|B| - |C_{G'} \cap B| + 2,$$

so Property 3 holds.

We conclude that if G has a leaf u, then we can construct a new graph $G' = G - u$ with weight function w' such that $w'(V(G')) = w(V(G)) - \frac{1}{2}$, for which the three properties hold. $G' \neq K_1$ and G' is connected, so by induction, $w(V(G)) = w'(V(G') + \frac{1}{2} = \frac{1}{2}|V(G')| + \frac{5}{2} = \frac{1}{2}|V(G)| + 2$, which proves the lemma for this case.

Now we consider the case that G has no leaves. Then G has a block B which contains only a single cut vertex u. Consider the graph $G' = G - (B - u)$ with weight function $w'(u) = \frac{3}{2}$ and $w'(x) = w(x)$ for all other vertices. $G' \neq K_1$, so if we can show that the three properties hold for G' and w', we can use induction.

If $d_{G'}(u) = 1$ or $d_{G'}(u) = 2$, then w' clearly satisfies the corresponding properties. For a block $G'[B']$, Property 3 clearly holds when $u \notin B'$ or u is still a cut vertex in G'. In the other case, let C_G resp. $C_{G'}$ denote the set of cut vertices of G and G'. Now

$$w'(B' \backslash C_{G'}) = w(B' \backslash C_G) + \frac{3}{2} \geq \frac{1}{2}|B'| - |C_G \cap B'| + \frac{7}{2} > \frac{1}{2}|B'| - |C_{G'} \cap B'| + \frac{5}{2},$$

which is even better than necessary.

Now for G' all weight properties are satisfied and we can use induction: let $V = V(G)$ and $V' = V(G')$.

$$w(V) \geq w'(V') + w(B - u) - \frac{3}{2} \geq (\frac{1}{2}|V'| + 2) + (\frac{1}{2}|B| + 1) - \frac{3}{2} =$$

$$\frac{1}{2}|V'| + \frac{1}{2}|B| + \frac{3}{2} = \frac{1}{2}(|V| - |B| + 1) + \frac{1}{2}|B| + \frac{3}{2} = \frac{1}{2}|V| + 2.$$

This proves the lemma for the case where G has no leaves, so the lemma is true in all cases. □

In Theorem 5.9, there is an additional condition that no block of $G'[S]$ contains triangles. Unfortunately, such a G' and S do not always exist for every connected graph G with $\delta(G) \geq 3$. Consider graphs with triangles consisting of three cut vertices: every standard CD-set and actually even every CD-set contains triangles. In fact, we can even construct planar, 3-connected, cubic graphs that do not have a CD-set without triangles:

Theorem 5.12 *Deciding whether a given graph has a CD-set without triangles is $\mathcal{NP}$-complete, even when restricted to planar, cubic, 3-connected graphs. The same is true for deciding whether a given graph has a potential standard CD-set without triangles.*

Proof: The problem of deciding whether a given graph has a spanning subgraph that is a path is called the Hamiltonian Path problem. This problem is $\mathcal{NP}$-complete even when restricted to planar, cubic, 3-connected graphs [31]. Let $G = (V, E)$ be such a Hamiltonian Path instance. Replacing every vertex with a triangle in the straightforward way gives a graph G', which is again planar, cubic and 3-connected. Suppose S is a CD-set without triangles for G'. Define $F \subset E$ as follows: if the edge corresponding to $uv \in E$ is part of $G'[S]$, add uv to F. Since S contains at most two vertices of every triangle in G', $\Delta((V, F)) \leq 2$. Suppose a vertex v with $d(v) = 0$ exists in (V, F). Then, because S is a dominating set, every neighbor of v has degree one in (V, F). So (V, F) has maximum degree two and at least three vertices with degree one, and therefore has multiple non-trivial components, contradicting the fact that $G'[S]$ is connected. Therefore (V, F) has minimum degree one and maximum degree two and is connected, which gives a Hamiltonian path for G. Similarly, a Hamiltonian path for G can be used to find a CD-set without triangles for G'. This shows that the first decision problem of Theorem 5.12 is $\mathcal{NP}$-complete.

In the graph constructed above, a CD-set without triangles exists if and only if a potential standard CD-set without triangles exists, so the second problem is also $\mathcal{NP}$-complete. □

So we cannot hope to improve our CD-set construction method to always find CD-sets that do not contain triangles. However, for graphs without triangles Theorem 5.8 and Theorem 5.9 immediately lead to the following positive statement:

Theorem 5.13 *A connected, triangle-free graph on n vertices with $\delta \geq 3$ has a CD-set S with $|S| \leq (2n - 4)/3$.*

This statement is equivalent with stating that a connected, triangle-free graph on n vertices with $\delta \geq 3$ has a spanning tree with at least $(n + 4)/3$ leaves (see Section 5.2.2), so we have proved Theorem 5.3. We have proved this by considering minimal CD-sets with a number of properties, and showing that for

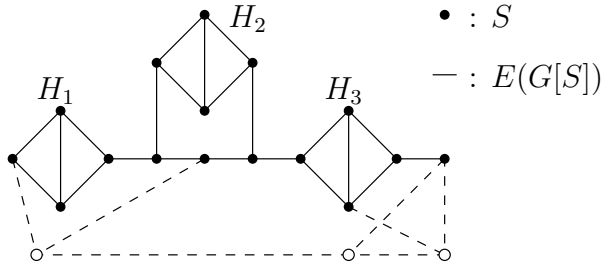


Figure 5.4: $G[S]$ contains one cubic diamond block

any such CD-set the bound holds. This differs from the techniques that were used previously to prove this kind of results.

For graphs with triangles, we use the CD-set construction method presented in the next section. The idea is to find a CD-set in which blocks and therefore blocks containing triangles are sparse. This leads to an upper bound for the CD-set size, and corresponding lower bound for the number of leaves of a spanning tree, that is better than the bound from Theorem 5.1 (except in some very special cases given by the worst case examples for that bound), but worse than the bound from Theorem 5.13.

5.5 Small CD-sets for graphs with few cubic diamonds

5.5.1 Construction and properties

The bound given in Theorem 5.1 is best possible for its class (connected graphs with $\delta \geq 3$). So if we want to improve the bound, we have to restrict the graph class. Fortunately, only a small restriction is needed.

Definition 5.14 *A subgraph H of G is a* cubic diamond *if H is a diamond that is induced by four vertices of degree three in G.*
If u and v are the two vertices in H that are not adjacent, it is called a cubic diamond between *u and v.*
Let $S \subseteq V(G)$. Subgraph H of $G[S]$ is a cubic diamond block *of $G[S]$ if H is a cubic diamond with respect to G and a block with respect to $G[S]$. (Vertices of H may have degree two in $G[S]$.)*

Figure 5.4 shows an illustration of cubic diamond blocks: H_1 is a cubic diamond block of $G[S]$ that contains one vertex with degree two in $G[S]$, but H_2 and H_3 are not cubic diamond blocks.

All of the examples showing that Theorem 5.1 is best possible contain many cubic diamonds. We will show that for graphs without cubic diamonds, the bound can be improved. Actually our main result is stronger: we give a new bound that depends on the number of cubic diamonds. The new bound is

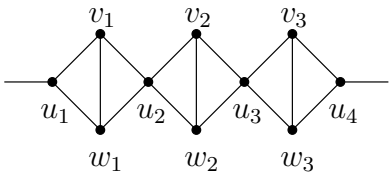


Figure 5.5: A diamond necklace

considerably better if this number is small, and it almost coincides with the bound from Theorem 5.1 when this number is maximum. The only difference is a discrepancy of $\frac{2}{7}$ in the constant.

Below we again prove the existence of a potential standard CD-set S for G and a realization G' that have a number of properties. The first five properties are the same as those stated in Theorem 5.8. In addition, since cubic diamonds are the only obstruction to an improved bound, we need to make sure that cubic diamonds in $G'[S]$ only occur when they cannot be avoided. This is the case when the graph G contains a *diamond necklace* (see Figure 5.5).

Definition 5.15 *An induced subgraph H of G is a* diamond necklace *in G if for some $k \geq 1$, the vertices of H can be labeled $u_1, \ldots, u_{k+1}$, $v_1, \ldots, v_k$, $w_1, \ldots, w_k$, such that the edge set of H consists exactly of u_iv_i, u_iw_i, v_iw_i, v_iu_{i+1}, w_iu_{i+1} for $i = 1, \ldots, k$, and such that $u_2, \ldots, u_k$ have degree four in G, and all other vertices of H have degree three in G.*

Note that a cubic diamond is also a diamond necklace. Property 6 in Theorem 5.18 below shows that in the CD-set we consider, the only cubic diamond blocks that occur are part of a diamond necklace. Before we can explain Property 7 of Theorem 5.18, we need the following definitions.

Definition 5.16 *A vertex $v \in S$ is a* simple CD-set vertex *with respect to the CD-set S for graph G if $d(v) = 3$ and*

- *$G[S - v]$ has two components, and $S - v$ is a dominating set for G, or*
- *$G[S - v]$ is connected, and v has exactly one 1-leaf neighbor.*

Definition 5.17 *A 1-leaf v is a* block-leaf *with respect to the CD-set S for graph G if there is a path P in $G[S]$ with end vertices x and y ($x = y$ is possible), consisting only of simple CD-set vertices, such that*

- *x is a neighbor of v.*
- *y is part of a block of $G[S]$, but none of the other vertices of P are part of a block.*

Sloppily speaking, in the proof of Theorem 5.9 and Lemma 5.11, a lower bound for $|\overline{S}|$ was found as follows. First weights were assigned to vertices

in S, mainly from leaves that distributed a weight of one equally among their neighbors in S. Then we showed that these weights can be redistributed over the vertices of S such that every vertex receives a weight of at least $\frac{1}{2}$.

If blocks of $G'[S]$ may contain triangles, then this is not always possible. However, for every block that is not a diamond, there is usually one additional block-leaf (there are exceptions to this statement, but those are easy to work with). If we can ensure that there are relatively few block-leaves (and thus that blocks are sparse), then we can change the weight assignment: every block-leaf will assign a weight of a little more than 1 to its CD-set neighbors, and other leaves will distribute a weight of a little less than 1 among its CD-set neighbors. This can be done such that the total weight is still at most $|\overline{S}|$. With this weight assignment, non-diamond blocks receive enough weight to prove the desired bound. Property 7 below shows that there are relatively few block-leaves.

Theorem 5.18 *A connected graph G with $\delta(G) \geq 3$ has a spanning subgraph G' and a CD-set S with the following properties. Degrees, neighbors and all other graph related notions are taken with respect to G'.*

1. *S is a standard CD-set (for G').*
2. *If $u, v \in S$, $d(u) \geq d(v) \geq 4$ and $uv \in E(G')$, then uv is a bridge of $G'[S]$.*
3. *Edges in $E(G) \backslash E(G')$ are between vertices in S, or between a vertex in S and a 2-leaf.*
4. *Every vertex $v \in S$ that is neither a dominator nor a connector has at least three neighbors in S, and all of its neighbors in S have degree three.*
5. *If $\{u, v\} \subseteq S$, $uv \in E(G')$ and u and v are both neither dominators nor connectors, then $G'[S] - u - v$ is not connected.*
6. *The number of cubic diamond blocks in $G'[S]$ is at most the number of diamond necklaces in G.*
7. *The number of block-leaves is at most half the total number of leaves.*

In addition, if $|S| = 2$, then $|\overline{S}| \geq 4$.

Proof: We consider a CD-set S and spanning subgraph G' such that S is a standard CD-set for G', that are optimal according to the following priorities:

- Minimize $|S|$.
- Maximize the number of 2-leaves in $\overline{S}$.
- Minimize the number of cubic diamond blocks in $G'[S]$.
- Minimize $|E(G'[S])|$.
- Maximize $|E(G')|$.

By this we mean that among all pairs of S and G' that minimize $|S|$, we consider pairs that maximize the number of 2-leaves. Among all these pairs, we choose one that minimizes the number of cubic diamond blocks in $G'[S]$, etc. We show that for S and G' chosen this way, the above properties hold. The proof is by contradiction: we consider a standard CD-set S for G'. If one of the properties does not hold, we find an improved S and G' according to the priorities.

Property 2: If $u, v \in S$, $d(u) \geq d(v) \geq 4$ and $uv \in E(G')$, then uv is a bridge of $G'[S]$.
Proof: Consider $u, v \in S$ with $d(u) \geq 4$, $d(v) \geq 4$ and $uv \in E(G')$. If uv is not a bridge in $G'[S]$, then we can delete uv such that S still is a standard CD-set for G'. Since this decreases $|E(G'[S])|$, this change is an improvement unless it introduces a new cubic diamond block D. In the latter case, assume w.l.o.g. $v \in V(D)$, and let $z \notin \{u, v\}$ be a vertex in D adjacent to the three other vertices of D. Consider $S - z$ and $G' - vz$ (with $uv \in E(G')$). This is again a standard CD-set, so an improvement is found.

The proofs of Property 4 and Property 5 are exactly the same as in the proof of Theorem 5.8, and the proof of Property 3 is very similar. We leave the details to the reader.

Property 6: The number of cubic diamond blocks in $G'[S]$ is at most the number of diamond necklaces in G.
Proof: We show that every cubic diamond block in $G'[S]$ is part of a diamond necklace in G, and that every diamond necklace in G contains only one cubic diamond block of $G'[S]$.

First we define an operation on a standard CD-set S for subgraph G' of G, which changes S and G'. This operation is illustrated in Figure 5.6. Let D be a cubic diamond block in $G'[S]$. Edges in $E(G)\backslash E(G')$ will be called *removed edges*. Suppose a removed edge xy exists with $x \in V(D)$ and y a 2-leaf. A *2-leaf change* using xy consists of the following steps: let $w \in V(D) - x$ be a vertex with three neighbors in $V(D)$. Add y to S, remove w from S, add xy to $E(G')$, and remove wx from $E(G')$. If the addition of y to S introduces a 3-leaf z, then in addition remove yz from $E(G')$, for every such leaf z. Since y now has three neighbors in S, this last operation does not reduce its degree below three. After this change, x has degree three and w is a 2-leaf. It follows that the result is a standard CD-set S^* for the new graph G^* and that D is not a block in $G^*[S^*]$. Note that $G^*[S^*]$ may contain a cubic diamond block that was not present in $G'[S]$, as illustrated in Figure 5.6(b).

Using 2-leaf changes, we will prove the statements from the beginning of the proof. Let $D_1 = G'[\{u_1, v_1, w_1, u_2\}]$ be a cubic diamond block in $G'[S]$, with $u_1u_2 \notin E(G')$. If D_1 is also a cubic diamond in G, the statements are obvious. Otherwise, there is a removed edge xy such that $x \in V(D_1)$. If $y \in S$, then add xy to G'. S is again a standard CD-set with respect to G', and $G'[S]$ contains one cubic diamond block less. This is an improvement. So we may assume

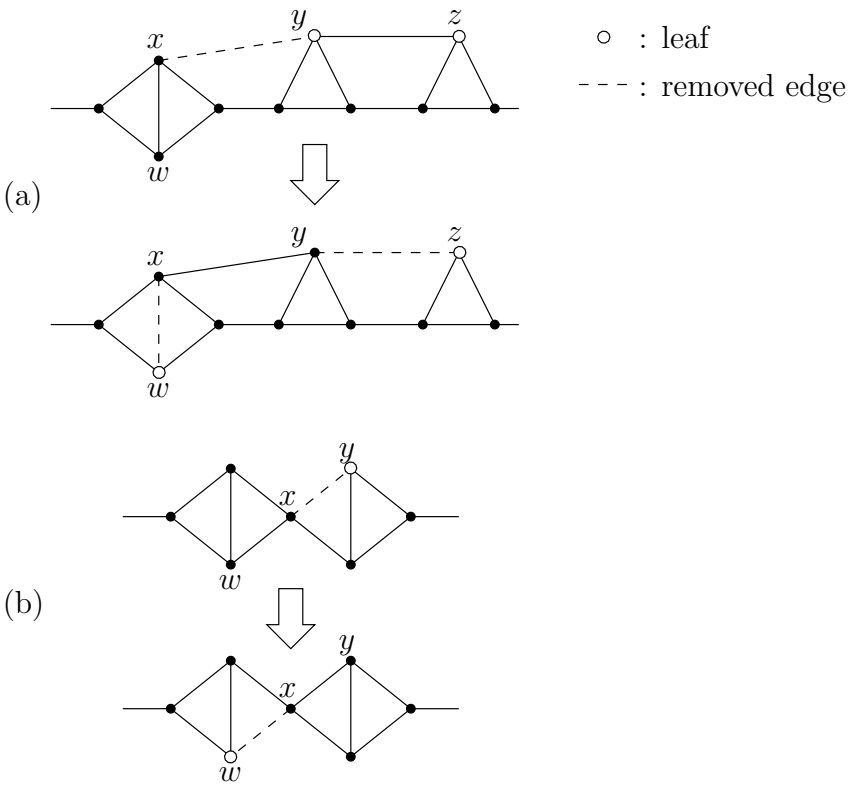


Figure 5.6: Two examples of a 2-leaf change

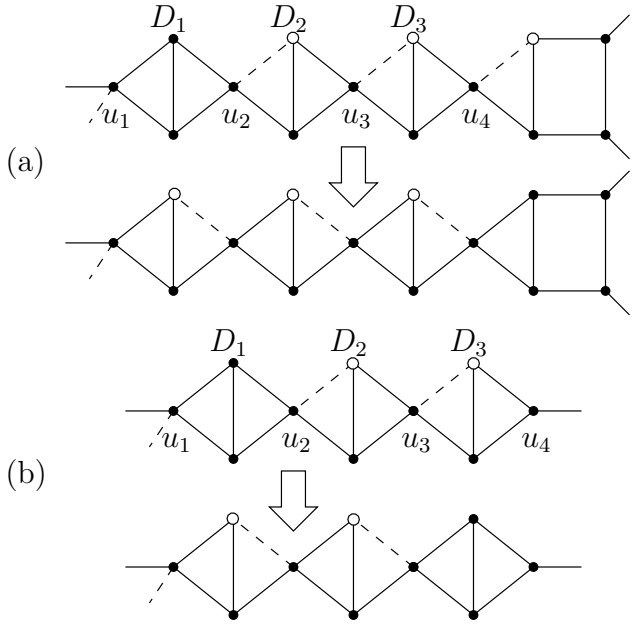


Figure 5.7: Two possible results of a series of 2-leaf changes

that $y \notin S$. Then y is a 2-leaf (Property 3). We can apply a 2-leaf change using the edge xy. If the number of cubic diamond blocks decreases, this is an improvement (as indicated in Figure 5.6(a)). Otherwise, x and y are part of a new cubic diamond block D_2. In this case, D_1 and D_2 only have x in common (the other two remaining vertices of D_1 cannot be part of a cubic diamond block since they are now adjacent to a 2-leaf), and w.l.o.g. $x = u_2$ (otherwise x would have degree at least four, and D_2 would not be cubic). This case is shown in Figure 5.6(b). We label the vertices of D_2 as $V(D_2) = \{u_2, v_2, w_2, v_3\}$, with $v_2v_3 \notin E(G')$.

If a 2-leaf change is possible for D_1, we first try to apply a 2-leaf change that decreases the number of cubic diamond blocks. If this is not possible, we apply an other 2-leaf change. So in this case, v_1 and w_1 are not incident with removed edges. We also know that w.l.o.g. u_2 is incident with exactly one removed edge (otherwise the neighbor of u_2 in $V(G')\backslash V(D_1)$ would have at least two 2-leaf neighbors and therefore would have degree at least four), and u_1 is incident with at most one removed edge. We apply a 2-leaf change using the removed edge incident with u_2, which gives a new cubic diamond block D_2, with $V(D_2) = \{u_2, v_2, w_2, u_3\}$ such that u_2 and u_3 are not adjacent. Now we try to find an improvement again: if an edge can be added such that D_2 is not a cubic diamond block anymore, or if a 2-leaf change can be applied such that the number of cubic diamond blocks decreases, we have found such an improvement. Otherwise, if a 2-leaf change is possible that does not introduce D_1 again, we apply this 2-leaf change introducing a cubic diamond block D_3 with $V(D_3) =$

$\{u_3, v_3, w_3, u_4\}$. We continue to apply 2-leaf changes (without returning to a previous state) until either an improvement is found (see Figure 5.7(a)), or no new 2-leaf change can be applied anymore. In the latter case, we have found k diamonds in G labeled $D_1, \ldots, D_k$ such that $V(D_i) = \{u_i, v_i, w_i, u_{i+1}\}$, where u_i and u_{i+1} are not adjacent, and we know that $d_G(v_i) = d_G(w_i) = 3$ for all i, $d_G(u_{k+1}) = 3$, $d_G(u_i) = 4$ for $i = 2, \ldots, k$ (see Figure 5.7(b)). If $d_G(u_1) = 3$, then we have found a diamond necklace in G which contains only one cubic diamond block of $G'[S]$. If $d_G(u_1) > 3$, then we consider the original G' and S again, and apply the same strategy starting with a 2-leaf change using an edge incident with u_1. If this also does not lead to an improvement, a structure like the one shown in Figure 5.7(b) is present on both sides of D_1, and we have again found a diamond necklace in G.

In every case we have either found an improvement, or have shown that the cubic diamond block is part of a cubic diamond necklace in G that contains only one cubic diamond block. This concludes the proof of Property 6.

Property 7: The number of block-leaves is at most half the total number of leaves.
Proof: We will show that if there are leaves with at least two block-leaf neighbors, we can make an improvement according to our priorities, except in one very specific case. Combined with the fact that 1-leaves have at least two leaf neighbors (Property 3), this enables us to prove that there are at least as many non-block-leaves as block-leaves in S and G' (a *non-block-leaf* is a leaf but not a block-leaf).

Below we consider a number of cases corresponding to structures in S and G'. In every case we will define a new pair S^* (or S_2) and G^* such that S^* is a standard CD-set for subgraph G^* of G, and S^*, G^* is an improvement. We use the following shorthand to indicate the four features we will check for S^* and G^*:

Connected: S^* induces a connected graph.
Dominating: S^* is a dominating set.
2-CD-set: There are only 1-leaves and 2-leaves with respect to S^* and G^*.
Minimum degree 3: Vertices in S^* have degree at least three in G^*.
Improvement: In every case we can show that $|S^*| < |S|$ or $|S^*| = |S|$ and S^* has more 2-leaves than S.

First we study these cases separately, and afterwards we use these cases to prove Property 7.

Suppose there is a leaf u with two block-leaf neighbors v and w. Let x and y be the CD-set neighbors of v resp. w. Consider $S^* = S + u - x - y$.

Case 1: S^* is again a CD-set.
We will show to construct graph G^*. It is clear that this change is an improvement.
2-CD-set: If there is a 2-leaf a with respect to S that becomes a 3-leaf because of the addition of u to the CD-set, then $d(u) > 3$, since v, w and at least one

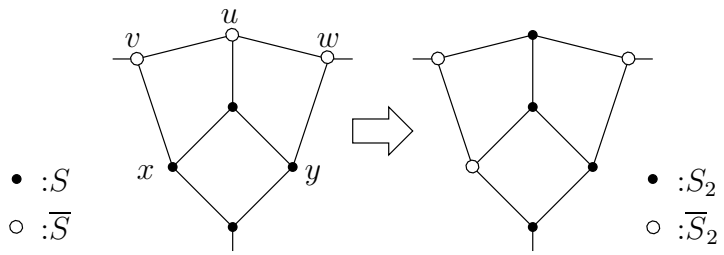


Figure 5.8: The change from case 2.1

CD-set vertex are also neighbors of u. So the edge au can be deleted from G' without decreasing the degree of a CD-set vertex below three. This holds for every such edge. Since v and w are block-leaves for S, $d(x) = d(y) = 3$, and they both have at least one leaf neighbor with respect to S^*, so the two new leaves are also not 3-leaves.
Minimum degree 3: Only vertex u with $d(u) \geq 3$ was added to the CD-set.

Case 2: $G'[S^*]$ is not connected. We consider two subcases which cover this case.

Case 2.1: $G'[S^*]$ and $G'[S - x - y]$ are not connected.
Since v and w are block-leaves and x and y are dominators, x and y cannot be connectors. So x and y form a minimal 2-cut for $G'[S]$. So both have at least two CD-set neighbors. Consider $S_2 = S + u - x$ (See Figure 5.8 for an example).
Connected: x is not a connector, so $G'[S - x]$ is connected. Since v is a block-leaf, x has degree three, and thus u is not adjacent to x. So $G'[S_2]$ is also connected.
Dominating: Since v is a block-leaf, only v is not dominated by $S - x$. u is a neighbor of v, so S_2 is a dominating set.
2-CD-set: If a 2-leaf becomes a 3-leaf because of the addition of u, we can delete an edge from G' (as in case 1). x becomes a 2-leaf.
Minimum degree 3: u has degree at least three.
Improvement: $|S_2| = |S|$. We gain a 2-leaf (x), and lose a leaf that may have been a 1-leaf or 2-leaf (u). v remains a 1-leaf, but with a new CD-set neighbor. w was a 1-leaf but becomes a 2-leaf after the addition of u. x has degree three, so it has no leaf neighbors other than v, and therefore no 2-leaves can become 1-leaves. It is possible that some other 1-leaves become 2-leaves after the addition of u. We see that there is at least one more 2-leaf.

Case 2.2: $G'[S^*]$ is not connected, but $G'[S - x - y]$ is connected.
In this case, u is an isolated vertex in $G'[S^*]$. This means that in S its CD-set neighbors are exactly x and y (not just one of them, since v and w are block-leaves). Consider $S_2 = S + u - x$ (See Figure 5.9).
Connected: x is not a connector, so $G'[S - x]$ is connected. u has y as a

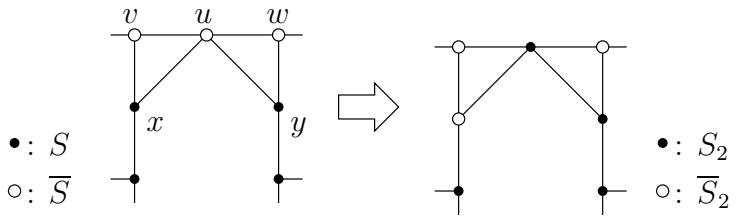


Figure 5.9: The change from case 2.2

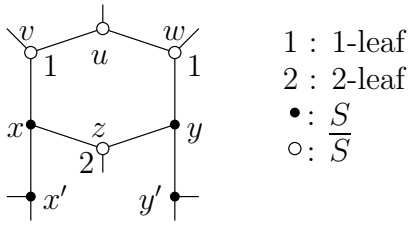


Figure 5.10: The situation and labeling in case 3

CD-set neighbor, so $G'[S_2]$ is connected.
Dominating: Only v is not dominated by $S - x$. v is adjacent to u, so S_2 is dominating.
2-CD-set: If the addition of u makes 2-leaves into 3-leaves, we can again delete edges from G' to prevent this (as in case 1). x has degree three and one leaf neighbor (v), so it is not a 3-leaf.
Minimum degree 3: u has degree at least four.
Improvement: $|S_2| = |S|$. We gain a 2-leaf (x), and lose a 2-leaf (u). v remains a 1-leaf, with a new CD-set neighbor. w was a 1-leaf but becomes a 2-leaf after the addition of u. x has degree three, so it has no leaf neighbors other than v, and therefore no 2-leaves can become 1-leaves. It is possible that some other 1-leaves become 2-leaves after the addition of u. We see that there is at least one more 2-leaf.

Case 3: Suppose S^* is not a dominating set.
This case is more complicated than the previous cases. We study the situation in more detail, and introduce some additional vertex labels, and then proceed to study six subcases. See Figure 5.10. The only 1-leaf neighbors of x and y are v resp. w, so there is a 2-leaf (with respect to S) that has CD-set neighbors x and y. Let z be this 2-leaf. $d(z) = 2$ only if in G, z is adjacent to another vertex $a \in S$ (Property 3). In this case, we can add the edge az to G', and find that there is a graph for which S^* is a standard CD-set (see case 1 for details). So we may now assume that z has at least one leaf neighbor. Since S^* is not dominating, this leaf neighbor is not equal to u, but can be equal to v or w. x and y have degree one in $G'[S]$. Let x' and y' be their respective CD-set

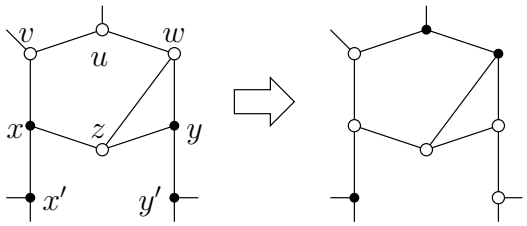


Figure 5.11: The change from case 3.1

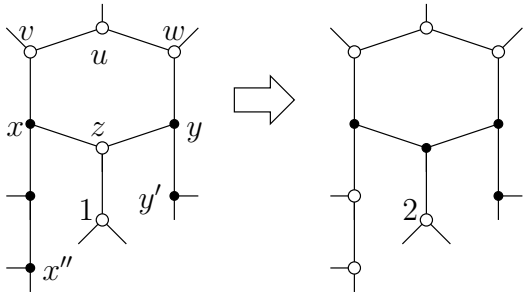


Figure 5.12: The change from case 3.2

neighbors. $x' \neq y'$ since v and w are block-leaves.

Case 3.1: Suppose z is adjacent to w. Now consider $S_2 := S+u+w-x-y-y'$ (See Figure 5.11).
Connected: $G'[S-x]$ and $G'[S-y-y']$ are connected since v and w are block-leaves. x is not part of a block (of $G'[S]$), so $G'[S-x-y-y']$ is also connected. y' is not a dominator, so u has a neighbor in S not in $\{x, y, y'\}$. v is adjacent to u. So $G'[S_2]$ is connected.
Dominating: The only vertices not dominated by $S-x-y-y'$ are y, v, w and z, since y' is not a dominator. u dominates v, w is in S_2, and w dominates z and y.
2-CD-set: z becomes a 1-leaf, so u and w both are adjacent to at least three vertices that will not become 3-leaves. So if for S_2 there would be 3-leaves adjacent to u or w, we can prevent this by deleting the corresponding edges, without decreasing the degrees of u and w below three. The new leaves have degree three and are adjacent to other leaves.
Minimum degree 3: u and w have degree at least three.
Improvement: $|S_2| < |S|$.

Case 3.2: Suppose the following properties hold:

- z has a 1-leaf neighbor.
- x' is not part of a block (of $G'[S]$), so it has one other simple CD-set

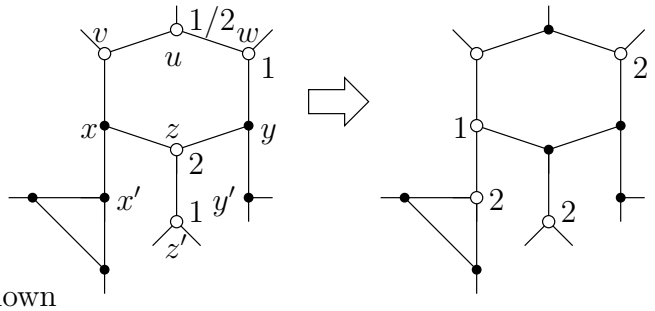


Figure 5.13: The change from case 3.3

neighbor x''.

Consider $S_2 = S + z - x' - x''$ (See Figure 5.12). If x' and x'' share a 2-leaf neighbor, then x'' is not part of a block and has one other simple CD-set neighbor x''' and we consider $S_2 = S + z - x'' - x'''$ instead.
Connected: $G'[S - x' - x'']$ has two components, one consisting of the single vertex x. z is adjacent to y and x, so $G[S_2]$ is connected. In the case where we consider $G'[S - x'' - x''']$, one component consists of vertices x and x', and the reasoning is the same.
Dominating: x' and x'' (or x'' and x''') are not dominators, and do not share a 2-leaf neighbor, and both have neighbors in $S - x' - x''$ ($S - x'' - x'''$), so $S - x' - x''$ ($S - x'' - x'''$) is already a dominating set.
2-CD-set: z has at least three neighbors that are not 2-leaves with respect to S, so if 3-leaves would be introduced by the addition of z, we can prevent this by deleting the corresponding edges. x' and x'' (x'' and x''') have degree three and are adjacent, so these will not become 3-leaves.
Minimum degree 3: z has degree at least three.
Improvement: $|S_2| < |S|$.

Case 3.3: Suppose the following properties hold:

- z has a 1-leaf neighbor $z' \neq v$, $z' \neq w$.
- x' is part of a block.

Consider $S_2 := S + u + z - x - x'$ (See Figure 5.13).
Connected: $G'[S - x - x']$ is connected. Since x' is not a dominator and $N(x) = \{v, z, x'\}$, u has a CD-set neighbor other than x' or x. z is adjacent to y. So $G'[S_2]$ is connected.
Dominating: Only v and x are not dominated by $S - x - x'$. u dominates v, and z dominates x.
2-CD-set: x and x' do not become 3-leaves since they have degree three. u is adjacent to at least three vertices that will not become 3-leaves. z is adjacent to at least three vertices that will not become 3-leaves.

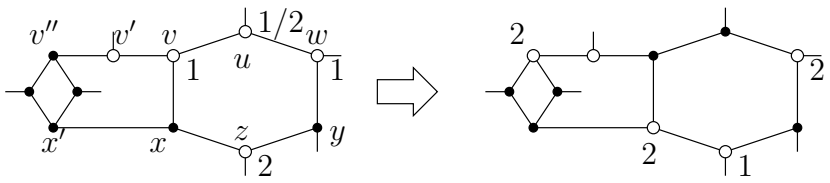


Figure 5.14: The change from case 3.4

Minimum degree 3: u and z have degree at least three.
Improvement: $|S_2| = |S|$. We lose u which can be a 1-leaf or 2-leaf, and we lose the 2-leaf z. We gain a 2-leaf x' (since it is part of a block) and a 1-leaf x. w becomes a 2-leaf, v remains a 1-leaf (with a new CD-set neighbor), and z' becomes a 2-leaf. Since x and x' have no other leaf neighbors in $\overline{S}$, there are no other 2-leaves that can become 1-leaves. It is possible that some 1-leaves become 2-leaves by the addition of u and z. So we gain at least one 2-leaf.

Case 3.4: Suppose the following properties hold:

- v has a neighbor $v' \neq u$ that is a block-leaf. Let v'' be its CD-set neighbor.
- x' and v'' are part of the same block.

Note that v' cannot be equal to w, because in that case $v'' = y$ and is not part of a block. Consider $S_2 = S + u + v - v'' - x$ (See Figure 5.14).
Connected: $G'[S - x]$ and $G'[S - v'']$ are connected since v resp. v' are block leaves. Since x is not part of a block, $G'[S - x - v'']$ is also connected. u has a CD-set neighbor other than x or v'', and v is adjacent to u.
Dominating: In $S - x - v''$, only v and v' are not dominated, since x and v'' do not share a 2-leaf neighbor. $v \in S_2$, and v' is dominated by v.
2-CD-set: u and v have at least three non-2-leaf neighbors, so any 3-leaves adjacent to u or v can be prevented by deleting edges from G'. v'' and x have degree three and have at least one leaf neighbor in $\overline{S}_2$.
Minimum degree 3: u and v have degree at least three.
Improvement: $|S_2| = |S|$. We lose the 1-leaf v and the leaf u which can be a 1-leaf or a 2-leaf. v'' is part of a block so it becomes a 2-leaf. x becomes a 2-leaf since $v \in S_2$. z becomes a 1-leaf instead of a 2-leaf, v' remains a 1-leaf (with a new CD-set neighbor), and w becomes a 2-leaf instead of a 1-leaf. In S there are no 2-leaves adjacent to v'' since $d(v'') = 3$ and it is part of a block, so the removal from the CD-set of v'' cannot make 2-leaves into 1-leaves. Something similar is true for x. The addition of u and v can make some 1-leaves into 2-leaves. So we gain at least one 2-leaf.

Case 3.5: Suppose the following properties hold:

- v has a neighbor $v' \neq u$, $v' \neq w$ that is a block-leaf.

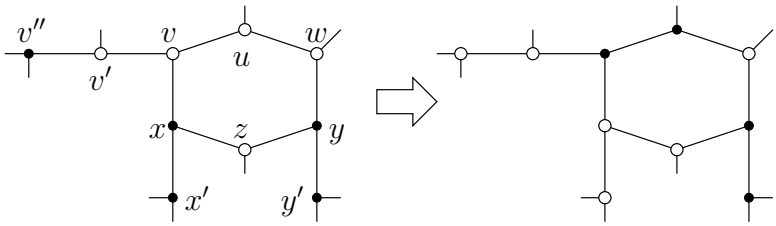


Figure 5.15: The change from case 3.5

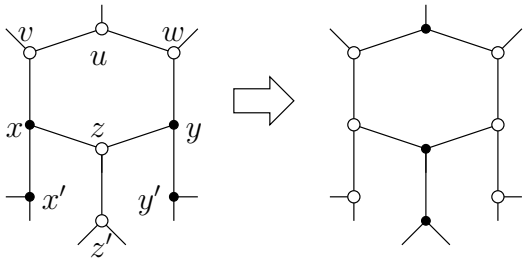


Figure 5.16: The change from case 3.6

- Let v'' be the unique CD-set neighbor of v'. v'' and x' are not part of the same block, and do not share a 2-leaf neighbor.

Consider $S_2 = S + u + v - x - x' - v''$ (See Figure 5.15).
Connected: Since v is a block-leaf, $G'[S - x - x']$ is connected. Since v' is a block-leaf, $G'[S - v'']$ is connected. In addition, since v'' and x' are not part of the same block, $G'[S - x - x' - v'']$ is connected. x' and v'' do not share a 2-leaf neighbor, so u has a CD-set neighbor other than x, x' or v''. v is adjacent to u. So $G'[S_2]$ is connected.
Dominating: The only vertices not dominated by $S - x - x' - v''$ are v, v' and x, since v and v' are the only 1-leaves adjacent to x resp. v'', x' is not a dominator, and x' and v'' do not share a 2-leaf neighbor. $v \in S_2$, and v' and x are dominated by v.
2-CD-set: x, x' and v'' all have degree three and have at least one leaf neighbor with respect to S_2. Both u and v have at least three neighbors that will not become 3-leaves.
Minimum degree 3: u and v have degree at least three.
Improvement: $|S_2| < |S|$.

Case 3.6: Suppose the following properties hold:

- x' and y' are not part of the same block and do not share a 2-leaf neighbor.

- z has a neighbor z' that is a 2-leaf.

Consider $S_2 = S + u + z + z' - x - x' - y - y'$ (See Figure 5.16).
Connected: Since v and w are block leaves, $G'[S-x-x']$ resp. $G'[S-y-y']$ are connected. Since x' and y' are not part of the same block, $G'[S-x-x'-y-y']$ is also connected. Since x' and y' do not share a 2-leaf neighbor, z' has a CD-set neighbor other than x', y', x and y. The same is true for u. z is adjacent to z'. So $G'[S_2]$ is connected.
Dominating: The only vertices not dominated by $S-x-y$ are v, w and z. x' and y' are not dominators and do not share a 2-leaf (also not with x and y), so the only additional vertices not dominated by $S-x-y-x'-y'$ are x and y. In S_2, v and w are dominated by u, z is part of S_2, and x and y are dominated by z.
2-CD-set: The new leaves have degree three and are adjacent to at least one leaf. u, z and z' have at least three neighbors that will not become 3-leaves.
Minimum degree 3: u, z and z' have degree at least three.
Improvement: $|S_2| < |S|$.

Unfortunately, the above cases do not cover all possibilities, so we cannot show that in S and G', every leaf has at most one block-leaf neighbor. However we can use the above cases to show that if a leaf has at least two block-leaf neighbors, it is part of a highly restricted structure. We will deduce a number of properties for this situation, and use this to conclude our proof of Property 7. For this we introduce some definitions and notations.

A cycle C in G' is called a *problem cycle* if its vertices can be labeled u, v, w, x, y and z as in Figure 5.10, such that u is a leaf with two block-leaf neighbors v and w, v and w have CD-set neighbors x and y, and x and y have the 2-leaf neighbor z in common. For any problem cycle C, we use the notation $u(C)$ to denote the vertex that is labeled u in such a labeling. We will also call $u(C)$ the *u-vertex of* C. For the other five vertices the notation is similar. Note that there is only one vertex in a problem cycle that can receive the label u, and the same holds for the label z, but the labels v and w are interchangeable, just like the labels x and y (though x and v should always be adjacent).

We will now prove that if a leaf u has at least two block-leaf neighbors v and w, then there is a problem cycle C with $u = u(C)$, $v = v(C)$ and $w = w(C)$. Then we will state a number of properties for problem cycles.

Consider a leaf u with at least two block-leaf neighbors v and w, which have CD-set neighbors x resp. y. If $S^* = S + u - x - y$ is again a CD-set, apply Case 1 to obtain a contradiction. If S^* is not connected, apply Case 2.1 or 2.2. If S^* is not dominating, then in Case 3 it is shown that these five vertices are part of a problem cycle C together with a vertex $z = z(C)$, with $d(z) \geq 3$. This covers all cases, so if a leaf u has block-leaf neighbors v and w with CD-set neighbors x resp. y, then $u = u(C)$, $v = v(C)$, $w = w(C)$, $x = x(C)$ and $y = y(C)$ for some problem cycle C. In this case we can define vertices x' and y' as it is done in Case 3, and use the subcases of Case 3 to deduce properties of C. These properties are stated in the following claims.

1. z has no 1-leaf neighbor, and has at least one 2-leaf neighbor.

Proof: If z has a 1-leaf neighbor, this cannot be u since then $S+u-x-y$ is a CD-set. If z is adjacent to w or v, we can apply Case 3.1 (using the symmetry of the cases). Otherwise z is adjacent to a 1-leaf $z' \notin V(C)$, and we can apply Case 3.2 or Case 3.3. Since $d(z) \geq 3$, z has a 2-leaf neighbor.

2. If v or w has a block leaf neighbor v', then $v' \in \{u, v, w\}$.
Proof: We prove the statement for v; for w it follows by symmetry. Suppose v has a block-leaf neighbor $v' \notin \{u, w\}$, with CD-set neighbor v''. If x' and v'' are part of the same block, we can apply Case 3.4. If x' and v'' are not part of the same block and do not share a 2-leaf neighbor, we can apply Case 3.5. So we now assume that x' and v'' share a 2-leaf neighbor. Then x' is not part of a block, and does not share a 2-leaf neighbor with y'. We know that z has a 2-leaf neighbor (Claim 1), so Case 3.6 can be applied.

In addition we state two claims for problem cycles that have vertices in common. A non-block-leaf is called *class 0* if it has no block-leaf neighbors, *class 1* if it has one block-leaf neighbor, and *class 2* if it has at least two block-leaf neighbors.

3. If C_1 and C_2 are problem cycles with $u(C_1) = u(C_2)$, then $C_1 = C_2$.
Proof: If $v(C_1) \in V(C_2)$ or $w(C_1) \in V(C_2)$, then w.l.o.g. $v(C_1) = v(C_2)$. Since $v(C_1)$ has exactly one CD-set neighbor, $x(C_1) = x(C_2)$. Since $v(C_1)$ is a block-leaf, $x(C_1)$ has only one other leaf neighbor, so $z(C_1) = z(C_2)$. $z(C_1)$ is a 2-leaf, so $y(C_1) = y(C_2)$. $y(C_1)$ is adjacent to only one block-leaf, so $w(C_1) = w(C_2)$. It follows that $C_1 = C_2$. We conclude that it is not possible that there are two different problem cycles C_1 and C_2 such that $u(C_1) = u(C_2)$ and $v(C_1)$ or $w(C_1)$ is equal to $v(C_2)$ or $w(C_2)$. Now if $u(C_1) = u(C_2)$ and the two cycles do not have any of these vertices in common, then we consider a new problem cycle C_3 that contains $u(C_1)$, $v(C_1)$ and $v(C_2)$ (since a leaf and two of its block-leaf neighbors are always part of a problem cycle). Now we have $u(C_3) = u(C_1)$ and $v(C_3) = v(C_1)$, but $C_1 \neq C_3$, a contradiction.

4. If the problem cycles C_1 and C_2 both contain a class 0 leaf z, then w.l.o.g. $v(C_1) = v(C_2)$, $x(C_1) = x(C_2)$, $z(C_1) = z(C_2)$, $y(C_1) = y(C_2)$ and $w(C_1) = w(C_2)$ (the u-vertices may be different).
Proof: The only class 0 leaf on a problem cycle can be the z-vertex, so $z = z(C_1) = z(C_2)$, and it must be a 2-leaf. So it has only two CD-set neighbors, and w.l.o.g. $x := x(C_1) = x(C_2)$ and $y := y(C_1) = y(C_2)$. Since their other neighbors on the cycles are block-leaves, x and y are only adjacent to one block-leaf, and $v(C_1) = v(C_2)$ and $w(C_1) = w(C_2)$.

For the purpose of proving that at most half of the leaves are block-leaves, we define a function f that assigns non-block-leaves to block-leaves. Using the above claims, we will prove that f is injective. The assignment is done using the following method which contains three assignment steps.

Consider a block-leaf v to which no non-block-leaf has been assigned.

1. If v has a class 1 neighbor u, then assign $f(v) := u$.

2. Otherwise, if v has a class 2 neighbor u, which has another block-leaf neighbor w, then let C be the problem cycle with $v = v(C)$, $u = u(C)$ and $w = w(C)$. Assign $f(v) := u$, $f(w) := z(C)$.

3. Otherwise, v has only block-leaf neighbors. Since $d(v) \geq 3$ (Property 3), $v = u(C)$ for some problem cycle C. Assign $f(v) := z(C)$.

Repeat this until all block-leaves are assigned a non-block-leaf. Clearly, one of the above steps applies for every block-leaf, so f is a function. We will prove that f is injective, so every non-block-leaf is assigned at most once.

Consider a non-block-leaf z. If z is class 1, then it is assigned at most once in step 1. All leaves assigned in steps 2 and 3 are class 0 (Claim 1) or class 2, so z is not assigned in these steps.

If z is class 2, then $z = u(C)$ for some problem cycle C. z is only part of one such problem cycle (Claim 3), so z is assigned at most once in step 2, and not in step 3 (step 3 only assigns class 0 vertices).

If z is class 0, then it can be assigned in step 2 or step 3. For every problem cycle C that z is part of, $v(C)$ and $w(C)$ are the same (Claim 4), so z is assigned at most once in step 2 (even though z can be part of multiple problem cycles, once the v and w-vertices are assigned a non-block-leaf, problem cycles with the same v and w-vertices are not considered anymore in step 2). Suppose z is assigned once in step 2 and once in step 3, or at least twice in step 3. Then $z = z(C_1) = z(C_2)$ for two different problem cycles C_1 and C_2, and w.l.o.g. $u(C_2)$ is a block-leaf (C_2 corresponds to the cycle used in step 3). C_1 and C_2 must also have the v, w, x and y-vertices in common (Claim 4). This means that $v(C_1)$ has a block-leaf neighbor $u(C_2) \notin V(C_1)$, a contradiction with Claim 2. We conclude that also in this case, z is assigned at most once.

We have shown that an injective function from the block-leaves to the non-block-leaves can be constructed. This shows that at most half of the leaves are block-leaves, and completes the proof of Property 7.

We complete the proof of Theorem 5.18 by proving the following statement.

If $|S| = 2$, then $|\overline{S}| \geq 4$.
Proof: Suppose $|S| = 2$. Since there are no connectors, both vertices in S are dominators, and there are at least two 1-leaves u and v. Since the vertices in S have degree at least three (Property 1), there is at least one other leaf w. If in addition to w there is at least one other leaf, we are done. Otherwise, w is a 2-leaf. Since u and v have degree at least three (Property 3), both u and v are also adjacent to w. We conclude that $\{w\}$ is a standard CD-set, which is an improvement. □

Analogously to the use of Theorem 5.8 and Theorem 5.9, now that the existence of a CD-set with the properties stated in Theorem 5.18 has been established,

we will use these properties to determine an upper bound on the size of such a CD-set. This will be done in the next section.

5.5.2 An upper bound for the size of the constructed CD-set

We introduce a notion that enables us to explore the existence of block-leaves when considering a weight assignment to $G'[S]$.

Definition 5.19 *A subgraph P of a graph G is a* block path *of G if it is a path with end vertices l and b such that $d(l) = 1$, $d(b) = 3$, b is part of a block of G, and all internal vertices of P have degree two in G. For such a path P, l_P denotes the end vertex with degree one in G, and b_P denotes the end vertex with degree three in G.*

Note that block paths contain at least two vertices. The relation between block-leaves and block paths is as follows. For a CD-set S of graph G', every block-leaf v is adjacent to either a vertex in a block of $G'[S]$, or the end vertex l_P of a block path P in $G'[S]$. Every block path in $G'[S]$ is adjacent to at most one block-leaf.

The next lemma is a variant of Lemma 5.11 that can be used for $G'[S]$ when S and G' have the properties stated in Theorem 5.18, and when triangles are allowed. We can now only prove that there are roughly $\frac{2}{5}$ leaves for every CD-set vertex, instead of roughly $\frac{1}{2}$. The first three properties in Lemma 5.20 then correspond directly to the first three properties in Lemma 5.11, although the block property we need is more sophisticated. When combined with the weight assignment we use in Theorem 5.21, the additional path property states that every block path either ends in a block-leaf from which it receives extra weight, or if it does not end in a block-leaf, it receives extra weight in another way.

Lemma 5.20 *Let $G = (V, E)$ be a connected non-tree graph with non-negative weights w on the vertices such that:*

1. *If $d(v) = 2$ then $w(v) \geq \frac{2}{5}$.*
2. *If $d(v) = 1$ then $w(v) \geq \frac{6}{5}$.*
3. **(Block property)** *Consider a block $G[B]$ of G. Let C denote the set of cut vertices of G, and let L denote the vertices in B that have two neighbors in B, and $H = B \backslash L$. Then $w(B \backslash C) \geq \frac{2}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| + \frac{12}{5}$.*
4. **(Path property)** *If P is a block path in G, then $w(P) \geq \frac{2}{5}|V(P)| + \frac{4}{5}$.*

For this graph, $w(V) \geq \frac{2}{5}|V| + \frac{12}{5}$.

Proof: If G is 2-connected, then the statement follows immediately from the inequality for blocks. For the other cases we will construct a smaller graph G' with weight function w', and use induction on $|V|$. For this we need to prove the four properties for G' and w'.

First, a remark on proving the path property for G' and w'. Suppose Property 1 and 2 of the lemma hold for G' and w'. Let P be a block path in G'. Note these two properties already imply that $w'(P) \geq \frac{2}{5}(|V(P)| - 2) + \frac{6}{5} = \frac{2}{5}|V(P)| + \frac{2}{5}$. So in order to prove that the path property holds for P, it suffices to show that $w'(l_P) \geq \frac{8}{5}$, or $w'(b_P) \geq \frac{2}{5}$, or $w'(x) \geq \frac{4}{5}$ for some internal vertex x of P.

If G is not 2-connected, it has at least one leaf or at least one block with exactly one cut vertex (Lemma 5.5). We distinguish these two cases.

Let u be a leaf of G, with neighbor v. Consider the graph $G' = G - u$ with weight function $w'(v) = w(v)+w(u)-\frac{2}{5}$, and $w'(x) = w(x)$ for all other vertices. We prove that G' and w' satisfy the four properties of the lemma. Note that since $w(u) \geq \frac{6}{5}$, $w'(v) \geq w(v) + \frac{4}{5}$.

If v has degree two in G' then since $w'(v) \geq \frac{4}{5}$, the first property holds for v. If v has degree one in G', then $w(v) \geq \frac{2}{5}$ (since v has degree two in G), so $w'(v) \geq \frac{6}{5}$, and the second property holds for v. For all other vertices, the weights and degrees do not change so the first two properties hold for all vertices.

For a block $G'[B]$, it is obvious that the block property still holds, unless $v \in B$ and v is not a cut vertex anymore in G'. In this case, let C_G resp. $C_{G'}$ denote the set of cut vertices of G and G' ($C_G = C_{G'} + v$), and let L denote the set of vertices in B with two neighbors in B, and $H = B \backslash L$ (H and L are the same in G and G'). If $v \in H$, then

$$w'(B\backslash C_{G'}) \geq w(B\backslash C_G) + \frac{4}{5} \geq$$

$$\frac{2}{5}|B| - \frac{6}{5}|C_G \cap L| - \frac{4}{5}|C_G \cap H| + \frac{16}{5} =$$

$$\frac{2}{5}|B| - \frac{6}{5}|C_{G'} \cap L| - \frac{4}{5}|C_{G'} \cap H| + \frac{12}{5}.$$

If $v \in L$, then v has degree two in G' and degree three in G, so $v = b_P$ for a block path P in G on vertices u and v. Therefore $w'(v) \geq \frac{2}{5}|V(P)| + \frac{4}{5} - \frac{2}{5} = \frac{6}{5}$, thus $w'(B\backslash C_{G'}) \geq w(B\backslash C_G) + \frac{6}{5}$, and the same reasoning as above shows that the block inequality holds again.

Now suppose v is part of a block path P' in G'. If $v = P'_l$, then v is an internal vertex of a block path P of G, with $V(P) = V(P') + u$. In that case, $w'(P') = w(P) - \frac{2}{5} = \frac{2}{5}|V(P)| + \frac{4}{5} - \frac{2}{5} = \frac{2}{5}|V(P')| + \frac{4}{5}$, so the path property holds for P'. If $v = P'_b$ or v is an internal vertex of P', then the path property holds for P' since the first two properties hold for G', and $w'(v) \geq \frac{4}{5}$. All other block paths in G' correspond to block paths in G with the same weights.

We conclude that if G has a leaf u, then we can construct a new graph $G' = G - u$ with weight function w' such that $w'(V(G')) = w(V(G)) - \frac{2}{5}$, for which the four properties hold. G' is not a tree since G is not a tree, so we can use induction. By induction, $w(V(G)) = w'(V(G') + \frac{2}{5} = \frac{2}{5}|V(G')| + \frac{12}{5} + \frac{2}{5} = \frac{2}{5}|V(G)| + \frac{12}{5}$, which proves the lemma for this case.

Now we consider the case that G has no leaves. Then G has a block B which contains only a single cut vertex u. Consider the graph $G' = G - (B - u)$ with weight function $w'(u) = \frac{8}{5}$ and $w'(x) = w(x)$ for all other vertices. Since G is not 2-connected and has no leaves, it has at least two blocks (Lemma 5.5), and therefore G' is not a tree. We will show that the four properties hold for G' and w', and use induction.

If $d_{G'}(u) = 1$ or $d_{G'}(u) = 2$, then w' clearly satisfies the corresponding properties. Since $w'(u) = \frac{8}{5}$, a block path in G' that contains u satisfies the path property. For a block $G'[B]$, the block property clearly holds when $u \notin B$ or u is still a cut vertex in G'. In the other case, let C_G resp. $C_{G'}$ denote the set of cut vertices of G and G', and let L denote the set of vertices in B with two neighbors in B, and $H = B \backslash L$. Now

$$w'(B \backslash C_{G'}) = w(B \backslash C_G) + \frac{8}{5} \geq$$

$$\frac{2}{5}|B| - \frac{6}{5}|C_G \cap L| - \frac{4}{5}|C_G \cap H| + \frac{20}{5} >$$

$$\frac{2}{5}|B| - \frac{6}{5}|C_{G'} \cap L| - \frac{4}{5}|C_{G'} \cap H| + \frac{12}{5}.$$

We conclude that G' and w' satisfy all properties of the lemma. Since G' is not a tree, we can use induction. Let $V = V(G)$ and $V' = V(G')$.

$$w(V) \geq w'(V') + w(B - u) - \frac{8}{5} \geq (\frac{2}{5}|V'| + \frac{12}{5}) + (\frac{2}{5}|B| + \frac{6}{5}) - \frac{8}{5} =$$

$$\frac{2}{5}|V'| + \frac{2}{5}|B| + \frac{10}{5} = \frac{2}{5}(|V| - |B| + 1) + \frac{2}{5}|B| + \frac{10}{5} = \frac{2}{5}|V| + \frac{12}{5}.$$

This proves the statement for the case that G has no leaves, which completes the proof of Lemma 5.20. □

Using the previous weight counting lemma, we can prove the following upper bound for the size of a minimal CD-set $S' \subseteq S$.

Theorem 5.21 *Let G' and S be a graph and a CD-set satisfying the properties from Theorem 5.18. If $S' \subseteq S$ is a minimal CD-set, then $|S'| \leq (5|V(G)| - 12 + D)/7$, where D is the number of cubic diamond blocks in $G'[S]$.*

Proof: In order to prove this theorem, we assign weights w to the vertices of S such that the total weight is at most $|V \backslash S| + \frac{7}{5}|S \backslash S'| + D/5$, and then show that $w(S) \geq \frac{2}{5}|S| + \frac{12}{5}$. Then we can combine the two inequalities to obtain the result.

The weight assignment is as follows: block-leaves assign a weight of $\frac{6}{5}$ to their CD-set neighbor, and all other leaves distribute a weight of $\frac{4}{5}$ equally among their neighbors in S' (vertices in $S \backslash S'$ receive no weight yet). Since at most half of the leaves are block-leaves (Property 7 of Theorem 5.18), we have assigned

a total weight of not more than $|V \backslash S|$. Next, we assign a weight of $\frac{7}{5}$ to each vertex in $S \backslash S'$. Vertices in $S \backslash S'$ that are part of a cubic diamond block of $G'[S]$ receive an additional weight of $\frac{1}{5}$.

We prove that $G'[S]$ and this weight assignment w satisfy the four properties of Lemma 5.20.

1. A vertex v with degree two in $G'[S]$ is a dominator or connector (Property 4 of Theorem 5.18), so $v \in S'$. v has at least one leaf neighbor u (Property 1), and u has at most two neighbors in $G'[S]$, so it assigns at least $\frac{2}{5}$ to v. This shows that $w(v) \geq \frac{2}{5}$.

2. A vertex v with degree one in $G'[S]$ must be a dominator, since it cannot be a connector, and it must be a connector or dominator (Property 4). From the 1-leaf adjacent to it, it gets a weight of at least $\frac{4}{5}$. In addition, since v has degree at least three and leaves have at most two CD-set neighbors, it gains an additional weight of at least $\frac{2}{5}$. So $w(v) \geq \frac{6}{5}$.

3. If $G'[B]$ is a block of $G'[S]$, and L is the set of vertices in B with two neighbors in B and H is the set of vertices in B with at least three neighbors in B, then $|L| \geq |H|$ (Lemma 5.10). Let C denote the set of connectors. A vertex in L that is not a connector must be a dominator (Property 4). A dominator in L has weight at least $\frac{6}{5}$, since either its 1-leaf neighbor is a block-leaf, or it has degree at least four and receives weight at least $\frac{4}{5} + \frac{2}{5}$. We consider two cases:

   (a) Suppose $|(S \backslash S') \cap B| \geq 1$. Note that this implies that $|H| \geq 1$ (Property 4). Then one of the vertices in H has an additional weight of at least $\frac{7}{5}$. Now if $|B| \geq 5$ then $w(B \backslash C) \geq \frac{6}{5}|L \backslash C| + \frac{7}{5} \geq \frac{3}{5}|B| - \frac{6}{5}|C \cap L| + \frac{7}{5} \geq \frac{2}{5}|B| - \frac{6}{5}|C \cap L| + \frac{12}{5} \geq \frac{2}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| + \frac{12}{5}$. If $|B| < 5$, then $G'[B]$ is a diamond, since $|H| \geq 1$. In this case, let $v \in (S \backslash S') \cap B$. v is not a connector or dominator in S. Therefore, all its neighbors have degree three in G'. So there is one other vertex in B that is not a connector or dominator, and therefore v also has degree three. We conclude that B is a cubic diamond block, so the additional weight of v is $\frac{8}{5}$, and $w(B \backslash C) \geq \frac{6}{5}|L \backslash C| + \frac{8}{5} \geq \frac{3}{5}|B| - \frac{6}{5}|C \cap L| + \frac{8}{5} = \frac{2}{5}|B| - \frac{6}{5}|C \cap L| + \frac{12}{5} \geq \frac{2}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| + \frac{12}{5}$.

   (b) Suppose $(S \backslash S') \cap B = \emptyset$. Suppose B contains a vertex v that is not a dominator or connector with respect to S. Since $v \in S'$, it is a connector or dominator with respect to S'. Since $(S \backslash S') \cap B = \emptyset$, it is not a connector. So in S, v is adjacent to a 2-leaf u, and the other CD-set neighbor w of u is in $S \backslash S'$. In this case, v receives a weight of $\frac{4}{5}$. So vertices in $H \backslash C$ receive at least $\frac{4}{5}$, and vertices in $L \backslash C$ receive at least $\frac{6}{5}$.

   If $|B| = 3$, then $L = B$, and $w(B \backslash C) \geq \frac{6}{5}|L \backslash C| = \frac{6}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| = \frac{2}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| + \frac{12}{5}$. Now we assume $|B| \geq 4$,

and also use $|L| + |H| = |B|$, $|L| \geq |B|/2$:

$$w(B \backslash C) \geq \frac{6}{5}|L \backslash C| + \frac{4}{5}|H \backslash C| =$$

$$\frac{6}{5}|L| + \frac{4}{5}|H| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| \geq \frac{2}{5}|L| + \frac{4}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| \geq$$

$$\frac{1}{5}|B| + \frac{4}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| \geq \frac{2}{5}|B| - \frac{6}{5}|C \cap L| - \frac{4}{5}|C \cap H| + \frac{12}{5}.$$

4. Let P be a block path in $G'[S]$. If l_P, b_P or an internal vertex x of P has degree at least four in G', then it can be checked that $w(l_P) \geq \frac{8}{5}$, $w(b_P) \geq \frac{2}{5}$ resp. $w(x) \geq \frac{4}{5}$. If l_P has two 1-leaf neighbors, then $w(l_P) \geq \frac{8}{5}$. If an internal vertex x is a dominator then $w(x) \geq \frac{4}{5}$. Otherwise, the 1-leaf neighbor of l_P is a block-leaf, and $w(l_P) \geq \frac{6}{5} + \frac{2}{5}$. Together with the first two properties above, this proves the path property.

If $G'[S]$ is not a tree, we can apply Lemma 5.20, which shows that $w(S) \geq \frac{2}{5}|S| + \frac{12}{5}$. It follows that

$$\frac{7}{5}|S \backslash S'| + D/5 + |V \backslash S| = w(S) \geq \frac{2}{5}|S| + \frac{12}{5} \Leftrightarrow$$

$$\frac{7}{5}|S| - \frac{7}{5}|S'| + D/5 + |V| - |S| \geq \frac{2}{5}|S| + \frac{12}{5} \Leftrightarrow$$

$$|V| + D/5 - \frac{12}{5} \geq \frac{7}{5}|S'| \Leftrightarrow |S'| \leq (5|V| - 12 + D)/7.$$

If $G'[S]$ is a tree, we prove the last inequality in another way. Since there are no block-leaves, we assign the weights as we did in the proof of Theorem 5.9: every leaf distributes a weight of 1 equally among its CD-set neighbors. Using the same reasoning as in the proof of Theorem 5.9, we see that

- If v has degree two in $G'[S]$, then $w(v) \geq 1/2$.
- If v has degree one in $G'[S]$, then $w(v) \geq 3/2$.

It is a standard exercise to show that the number of leaves in a tree (on at least two vertices) is at least $h + 2$, where h is the number of vertices in the tree with degree at least three. From this fact it follows that $w(S) \geq |S|/2 + 2$. If $|S| \geq 4$, then $|\overline{S}| = w(S) \geq |S|/2 + 2 \geq \frac{2}{5}|S| + \frac{12}{5}$. If $|S| = 3$, then $G'[S] = P_3$. There are at least two dominators in S, so there are at least two 1-leaves in $\overline{S}$. Since every vertex in S has degree at least three (Property 1), there are at least three edges between S and leaves other than the two aforementioned 1-leaves. Since there are no 3-leaves (Property 1), we conclude that there are at least four leaves. So $|\overline{S}| \geq 4 > \frac{2}{5}|S| + \frac{12}{5}$. If $|S| = 1$ then since the vertex in S has degree at least three, $|\overline{S}| \geq 3$, so $|\overline{S}| \geq 3 > \frac{2}{5}|S| + \frac{12}{5}$. Finally, if $|S| = 2$, then $|\overline{S}| \geq 4$ (Theorem 5.18), so $|\overline{S}| \geq 4 > \frac{2}{5}|S| + \frac{12}{5}$. So in all cases:

$$|\overline{S}| \geq \frac{2}{5}|S| + \frac{12}{5} \Leftrightarrow$$

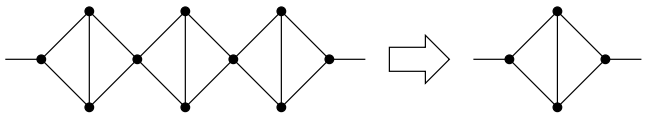


Figure 5.17: Reducing a necklace containing multiple diamonds

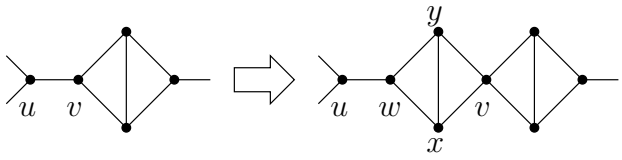


Figure 5.18: Reversing the necklace reduction

$$|V| - \frac{12}{5} \geq \frac{7}{5}|S| \Leftrightarrow |S| \leq \frac{5}{7}|V| - \frac{12}{7}.$$

Noting that $S' = S$ and $D = 0$ in this case, we obtain the desired inequality. □

In the bound from Theorem 5.21, the term D is equal to the number of cubic diamond blocks in $G'[S]$. Using Property 6 from Theorem 5.18, we know that for the S and G' we consider, all cubic diamond blocks are part of diamond necklaces in G, so the number of diamond necklaces is an upper bound for D. In the theorem below we show that diamond necklaces can be reduced when they consist of more than one diamond, and that therefore this term D can even be bounded by the number of cubic diamonds in G (diamond necklaces consisting of one diamond).

Theorem 5.22 *Let G be a connected graph on n vertices with $\delta \geq 3$. Then G has a CD-set S with $|S| \leq (5n - 12 + D)/7$, where D is the number of cubic diamonds in G that contain three vertices of S.*

Proof: First we reduce diamond necklaces that are not cubic diamonds in G: replace every diamond necklace containing at least two diamonds by a single cubic diamond as shown in Figure 5.17. These are called the new cubic diamonds, and the other cubic diamonds are called the original cubic diamonds. The resulting graph G_1 is again a simple connected graph with $\delta(G_1) \geq 3$.

Consider a CD-set S and subgraph G' of G_1 that satisfy the properties from Theorem 5.18. Let D and D' denote the number of original resp. new cubic diamonds of G_1 that are a block of $G'[S]$. Since all diamond necklaces are now cubic diamonds, the number of cubic diamond blocks in $G'[S]$ is $D + D'$ (Property 6). By Theorem 5.21, any minimal CD-set $S' \subseteq S$ has $|S'| \leq (5|V(G_1)| - 12 + D + D')/7$. Note that for any minimal CD-set $S' \subseteq S$, the cubic diamonds in G' that contain three vertices of S' are exactly those that are blocks in $G'[S]$.

From S' we can construct a CD-set for G: we can reconstruct G from G_1 by applying the operation illustrated in Figure 5.18 a number of times for every

new cubic diamond. This operation introduces three new vertices. At the same time, we can maintain a CD-set by adding at most two new vertices to the CD-set every time we apply the operation: if u and v in Figure 5.18 are not in the previous CD-set, then we add v and x. If one of u and v is in the previous CD-set, then we add w and x. Let k be the number of times this operation has to be applied in order to reconstruct G from G_1, and construct the corresponding CD-set S''. Using $k \geq D'$, we can bound $|S''|$ as follows:

$$|S''| \leq |S'| + 2k \leq (5|V(G_1)| - 12 + D + D')/7 + 2k \leq$$

$$(5(|V(G)| - 3k) - 12 + D + k)/7 + 2k = (5|V(G)| - 12 + D)/7.$$

The reconstruction does not affect the original cubic diamonds, and also does not affect which vertices of these diamonds are in the CD-set. So D is the number of cubic diamonds in G that contain three vertices of S''. □

Note that Theorem 5.4 from the introduction follows immediately from Theorem 5.22.

5.6 Worst case examples for the CD-set size

In this section we show that the bounds in Theorem 5.13 and Theorem 5.22 are best possible (in the same sense as explained in Section 6.1). Throughout this section, n denotes the number of vertices of G.

Theorem 5.13 shows that every connected, triangle-free graph on n vertices with $\delta \geq 3$ has a CD-set S with $|S| \leq (2n-4)/3$. We use examples very similar to those mentioned in [34] to show that this bound is best possible: for $n = 6k$, take k copies of the graph shown in Figure 5.19(a), and connect them in a cyclic form as shown in Figure 5.19(b). The resulting graph is cubic, connected and contains no triangles. It can be checked that it has no CD-set with fewer than $\frac{2}{3}n-2 = \lfloor(2n-4)/3\rfloor$ vertices. This shows that for a linear bound $\alpha n-\beta$, $\alpha = \frac{2}{3}$ is best possible. When we consider the graph Q_3 on eight vertices which has no CD-set with fewer than four vertices, we see that $\beta = \frac{4}{3}$ cannot be decreased.

Theorem 5.22 shows that every connected graph G with $\delta \geq 3$ has a CD-set S with $|S| \leq (5n - 12 + D)/7$, where D is the number of cubic diamonds in G. To prove that this bound is best possible we use the graph N_0 in Figure 5.20 as a building block, together with diamonds. For any D and $n = 4D + 7k$, take D diamonds and k copies of N_0 and connect them in a cycle. The resulting graph is connected, has $\delta = 3$, and has D cubic diamonds. It can be checked that it has no CD-set with fewer than $5k+3D-2 = \frac{5}{7}n+\frac{1}{7}D-2 = \lfloor(5n+D-12)/7\rfloor$ vertices. It follows that for a bound of the form $\alpha n + \gamma D - \beta$, only the choices $\alpha = \frac{5}{7}$ and $\gamma = \frac{1}{7}$ give an asymptotically sharp bound. Q_3 again shows that $\beta = \frac{12}{7}$ cannot be improved.

Just as for the bounds from Theorem 5.2 and 5.13, other than Q_3 we do not know any examples that show that β cannot be increased. However, for *every* $n \geq 4$ and D with $0 \leq 4D < n - 4$ we can construct examples with D

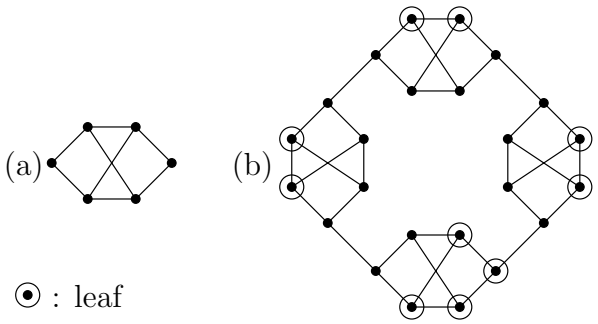


Figure 5.19: The bound of Theorem 5.13 is best possible

cubic diamonds that do not have a CD-set with fewer than $\lfloor (5n + D)/7 - 2 \rfloor$ vertices. For the construction of these examples, we take D diamonds, one copy of $N_{(n-4D) \bmod 7}$ (see Figure 5.20), and add copies of N_0 until we have exactly n vertices in total. These building blocks are again connected in a cycle. See Figure 5.21 for an example. The resulting graph has n vertices, is connected, has $\delta = 3$, has D cubic diamonds and has no CD-set with fewer than $\lfloor (5n+D)/7-2 \rfloor$ vertices. Note that this difference of $\frac{2}{7}$ in the constant is the same difference that prevents Theorem 5.22 (Theorem 5.4) from being a direct generalization of Theorem 5.1 (see Section 6.1).

5.7 An algorithmic viewpoint

It can be verified that the proofs of Theorem 5.8 and Theorem 5.18 are constructive: if we start with any potential standard CD-set for the graph G (this can be simply $V(G)$), we can apply the steps mentioned in the proofs until a CD-set S is obtained that satisfies the desired properties. For Theorem 5.8, these steps are adding and removing edges, removing a single vertex from the CD-set and possibly one incident edge, or removing a pair of vertices from the CD-set. In fact, for both proofs it can be checked that recognizing violations of the properties and applying the corresponding improvement steps (according to the three resp. five priorities mentioned in the proofs) can be done in polynomial time.

Such algorithms can be seen as local search algorithms, where the objective value is defined by the priorities stated in the proofs, and the neighborhood of a solution is defined by exactly those steps that are applied in the proofs.

However, implementing a local search algorithm using exactly these steps is not the most practical or most general way to program algorithms for this problem that guarantee the bounds from Theorem 5.9 and Theorem 5.21. In this section, we present short, practical, and stronger algorithms that generalize the methods from these proofs. These algorithms can again be seen as local search algorithms, with the same objective value as before, but this time the

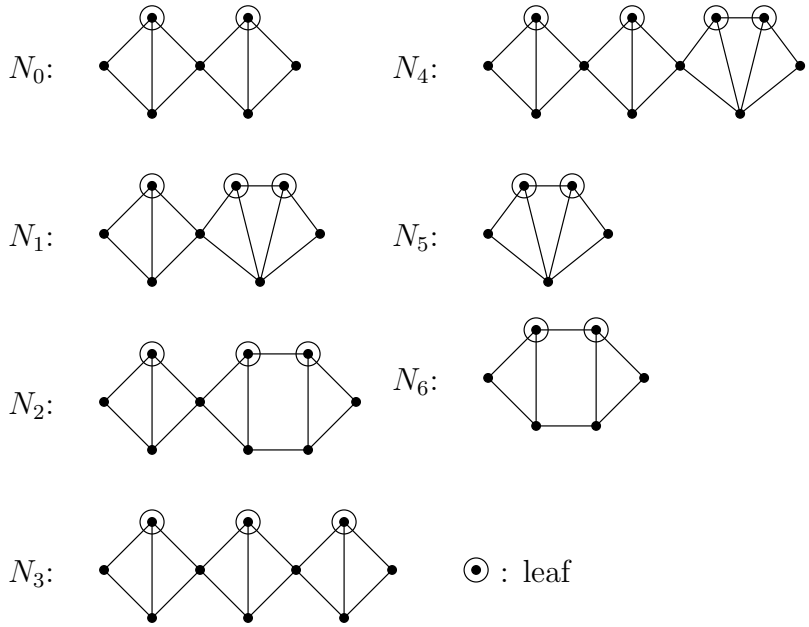


Figure 5.20: Building blocks for the worst case examples

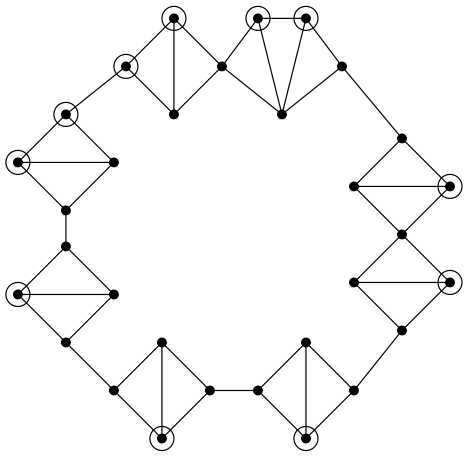


Figure 5.21: A worst case example with four cubic diamonds

neighborhood is defined more general (it contains the previous neighborhoods), and therefore the algorithms will give results that are at least as good, and probably better. Since the neighborhood is larger, the time complexity of the new algorithms is worse. For a practical implementation, the time complexity can be improved by making smarter choices of solutions to consider in the neighborhood, but we will not go into such details. We prefer to state the algorithms in a short and clear way, and leave the practical improvements to those that are interested.

We will consider potential standard CD-sets instead of standard CD-sets: for the first algorithm, this allows us to use a very 'linear' algorithm which only removes vertices from the CD-set until a minimal CD-set is obtained; for the second algorithm this enables us to ensure that the new neighborhood is of polynomial size. Before we can use potential standard CD-sets in the algorithm, we need the next lemma.

Lemma 5.23 *It can be checked in polynomial time whether $S \subseteq V(G)$ is a potential standard CD-set for G, and if so, a maximal realization G' of S can be found in polynomial time.*

Proof: We construct an auxiliary bipartite graph H with vertex set $A \cup B$: for every vertex $v \in S$, we add $d(v) - 3$ vertices to A (only vertices with degree at least four will correspond to vertices in A). For every i-leaf in $\overline{S}$ with $i > 2$, we add $i - 2$ vertices to B. We join vertices in A to vertices in B if in G edges exist between the corresponding vertices.

We show that a realization G' of S exists if and only if H has a matching M that saturates every vertex in B. If M is such a matching, delete every edge in G that corresponds to an edge of M, to obtain G'. For every former i-leaf (with $i > 2$), $i - 2$ incident edges are deleted since M saturates B, so only 1-leaves and 2-leaves remain. For every vertex in S with original degree d, at most $d - 3$ incident edges are deleted, so vertices in S still have degree at least three. Clearly, S is still a CD-set in G'.

Similarly, every maximal realization of S corresponds to a B-saturating matching in H.

The existence of polynomial time algorithms now follows from the well-known fact that polynomial time algorithms exist for finding a maximum matching (in fact, for bipartite graphs a specialized algorithm exists), and that if a B-saturating matching exists, every maximum matching saturates B. □

Algorithm 4 returns a CD-set that satisfies the bound from Theorem 5.13 when the input graph is triangle-free, but it is not required that the input graph is triangle-free. Using Lemma 5.23 we see that every step of the algorithm can be implemented in polynomial time. Since every step decreases the size of S or S', Algorithm 4 terminates in polynomial time. Note that for every graph G, $V(G)$ is a potential standard CD-set, so it is easy to find a correct input for the algorithm for every G. We prove that the output of this algorithm satisfies the bound from Theorem 5.13, regardless of the choice of input S.

Algorithm 4 The construction corresponding to Theorem 5.13

INPUT: A potential standard CD-set S for a connected graph G with $\delta(G) \geq 3$.

while there is a $U \subset S$ with $|U| \leq 2$ such that $S - U$ is a potential standard CD-set for G **do**
  $S := S - U$
end while
$S' := S$.
while there is a vertex $u \in S'$ such that $S' - u$ is a CD-set for G **do**
  $S' := S' - u$.
end while

Theorem 5.24 *For a connected, triangle-free graph G with $\delta(G) \geq 3$, and any potential standard CD-set S for G, Algorithm 4 returns a CD-set S' such that $|S'| \leq (2|V(G)| - 4)/3$.*

Proof: Let S and S' be the two CD-sets as they are after the algorithm has terminated. Choose a maximal realization G' for S, and in addition remove some edges from G': if $uv \in E(G')$, $\{u, v\} \subseteq S$ with $G'[S] - uv$ connected and $d_{G'}(u) \geq d_{G'}(v) \geq 4$, then delete uv from G'. Repeat this as long as such edges exist.

We prove that for these S and G', the properties from Theorem 5.8 are satisfied.

Property 2: If $u, v \in S$, $d(u) \geq d(v) \geq 4$ and $uv \in E(G')$, then uv is a bridge of $G'[S]$.
Proof: If uv is not a bridge, it would have been deleted from G' in the above step.

Property 3: Edges in $E(G) \backslash E(G')$ are between vertices in S, or between a vertex in S and a 2-leaf.
Proof: Let G'' be the maximal realization from which G' is obtained by deleting some edges between vertices in S. If an edge $e \in E(G) \backslash E(G')$ has no end vertices in S, or is incident with a 1-leaf, then also $e \notin E(G'')$. This is a contradiction with the fact that G'' is a maximal realization.

Property 4: Every vertex $v \in S$ that is neither a dominator nor a connector has at least three neighbors in S, and all of its neighbors in S have degree three.
Proof: Property 2 holds, so the same reasoning as in the proof of Theorem 5.8 applies. Note that in that proof, the only possible improvement to S that is considered consists of removing one vertex from S. This change is also considered in Algorithm 4.

Property 5: If $\{u, v\} \subseteq S$, $uv \in E(G')$ and u and v are both neither dominators nor connectors, then $G'[S] - u - v$ is not connected.

Proof: Since Property 4 holds, we can again apply the reasoning from the proof of Theorem 5.8. The improvement considered there consists of removing two vertices from S, which is also considered in Algorithm 4.

Since these properties hold and G contains no triangles, Theorem 5.9 shows that for any minimal CD-set $S^* \subseteq S$, S^* satisfies the bound. By the final step of the algorithm, S' is a minimal CD-set with $S' \subseteq S$. □

We next present an algorithm that can be used to find a CD-set S' that satisfies the bound from Theorem 5.22. We first point out a potential difficulty we have to cope with, and explain why we use potential standard CD-sets in the algorithm. In the proof of Property 6 from Theorem 5.18, the 'diamond necklace-like' structures can be arbitrarily long. So if we view a pair of a potential standard CD-set S and corresponding realization G' as a solution, there is no fixed upper bound on the number of edges that need to be changed in G' when going from one solution to the next. Therefore, without any additional rules on what edge sets to consider, we would have to consider an exponential size neighborhood. We want to avoid such additional rules since we want to state a simple, insightful formulation of the algorithm. Therefore we consider potential standard CD-sets instead. Another advantage is that this way, we consider a larger neighborhood, and therefore have a stronger algorithm. A disadvantage of using potential standard CD-sets is that for comparing two potential standard CD-sets S and S_2, we need to consider values that depend on (arbitrarily chosen) realizations. The following definition shows which values we take into account when evaluating such S and S_2.

Definition 5.25 *Let S and S_2 be potential standard CD-sets for graph G, and let G' resp. G'_2 be realizations for these CD-sets. We write $(S_2, G'_2) \prec (S, G')$ if*

- *$|S_2| < |S|$, or if*
- *$|S_2| = |S|$ and S_2 has fewer 1-leaves than S, or if*
- *$|S_2| = |S|$, S_2 and S have the same number of 1-leaves, and $G'_2[S_2]$ contains fewer cubic diamond blocks than $G'[S]$.*

Note that the number of 1-leaves is the same for every maximal realization. Unfortunately this is not true for the number of cubic diamond blocks. However, in the proof of Lemma 5.26 we show that for the final outcome, it does not matter that we choose arbitrary realizations in the algorithm.

Algorithm 5 is the main part of the algorithm we use to find a CD-set satisfying the properties from Theorem 5.18. Note again that input $S = V(G)$ and $G' = G$ can be chosen for any graph G. Formulated this way, the algorithm is not very fast because of the number of choices of U and W that are considered in each iteration. But we note that in every iteration, most of the possible choices of U and W that are considered are easily seen to be useless or redundant, so a large gain can be made here. See the proof of Theorem 5.18 for some ideas on

Algorithm 5 The construction of a CD-set corresponding to Theorem 5.18

INPUT: A potential standard CD-set S and a maximal realization G', for a connected graph G with $\delta(G) \geq 3$.

for all $U \subseteq \overline{S}$ and $W \subseteq S$ with $|U| \leq 3$ and $|U| \leq |W| \leq \min\{|U|+2, 4\}$ **do**
  $S_2 := (S \cup U) \backslash W$
  if S_2 is a potential standard CD-set **then**
    Find a maximal realization G'_2 of S_2.
    if $(S_2, G'_2) \prec (S, G')$ **then**
      Repeat the algorithm with input S_2, G'_2 and G.
    end if
  end if
end for
$S' := S$.
while there is a vertex $u \in S'$ such that $S' - u$ is a CD-set for G **do**
  $S' := S' - u$.
end while

useful choices of U and W. However, we preferred a short and clear formulation over speed, and therefore did not add additional rules for the choice of U and W.

Lemma 5.26 *For a connected graph G with $\delta(G) \geq 3$ and any potential standard CD-set and realization as input, Algorithm 5 finds in polynomial time a potential standard CD-set S, which has a realization G^* such that S and G^* satisfy the properties stated in Theorem 5.18.*

Proof: Let potential standard CD-set S and maximal realization G' be the S and G' as they are when the algorithm has terminated. In addition we consider another realization G^* for S that is obtained from G' by deleting some edges: start with $G^* = G'$. If $uv \in E(G^*)$, $\{u, v\} \subseteq S$ with $G^*[S] - uv$ connected and $d_{G^*}(u) \geq d_{G^*}(v) \geq 4$, then delete uv from G^*. Repeat this as long as such edges exist.

We prove that for the resulting S and G^*, the properties from Theorem 5.18 are satisfied. We omit the proof of the first properties, as they can be proved in exactly the same way as in the proof of Theorem 5.24.

Property 6: The number of cubic diamond blocks in $G^*[S]$ is at most the number of diamond necklaces in G.
Proof: Before we prove this property, we will prove two other statements. These are about the number of cubic diamond blocks in a realization of a given CD-set S. Observe that different (maximal) realizations of S can have a different number of cubic diamond blocks.

Claim 1: If a potential standard CD-set S has two maximal realizations G'

and G'_2 such that the number of cubic diamond blocks in $G'[S]$ and $G'_2[S]$ differs, then there is a vertex u such that $S-u$ is again a potential standard CD-set.

Since G' and G'_2 are maximal, $G'[S] = G'_2[S]$. Let D be a cubic diamond block in $G'[S]$ that is not a cubic diamond block in $G'_2[S]$. Since D is a diamond and a block in both, there is a $v \in V(D)$ with $d_{G'_2}(v) \geq 4$. Let $u \in V(D) - v$ be adjacent to all other vertices in D. Now $S - u$ is a standard CD-set for $G'_2 - uv$ (u becomes a 2-leaf, and v still has degree at least three. u is not a connector or dominator in any maximal realization).

Claim 2: Let G' and G'_2 be two realizations for S such that $G'_2 = G' - e$ for some edge e. If $G'[S]$ and $G'_2[S]$ have a different number of cubic diamond blocks, then there are vertices u or u and v such that $S - u$ resp. $S - u - v$ is again a potential standard CD-set.

First observe that since G' and G'_2 are both realizations of S, any cubic diamond block in $G'[S]$ is a cubic diamond block in $G'_2[S]$. So if the number of cubic diamond blocks differs, then there is a cubic diamond block D in $G'_2[S]$ that is not a cubic diamond block in $G'[S]$. If vertices of D are incident with e, then there is a vertex $v \in V(D)$ with $d_{G'}(v) \geq 4$. Let $u \in V(D) - v$ be adjacent to all other vertices in D. In this case, $S - u$ is a standard CD-set for $G' - uv$ (see Claim 1). Now suppose no vertices of D are incident with e. In this case, D is a cubic diamond in $G'[S]$, but not a block. Let u and v be the two vertices in D that are adjacent to all other vertices of D. $S - u - v$ is a standard CD-set for G' (u and v become 2-leaves, and since D was not a block and $d_{G'}(u) = d_{G'}(v) = 3$, $S - u - v$ is a CD-set).

Using these two claims we prove Property 6. Let S and G' be the S and G' as the are when Algorithm 5 has terminated (G' is a maximal realization for S). Let G^* again be a realization for S obtained from G' by deleting edges until Property 2 is satisfied. Suppose that in $G^*[S]$, cubic diamond blocks exist that are not part of a diamond necklace in G, or multiple cubic diamond blocks are part of the same cubic diamond necklace. We show that this leads to a contradiction with the fact that S is the output from algorithm 5.

In the proof of Theorem 5.18 it is shown that in this case a new potential standard CD-set $S_2 = S - x + y$ and realization G^*_2 exist such that $G^*_2[S_2]$ has fewer cubic diamonds than $G^*[S]$, and S_2 has the same number of 1-leaves. Note that even though this change from S to S_2 is described as a series of changes, the end result differs only in one vertex. In Algorithm 5, S_2 is considered. Let G'_2 be the maximal realization for S_2 that is considered in the algorithm. Finally, let G''_2 be a maximal realization of S_2 of which G^*_2 is a subgraph. (G''_2 and G'_2 do not have to be the same.) We will show that an improvement exists that is considered in the algorithm, regardless of the arbitrary choices of G' and G'_2, which is a contradiction.

Realization G^*_2 for S_2 is a (spanning) subgraph of maximal realization G''_2. Therefore if $G^*_2[S_2]$ and $G''_2[S_2]$ have a different number of cubic diamond blocks,

Claim 2 shows that an improvement $S_2 - u$ or $S_2 - u - v$ exists. These can be written as $S - x + y - u$ resp. $S - x + y - u - v$, so these improvements are considered in the algorithm, a contradiction. So now we may assume that $G_2^*[S_2]$ and $G_2''[S_2]$ have the same number of cubic diamond blocks. G_2' and G_2'' are both maximal realizations of S_2. If $G_2'[S_2]$ and $G_2''[S_2]$ have a different number of cubic diamond blocks, Claim 1 shows that an improvement $S_2 - u$ exists. The set $S_2 - u = S - x + y - u$ is considered in the algorithm, a contradiction. We conclude that $G_2'[S_2]$ also has the same number of cubic diamond blocks as $G_2^*[S_2]$. Realization G^* of S is a subgraph of maximal realization G', so similar reasoning as previously shows that $G'[S]$ and $G^*[S]$ have the same number of cubic diamond blocks. Since $G_2^*[S_2]$ has fewer cubic diamonds than $G^*[S]$, $G_2'[S_2]$ has fewer cubic diamonds than $G'[S]$, and therefore the algorithm finds an improvement when S_2 and G_2' are compared with S and G', a contradiction.

In every case an improvement is found. We conclude that every cubic diamond block of $G^*[S]$ is part of a diamond necklace of G, and that every diamond necklace of G contains at most one cubic diamond block of $G^*[S]$, so Property 6 holds for S and G^*.

Property 7: The number of block-leaves is at most half the total number of leaves.
The improvement steps used in cases 1, 2.1, 2.2, 3.1-3.6 of the proof of Property 7 (Theorem 5.18) are all considered in the algorithm. In all of these cases, $|S|$ decreases or the number of 2-leaves increases. The number of 2-leaves in a maximal realization does not depend on the choice of the realization, so all of these changes are always accepted as an improvement by Algorithm 5. So for S and G', these cases are excluded, and the same reasoning as in the rest of the proof of this property applies.

We now use a very rough analysis to show that the algorithm can be implemented in polynomial time. The number of sets U and W that satisfy the first step is bounded by a polynomial in $|V(G)|$. Every step of the algorithm can be implemented in polynomial time (Lemma 5.23). Every time the algorithm is repeated, the input is 'smaller' with regard to the relation '$\prec$'. The number of possible combinations of values that are considered for evaluating this relation is bounded by a polynomial in $|V(G)|$. □

It follows that for finding the CD-set described in Theorem 5.22, also a simple polynomial time algorithm exists.

Theorem 5.27 *Let G be a connected graph with $\delta(G) \geq 3$. We can find in polynomial time a CD-set S for G with $|S| \leq (5|V(G)| - 12 + D)/7$, where D is the number of cubic diamonds in G that contain three vertices of S.*

Proof: The algorithm is as follows. First replace diamond necklaces by cubic diamonds, as described in the proof of Theorem 5.22. For the resulting graph G_1, use Algorithm 5 to find a potential standard CD-set S and realization G'

that satisfy the properties stated in Theorem 5.18 (Lemma 5.26), and a CD-set $S' \subseteq S$. By adding vertices in the diamond necklaces of G, S' can be made into a CD-set S'' of G. See the proof of Theorem 5.22 for details.

We see that S' is a minimal CD-set, by the last step of Algorithm 5. So S'' satisfies the above bound (see again the proof of Theorem 5.22 for details). □

5.8 Discussion

In this chapter, we introduced new techniques to find small CD-sets/spanning trees with many leaves. We feel that our methods show that considering small CD-sets is more practical for this purpose than looking at leafy spanning trees, which was previously done to deduce bounds of this kind. For instance, the concept of minimality of a CD-set proves to be very useful, but no similar concept exists for spanning trees. We have also shown that there are simple but powerful ways of doing local search on minimal CD-sets, which again have no easy and equally powerful equivalent in the world of leafy trees. To put it differently, we feel that spanning trees contain too much information for our purposes: it does not matter which leaves are connected to which CD-set vertices, or which tree is chosen to connect the CD-set vertices.

These techniques may be useful to prove other similar results: for instance Linial conjectured that every graph on n vertices with minimum degree δ has a spanning tree with at least $n - 3n/(\delta + 1) + c_\delta$ leaves, for some appropriate constant c_δ. For sufficiently large δ, this conjecture has been disproved by Alon [1], but for small values of δ the conjecture has been shown to be true. For $\delta = 3$, Theorem 5.1 proves the conjecture. For $\delta = 4$, a proof appears in [38]. For $\delta = 5$, Griggs and Wu [35] gave a proof. For small values of δ greater than five, little is known. For more information, see [15].

The worst case examples for the bound from Theorem 5.22 presented in Section 5.6 still contain many diamonds: we expect that if we forbid all diamonds (not just cubic diamonds), every graph in this class has a CD-set with at most $\frac{2}{3}n - \frac{4}{3}$ vertices. So we expect that Theorem 5.2 can be generalized to all graphs with $\delta \geq 3$. Even though proving this will be hard, we think that the local search algorithm presented in Section 5.7, or a similar algorithm that considers a slightly larger neighborhood, might attain this bound. If this generalization can be proved, this would allow a further improvement of the time complexity of the algorithm presented in Chapter 6.

We mentioned that for our two bounds (Theorem 5.13 and 5.22) and the bound in Theorem 5.2, Q_3 is the only known example that prevents us to find a stronger linear bound. If we exclude this graph and a few other small graphs (in [34], $K_2 \times C_5$ and another graph on ten vertices are mentioned), it may be possible to show that the bounds become $|S'| \leq \frac{2}{3}|V| - 2$ resp. $|S'| \leq \frac{5}{7}|V| + \frac{1}{7}D - 2$. We have shown in Section 5.6 that the second bound cannot be improved further unless we exclude infinitely many graphs. For the other bound, similar examples can be constructed that show the same. Again we do not expect an easy proof for these improved bounds.

We remark that Algorithm 4 mentioned in Section 5.7 does not attain this improved bound: if we consider $K_2 \times K_3$, we see that Algorithm 4 may output a CD-set on three vertices (this is a minimal CD-set). Observe that if we add a local search step as used in Algorithm 5, we do find a good CD-set in this case (on two vertices). We expect that Algorithm 5 always finds CD-sets that satisfy the improved bounds.

Chapter 6

A fast FPT algorithm for finding spanning trees with many leaves

6.1 Introduction

We assume all graphs to be simple unless noted otherwise. The number of vertices of graph G is denoted by n. In the Max-Leaf Spanning Tree problem (MaxLeaf), an input consists of a connected graph G. The objective is to find a spanning tree for G with the maximum number of leaves. Such a tree will be called an *optimal tree*. We denote the set of leaves of graph T by $L(T)$. This problem has been well-studied over the last twenty years. On the negative side, MaxLeaf is known to be $\mathcal{NP}$-hard (Garey and Johnson [31]), and $\mathcal{APX}$-hard (Galbiati, Maffioli and Morzenti [29, 30]). On the positive side, the literature contains some polynomial time approximation algorithms for MaxLeaf that have fairly small worst case performance guarantees (a guarantee of 3 by Lu and Ravi [42], and a guarantee of 2 by Solis-Oba [53]).

We remark that in the literature, results on MaxLeaf also appear under the name *Minimum Connected Dominating Set*. It is easy to see that for a spanning tree T of a graph G, $V(G)\backslash L(T)$ is a connected dominating set (unless $T = K_2$), and that for every connected dominating set S, a spanning tree T exists with $\overline{S} \subseteq L(T)$. It follows that the problems are equivalent for most purposes (though not from an approximability viewpoint; see e.g. [36]).

In the remainder of this chapter, we consider the decision version of MaxLeaf:

INPUT: A connected, simple graph G and a positive integer k.
QUESTION: Does G have a spanning tree T with $|L(T)| \geq k$?

We can split the input of this problem into two parts, denoted as (G, k). This is a *parametrization* of the problem where the second part, k, is called the *parameter*. A parametrized problem (x, y) belongs to the complexity class FPT, if it has an algorithm with time complexity bounded by a function $f(|y|) \cdot g(|x|)$, where g is a polynomial (Downey and Fellows [23]). Here $|x|$ and $|y|$ denote the input size of x and y (in a reasonable encoding). The dependence $f(|y|)$ of the running time on $|y|$ may be arbitrary; for instance, $f(|y|)$ may grow doubly exponentially with $|y|$, or even worse. A problem with such an algorithm is said to be *fixed-parameter tractable* (FPT, for short), and such an algorithm is called an FPT algorithm. Note that an algorithm with time complexity $f(|y|) + g(|x|)$ is also an FPT algorithm if g is a polynomial.

To improve the readability, in our case we replace $|x|$ by n and $|y|$ by k in this definition. Observe that this does not functionally change the definition of FPT algorithms for MaxLeaf. We will call $f(k)$ the *parameter function* of the FPT algorithm. In most FPT algorithms, the factor $g(n)$ in the complexity is a low degree polynomial (often n^3 or lower). Therefore the factor $f(k)$ is most often used to compare the strength of different FPT algorithms.

Fellows and Langston [27] observed that MaxLeaf belongs to FPT via the graph minors machinery of Robertson and Seymour; their argument was non-constructive and did not explicitly yield an algorithm. Bodlaender [6] constructed the first FPT algorithm for MaxLeaf. Its time complexity was linear in n and had a parameter function $f(k)$ of roughly $(17k^4)!$; we stress that [6] was only interested in proving the existence of such an algorithm, and did not put any effort in getting a good time complexity. A little later, Downey and Fellows [22] constructed a better FPT algorithm for MaxLeaf with $f(k) = (2k)^{4k}$. This was further improved by Fellows, McCartin, Rosamond and Stege [28]; their parameter function is roughly $k\,(14.23)^k$. In a previous version of this chapter [12] we obtained an FPT algorithm with parameter function $O(9.49^k)$. Recently Estivill-Castro, Fellows, Langston and Rosamond [24] presented a general approach for constructing FPT algorithms and showed how this can be used to find an FPT algorithm for MaxLeaf with parameter function $O(8.80^k)$. See Section 6.6 for more details on this algorithm and more comparisons between these FPT algorithms. In this chapter, we use the new extremal result from Chapter 5 to obtain an FPT algorithm with parameter function $O(8.12^k)$.

The literature also contains a number of purely combinatorial results around problem MaxLeaf, mainly in extremal graph theory. Ding, Johnson and Seymour [21] prove that whenever a graph G satisfies $|E(G)| \geq n + \frac{1}{2}(k-1)(k-2)$ and $n \neq k+1$, then it has a spanning tree with at least k leaves. Another branch of extremal results deals with graphs with large minimum degree. Linial conjectured around 1987 that every graph G with $\delta(G) \geq \delta$ has a spanning tree with at least $n(\delta-2)/(\delta+1) + c_\delta$ leaves, where c_δ is a small positive constant only depending on δ. Alon [1] used a probabilistic argument to disprove Linial's conjecture for the cases where δ is sufficiently large. However, for small values of δ the conjecture turned out to be true. Kleitman and West [38] and Linial and Sturtevant [41] proved Linial's conjecture for $\delta = 3$ with $c_3 = 2$. Kleitman and West [38] proved it for $\delta = 4$ with $c_4 = 8/5$, and Griggs and Wu [35] proved

it for $\delta = 5$ with $c_5 = 2$. All these bounds are best possible. The cases with $\delta \geq 6$ are not well-understood. In particular, we do not know the value of the smallest δ for which Linial's conjecture is false. For more information, we refer the reader to Caro, West and Yuster [15].

Our previous result was based on the aforementioned bound for $\delta = 3$: every connected graph G with $\delta(G) \geq 3$ has a spanning tree with at least $n/4 + 2$ leaves [38]. This bound is best possible, but all worst case examples contain many cubic diamonds: a *diamond* is a complete graph on four vertices minus one edge, and a subgraph H of G is called a *cubic diamond* of G if it is a diamond induced by four vertices of degree three. Griggs, Kleitman and Shastri [34] showed that every connected cubic graph without diamonds, contains a spanning tree with at least $n/3+4/3$ leaves, and that this is the best possible linear bound. In Chapter 5 we showed that every connected graph G with $\delta(G) \geq 3$ without cubic diamonds has a spanning tree with at least $\frac{2}{7}n+\frac{12}{7}$ leaves. This is again the best possible linear bound. In fact, the following strictly stronger statement was proved, which gives the bound that we will use for our FPT algorithm:

Theorem 6.1 *Let G be a connected graph on n vertices with $\delta(G) \geq 3$. G has a spanning tree T with $|L(T)| \geq \frac{2}{7}n + \frac{12}{7} - \frac{d}{7}$, where d is the number of cubic diamonds of G that contain exactly one leaf of T.*

Using the trivial bound $d \leq n/4$, it follows that this can be seen as a generalization of the aforementioned bound for $\delta = 3$, apart from a small difference in the constant.

The contribution of this chapter is the following: we construct a fast FPT algorithm for MaxLeaf with parameter function $f(k) \in O(8.12^k)$. Our solution approach is quite different from the previous lines of attack in [6, 22, 28]. It is based on three main ingredients. The first ingredient is a preprocessing procedure that translates any instance (G, k) of MaxLeaf into an (equivalent) smaller instance (G', k') that satisfies a number of nice structural properties. This preprocessing procedure is described in detail in Section 6.3. The second ingredient is an (expensive) enumerative procedure that considers an exponential number of vertex subsets and checks whether they can form the leaf set of a spanning tree. This procedure is described in Section 6.4. The third ingredient is Theorem 6.1, which can be used to combine these algorithms into an FPT algorithm, and deduce its time complexity. The FPT algorithm is stated in Section 6.5, which also contains a proof of correctness and its time complexity. In Section 6.6 we discuss an alternative analysis of our algorithm, practical applications and further improvements. But first we define some key concepts for our methods in the next section.

6.2 Preliminaries

The operation of *suppressing* a vertex u of degree two consists of contracting an incident edge uv, and assigning the label v to the new vertex again. This is

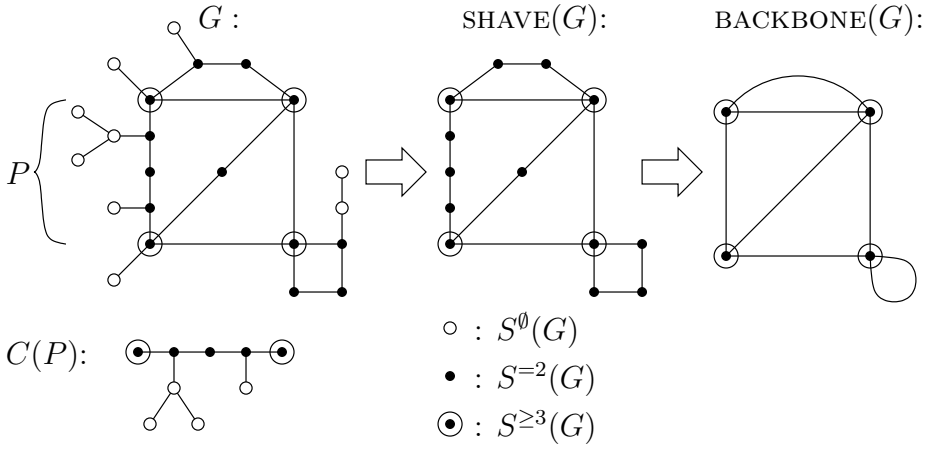


Figure 6.1: SHAVE(G), BACKBONE(G), the vertex partition and a c-path.

the reverse operation of an edge subdivision. Suppressing a vertex preserves all vertex degrees.

For the following definitions, see Figure 6.1 for an example. Recall that we allow a graph to have an empty vertex set; this graph is called the empty graph. We define a subgraph of G called SHAVE(G): if G is a tree, SHAVE(G) is the empty graph. Otherwise, SHAVE(G) is the graph obtained from G by repeatedly removing leaves, until no leaves remain. If SHAVE(G) has no vertices of degree at least three, BACKBONE(G) is the empty graph. Otherwise, BACKBONE(G) is obtained by suppressing all degree two vertices of SHAVE(G). So if G is not a tree and SHAVE(G) is not a cycle, BACKBONE(G) has minimum degree at least three. BACKBONE(G) can be a multi-graph with loops.

Related to these definitions, we will partition the vertex set of G into three disjoint classes $S^{\emptyset}(G)$, $S^{=2}(G)$, and $S^{\geq 3}(G)$.

- The set $S^{\emptyset}(G) = V(G) \backslash V(\text{SHAVE}(G))$ contains the vertices that do not appear in SHAVE(G).
- The set $S^{=2}(G) = V(\text{SHAVE}(G)) \backslash V(\text{BACKBONE}(G))$ contains the vertices that have degree two in SHAVE(G).
- The set $S^{\geq 3}(G) = V(\text{BACKBONE}(G))$ contains the rest of the vertices.

Let u and v be two distinct vertices in $S^{\geq 3}(G)$. A *caterpillar path* (or *c-path*, for short) between u and v is a subgraph of G that is a (u, v)-path, with all internal vertices in $S^{=2}(G)$. If $x_1, \ldots, x_r$ are the internal vertices of a c-path ordered from u to v, then $P = \langle u, x_1, \ldots, x_2, v \rangle$ denotes this c-path. Note that a single edge uv with $u, v \in S^{\geq 3}(G)$ also forms a c-path. Every edge in BACKBONE(G) that is not a loop corresponds to a c-path in G. A vertex $x \in S^{\emptyset}(G)$ is said to be in the *neighborhood* of a c-path from u to v, if there is a path P from an interior vertex of the c-path to x of which all internal

vertices are in $S^{\emptyset}(G)$. In other words, x is in the neighborhood if it belongs to one of the branches connected to the c-path. If P is a c-path, $C(P)$ denotes the (connected) subgraph of G induced by $V(P)$ and all the vertices in $S^{\emptyset}(G)$ that are in the neighborhood of the c-path P ($C(P)$ is the actual caterpillar). A vertex $x \in S^{=2}(G)$ without neighbors in $S^{\emptyset}(G)$ is called an *α-vertex*, and a vertex $x \in S^{=2}(G)$ with at least one neighbor in $S^{\emptyset}(G)$ is called a *β-vertex*.

6.3 The preprocessing phase

In this section, we will prove a number of reduction lemmas that altogether lead to the following theorem.

Theorem 6.2 *There exists a polynomial time algorithm that translates any instance (G, k) of* MaxLeaf *into another instance (G', k') that satisfies the following properties:*

1. *(G, k) and (G', k') are equivalent.*
2. *$\textsc{backbone}(G')$ is a simple graph.*
3. *$k' \leq k$.*
4. *G' contains no cubic diamonds.*
5. *Every vertex in $S^{\emptyset}(G')$ is a leaf, unless $G' = P_3$.*
6. *For every c-path in G', its internal vertices form an alternating sequence of α- and β-vertices.*

The second property states the main goal of the preprocessing: if $\textsc{backbone}(G')$ is simple, we can use existing extremal results. This will show that if $|S^{\geq 3}(G')| \geq 3.5k'$, the instance (G', k') is a YES-instance. The third property is needed to express this bound in k instead of k'. The final three properties give some additional structural information.

We will present a number of lemmas that correspond to operations on the instance. All of these operations work on a specific structure and remove it, such that step by step the desired structural properties are obtained. When we say an instance (G, k) containing a certain structure is *reducible*, we mean that we can replace this structure with a different structure giving a new equivalent instance (G', k'), such that $k' \leq k$. G' is again a simple graph. Furthermore, in such a replacement step the number of edges or the number of c-paths decreases, and the other number does not increase. This is needed in order to prove that the algorithm terminates in polynomial time.

An instance that is not reducible with our reduction rules is called a *reduced* instance. Our first four reduction rules already ensure that c-paths in a reduced instance will have one of only five different forms.

Lemma 6.3 *If G has a bridge $e = uv$ with $d(u) \geq 2$ and $d(v) \geq 2$, then (G, k) is reducible.*

Proof: Consider $G' = G \cdot e$, and $k' = k$. In every spanning tree of G, the bridge e is used and the vertices u and v are non-leaves. Hence, if T is a spanning tree of G with at least k leaves, $T \cdot e$ is a spanning tree of G' with at least k leaves. Similarly, the converse is also true. □

Lemma 6.4 *If G contains two leaves v_1 and v_2 that are adjacent to the same vertex u with $d(u) \geq 3$, then (G, k) is reducible.*

Proof: Let $G' = G - v_1$, and $k' = k - 1$. T is a spanning tree of G with k leaves if and only if $T - v_1$ is a spanning tree of G' with $k - 1$ leaves, since u is not a leaf in either graph. □

These two reduction rules can be applied until the graph G has the following structure:

Proposition 6.5 *Let (G, k) be a reduced instance. Every vertex in $S^{\emptyset}(G)$ is a leaf, unless $G = P_3$. Every vertex in $S^{\geq 3}(G) \cup S^{=2}(G)$ is adjacent to at most one vertex from $S^{\emptyset}(G)$.*

Proof: Observe that edges incident with $S^{\emptyset}$-vertices are bridges. So if $u \in V(G)$ is a $S^{\emptyset}$-vertex, (G, k) is reducible unless all neighbors of u are leaves (Lemma 6.3). In that case, if $d(u) \geq 3$ then (G, k) is reducible (Lemma 6.4). It follows that $G = P_3$.

If $S^{\geq 3}(G) \cup S^{=2}(G)$ is non-empty, G is not a tree, and thus all $S^{\emptyset}$-vertices are leaves. Therefore if a vertex in $S^{\geq 3}(G) \cup S^{=2}(G)$ is adjacent to multiple vertices from $S^{\emptyset}(G)$, Lemma 6.4 can be applied. □

In the following lemmas about reducibility, we use this proposition implicitly; when proving that a structure is reducible, we assume the properties in Proposition 6.5 hold, because otherwise the reduction rules from Lemma 6.3 and 6.4 can be applied. Observe that if $S^{\geq 3}(G) = \emptyset$, then (G, k) already satisfies the properties in Theorem 6.2. So from now on we will assume that $S^{\geq 3}(G) \neq \emptyset$.

Lemma 6.6 *Consider a c-path $P = \langle u, x_1, \ldots, x_r, v \rangle$.*

(a) If for some i, the vertices x_i and x_{i+1} both are α-vertices, then (G, k) is reducible.

(b) If for some i, the vertices x_i and x_{i+1} both are β-vertices, then (G, k) is reducible.

Proof: (a) Consider $G' = G - x_i x_{i+1}$, and $k' = k$. We argue that that there always exists an optimal tree for G that avoids the edge $x_i x_{i+1}$. Consider an optimal tree T that does use $x_i x_{i+1}$. $T - x_i x_{i+1}$ consists of two subtrees T_1 and T_2. By Lemma 6.3 we may assume that $x_i x_{i+1}$ is not a bridge in G, and so there is an edge $y_1 y_2$ in G that connects T_1 and T_2. Then it can be seen that the new tree $T' := T - x_i x_{i+1} + y_1 y_2$ is an optimal tree, too (the transformation cannot decrease the number of leaves). We conclude that optimal trees for G and G' have the same number of leaves.

(b) Consider $G' = G \cdot x_i x_{i+1}$, and $k' = k$. Observe that β-vertices are cut vertices, and therefore are never leaves in a spanning tree. We first prove the existence of an optimal tree of G that contains $x_i x_{i+1}$. Consider an optimal tree T of G with $x_i x_{i+1} \notin E(T)$. $T + x_i x_{i+1}$ contains a cycle C. Choose an edge $yz \neq x_i x_{i+1}$ on this cycle. $T_2 = T + x_i x_{i+1} - yz$ is again a tree, with at least as many leaves (x_i and x_{i+1} were not leaves in T). So if a spanning tree of G with at least k leaves exists, we may consider a spanning tree T of G with at least k leaves that contains $x_i x_{i+1}$. We use T to construct a spanning tree of G' with at least k leaves: consider $T' = T \cdot x_i x_{i+1}$. All leaves of T are leaves of T'. Similarly, every spanning tree T' of G' with at least k leaves corresponds to a spanning tree of G with at least k leaves, since the vertex resulting from the contraction of $x_i x_{i+1}$ is a β-vertex of G'. □

Lemma 6.7 *Consider a c-path $P = \langle u, x_1, \ldots, x_r, v \rangle$. If for some i, the vertices x_{i-1} and x_{i+1} both are α-vertices and x_i is a β-vertex, then (G, k) is reducible.*

Proof: We reduce this graph by contracting $x_{i-1}x_i$, $x_i x_{i+1}$ and $x_i y$, where y is the single leaf adjacent to x_i, and setting $k' = k - 1$. Vertex x_{i-1} is incident to the edge $x_{i-1}x_i$ and to a second edge e. Vertex x_{i+1} is incident to the edge $x_i x_{i+1}$ and to a second edge f.

We claim that there always exists an optimal tree for G that uses the two edges $x_{i-1}x_i$, $x_i x_{i+1}$: consider an optimal tree T with $x_{i-1}x_i \notin E(T)$. Then $T' = T - e + x_{i-1}x_i$ is a spanning tree with at least as many leaves. Similarly, if $x_i x_{i+1} \notin T$, then $T' = T - f + x_i x_{i+1}$ is a spanning tree with at least as many leaves. Clearly, $x_i y \in E(T)$. A spanning tree T with at least k leaves that contains these three edges, can be made into a spanning tree T' of G' with at least $k - 1$ leaves: in T, contract the same three edges into the single vertex z. If x_{i-1} or x_{i+1} was a leaf of T, z is a leaf of T'. These two vertices cannot both be leaves. So we only lose leaf y. Similarly, any spanning tree T' of G' with at least $k-1$ leaves corresponds to a spanning tree T of G with at least k leaves. □

We have the following immediate consequence of Lemmas 6.6 and 6.7.

Proposition 6.8 *For any c-path $P = \langle u, x_1, \ldots, x_r, v \rangle$ in a reduced instance, $r \leq 3$ holds, and the interior vertices are alternatingly α-vertices and β-vertices. Moreover, if $r = 3$ then x_1 and x_3 are β-vertices and x_2 is an α-vertex.*

At this point we have sufficiently characterized the possible forms of c-paths in a reduced instance: c-paths in a reduced instance will have one of only five different forms (no internal vertices; α-vertex; β-vertex; α-vertex and β-vertex; β-vertex, α-vertex and β-vertex). In the remaining proofs this proposition will be used implicitly again.

Now we will present a number of reduction rules that reduce *parallel c-paths*: c-paths that correspond to parallel edges in BACKBONE(G). It will turn out that in all cases we can reduce parallel c-paths.

6.3.1 The cases where both caterpillar paths have interior vertices

Lemma 6.9 *Let $P = \langle u, x_1, \ldots, x_r, v\rangle$ and $P' = \langle u, x'_1, \ldots, x'_s, v\rangle$ be two c-paths in G. If $r \geq 2$ and $s \geq 1$, and if x_1 and x'_1 both are β-vertices, then (G, k) is reducible.*

Proof: We will reduce (G, k) to $G' = G \cdot ux_1$ and $k' = k$.

In order to show that this is a valid reduction, we first argue that an optimal tree exists for G that contains ux_1, and does not have u as a leaf: consider an optimal tree T that does not contain ux_1. In that case, $x_1x_2 \in E(T)$. Then $T - x_1x_2 + ux_1$ is again a spanning tree, and has at least as many leaves: x_2 is an α-vertex and therefore a leaf in the new tree. Now let T be an optimal tree that contains ux_1, but has u as a leaf. In that case, $ux'_1 \notin E(T)$, and $x_1x_2 \in E(T)$. Then we see that $T + ux'_1 - x_1x_2$ is again a spanning tree, with the same number of leaves (we lose u and gain x_2 as a leaf).

We conclude that there always exists an optimal tree T that contains the edge ux_1, in which u and x_1 are non-leaves. Therefore, if T has at least k leaves, $T \cdot ux_1$ is a spanning tree of G' with at least k leaves. Similarly, a spanning tree T' of G' with at least k leaves corresponds to a spanning tree of G with at least k leaves, since the vertex resulting from the contraction is a cut vertex of G'.□

Lemma 6.10 *Consider two c-paths $P = \langle u, x_1, \ldots, x_r, v\rangle$ and $P' = \langle u, x'_1, v\rangle$ in G. If $r \geq 1$ and if the (unique) interior vertex x'_1 of P' is an α-vertex, then (G, k) is reducible.*

Proof: We reduce this instance to $G' = G - x'_1$ and $k' = k - 1$.

We claim there always exists an optimal tree for G that does not use both of the edges ux'_1 and x'_1v. Consider an optimal tree T. Suppose that it uses both edges ux'_1 and x'_1v. Then T must avoid exactly one edge yz in P. We distinguish three cases: first, if y and z both are interior vertices of P, then one of them is an α-vertex and one of them is a β-vertex. Hence, at most one of them can be a leaf in T. Then $T + yz - ux'_1$ has at least as many leaves as T. In the second case, we assume that T avoids the edge $yz = ux_1$. Consider $T + ux_1 - ux'_1$: this does not change whether u is a leaf. x'_1 becomes a leaf, so this new spanning tree has at least as many leaves as T. If $yz = x_rv$, the desired spanning tree is $T + x_rv - x'_1v$.

To summarize, there always exists an optimal spanning tree T in which x'_1 is a leaf. If T has at least k leaves, then $T - x'_1$ has at least $k - 1$ leaves. u and v cannot both be leaves in a spanning tree T' of G'. So if T' has at least $k - 1$ leaves, we can add either ux'_1 or vx'_1 to T', such that no leaves are lost. One leaf is gained, so G then has a spanning tree with at least k leaves. □

Lemma 6.11 *Consider two c-paths $P = \langle u, x_1, v\rangle$ and $P' = \langle u, x'_1, v\rangle$ in G. If x_1 and x'_1 both are β-vertices, then (G, k) is reducible.*

Proof: Let y be the leaf neighbor of x_1. We reduce (G, k) to $G' = G - x_1 - y$, $k' = k - 1$.

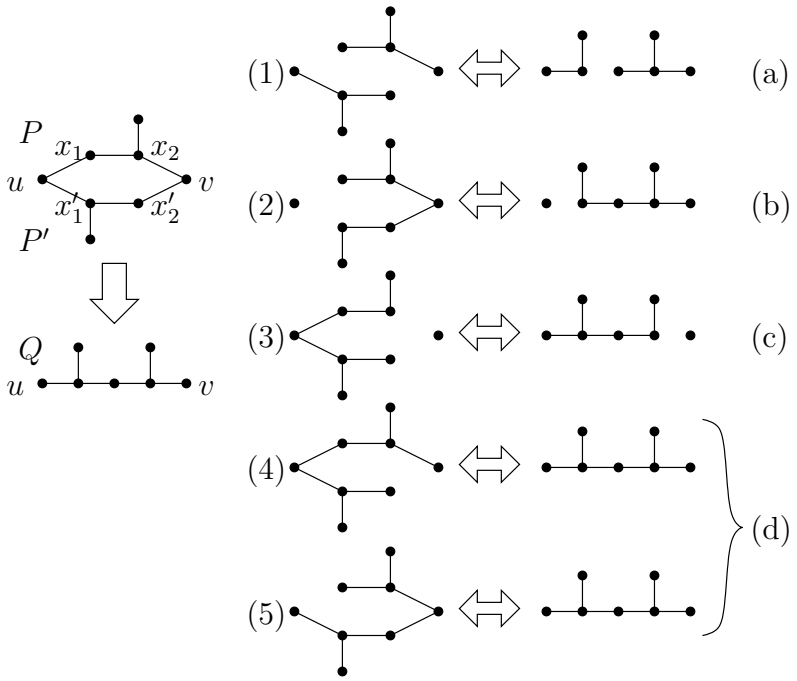


Figure 6.2: The reduction rule from Lemma 6.12

W.l.o.g., an optimal tree T of G avoids either ux_1 or x_1v. In both cases, if T has at least k leaves, then $T' = T - x_1 - y$ is a spanning tree of G' with at least $k-1$ leaves. If T' is a spanning tree of G' with at least $k-1$ leaves, then u or v is not a leaf. So adding edges ux_1 and x_1y resp. vx_1 and x_1y gives a spanning tree of G with at least k leaves. □

Lemma 6.12 *Consider two c-paths $P = \langle u, x_1, x_2, v\rangle$ and $P' = \langle u, x'_1, x'_2, v\rangle$ in G. If x_1 and x'_2 both are α-vertices, and if x_2 and x'_1 both are β-vertices, then (G, k) is reducible.*

Proof: Construct G' as follows: remove all internal vertices of P and P', and all adjacent leaves. Now add a c-path $Q = \langle u, y_1, y_2, y_3, v\rangle$, consisting of a β-vertex, an α-vertex and a β-vertex, in this order. Set $k' = k - 1$. Figure 6.2 illustrates this rule, and shows the five corresponding tree replacement rules.

There always exists an optimal tree T of G such that when restricted to the edges of P and P', it has one of the five forms shown in the middle of Figure 6.2: for any spanning tree T we can replace $E(T) \cap (E(P) \cup E(P')$ by one of these edge sets without destroying connectivity, introducing cycles, decreasing the number of leaves, or changing the leaf status of u or v. For these five cases, we can replace these edges by the corresponding set shown on the right side, which gives a spanning tree of T'. This replacement does not increase the degree of u and v. So if T has at least k leaves, T' has at least $k-1$ leaves.

On the other hand, we can see that G' always has an optimal tree that has one of the four different forms shown on the right side of Figure 6.2, when restricted to the edges of Q. Suppose such an optimal tree T' has at least $k-1$ leaves. Replace these edges by the corresponding edge set of G: in case (d), if v is a leaf in T', choose the fourth set, otherwise the fifth set.

In all cases we obtain again a spanning tree T of G. This replacement may increase the degree of u or v, but only if they are not a leaf in T'. So T has at least k leaves. □

We finish this section by checking that we have indeed settled all the cases where both c-paths $P = \langle u, x_1, \ldots, x_r, v\rangle$ and $P' = \langle u, x'_1, \ldots, x'_s, v\rangle$ have at least one interior vertex (and hence satisfy $r, s \geq 1$):

Lemma 6.13 *If G contains c-paths $P = \langle u, x_1, \ldots, x_r, v\rangle$ and $P' = \langle u, x'_1, \ldots, x'_s, v\rangle$ with $r \geq s \geq 1$, then (G, k) is reducible.*

Proof: We may assume that these c-paths alternatingly consist of α-vertices and β-vertices (Proposition 6.8). Moreover, there are at most three interior vertices, and if there are exactly three then the first and last one are β-vertices and the middle one is an α-vertex (Proposition 6.8).

First assume that $s = 1$: If x'_1 is an α-vertex, then by Lemma 6.10 this situation is reducible. If x'_1 is a β-vertex and $r \geq 2$, then Lemma 6.9 can be applied to reduce the instance. If x'_1 is a β-vertex and $r = 1$, then either Lemma 6.10 or Lemma 6.11 leads to a reduction. Next assume that $s = 2$ (and $r \geq 2$): if u or v are adjacent to two β-vertices on P and P', then Lemma 6.9 applies, otherwise Lemma 6.12 applies. The case $s = r = 3$ can be settled by Lemma 6.9. □

6.3.2 The cases where one of the caterpillar paths is an edge

In this section, we discuss the case where $u, v \in S^{\geq 3}(G)$ are connected by the edge uv and by a c-path $P = \langle u, x_1, \ldots, x_r, v\rangle$. There are four possible cases that are handled in the four lemmas below: (i) $r = 3$; (ii) $r = 2$; (iii) $r = 1$ and x_1 is a β-vertex; (iv) $r = 1$ and x_1 is an α-vertex.

Lemma 6.14 *If $uv \in E(G)$ and there is a c-path $P = \langle u, x_1, x_2, x_3, v\rangle$ in G, then (G, k) is reducible.*

Proof: We reduce to $G' = G - x_1x_2$, $k' = k$. x_1 and x_3 are β-vertices and x_2 is an α-vertex. We claim that there always exists an optimal tree for G that does not use the edge x_1x_2: if an optimal tree T does not use ux_1, x_2x_3 or x_3v, then we can simply add this edge and remove x_1x_2, without decreasing the number of leaves. Otherwise, $uv \notin E(T)$. In this case, u or v is not a leaf, so $T + uv - x_1x_2$ has at least as many leaves.

Since the edge x_1x_2 is not needed in an optimal tree, we may remove it from G and reduce the instance. □

Lemma 6.15 *If $uv \in E(G)$ and there is a c-path $P = \langle u, x_1, x_2, v\rangle$ in G, then (G, k) is reducible.*

Proof: W.l.o.g., x_2 is an α-vertex. We reduce (G, k) to $G' = G - x_2v$, $k' = k$.

Similar to the previous proof, we can show that there exists an optimal tree T of G which avoids x_2v, which shows that the reduction is valid. □

Lemma 6.16 *If $uv \in E(G)$ and there is a c-path $P = \langle u, x_1, v\rangle$ in G, where x_1 is a β-vertex, then (G, k) is reducible.*

Proof: In this case, there always exists an optimal tree for G that does not use the edge uv. We reduce the instance by removing uv from G, and setting $k' = k$. □

Lemma 6.17 *If $uv \in E(G)$ and there is a c-path $P = \langle u, x_1, v\rangle$ in G, where x_1 is an α-vertex, then (G, k) is reducible.*

Proof: We remove uv, and add a vertex y and edge x_1y. $k' = k$. Note that this operation does not decrease the number of edges (this number remains the same), but it does decrease the number of c-paths. Therefore, this is still a valid reduction rule according to our conditions, once we establish that the resulting instance is equivalent.

Similar to the previous proofs, we find that an optimal tree T exists that avoids ux_1 or x_1v. W.l.o.g., $ux_1 \notin E(T)$. If $uv \in E(T)$, we consider $T' = T - uv + y + ux_1 + x_1y$, which has the same number of leaves (y is a new leaf, x_1 is not a leaf anymore, and the degree of u does not change). If $uv \notin E(T)$, we consider $T' = T + x_1y$, which also has the same number of leaves. Similarly, a spanning tree T' of G' corresponds to a spanning tree T of G with the same number of leaves, since u and v cannot both be leaves in T'. □

These reductions cover all cases where one of the c-paths is an edge. So we may conclude that for a reduced instance (G, k), BACKBONE(G) contains no parallel edges. Now we only need to remove loops to ensure that BACKBONE(G) is simple.

6.3.3 Removing loops and cubic diamonds

Lemma 6.18 *If G has a cycle C that contains exactly one vertex $u \in S^{\geq 3}(G)$, then (G, k) is reducible.*

Proof: u is a cut vertex of G, so in any spanning tree of G, u is not a leaf. A spanning tree T of G avoids exactly one edge of C. If C contains two adjacent α-vertices v and w, there exists an optimal spanning tree that avoids vw. Otherwise, if C contains at least one α-vertex v, we can choose an arbitrary neighbor w, and observe that there always exists an optimal tree that avoids vw. In the final case, when C contains only β-vertices (and u), we can choose an arbitrary edge vw and observe that an optimal tree exists that avoids vw.

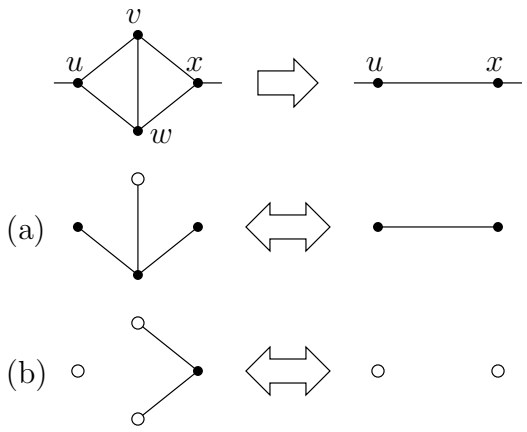


Figure 6.3: Removing a cubic diamond, and the corresponding spanning trees

In all cases, we can reduce to $G' = G - vw$, $k' = k$. □

Our final reduction rule is concerned with removing cubic diamonds from BACKBONE(G), but only if these diamonds also correspond to cubic diamonds in G.

Lemma 6.19 *If G contains a cubic diamond $D = G[\{u, v, w, x\}]$ with $ux \notin E(G)$, then (G, k) is reducible.*

Proof: We reduce to $G' = G - v - w + ux$, $k' = k - 1$.

Let T be an optimal tree of G, and suppose it has at least k leaves. We may assume w.l.o.g. that $E(T) \cap E(D)$ contains either the edges shown in Figure 6.3(a) or those shown in Figure 6.3(b). In the first case, we replace these edges of T by a single edge uv, to obtain a spanning tree T' of G' with at least $k - 1$ leaves. (If u or v previously was a leaf, it is again a leaf.) In the second case, we remove these edges to obtain a spanning tree T' of G' with at least $k - 1$ leaves. Similarly, if a spanning tree of G' contains uv, we replace this by the edge set in Figure 6.3(a), otherwise we add the edge set in Figure 6.3(b). In both cases this yields a spanning tree of G with one more leaf. □

6.3.4 Summary of the preprocessing algorithm

In this section, we use the previous lemmas to prove Theorem 6.2.

Proof of Theorem 6.2:
Our pre-processing algorithm on instance (G, k) works as follows: as long as one of the structures mentioned in Lemmas 6.3-6.19 is present, we apply the corresponding reduction rule. The priority of these rules corresponds to the order in which the lemmas are stated, so for instance the rule in Lemma 6.9 is

only applied on instances for which the properties in Proposition 6.8 hold. Let (G', k') be the final instance which cannot be reduced any further, the reduced instance.

All of the aforementioned reduction rules work on structures that can be recognized in polynomial time, and the operations can be carried out in polynomial time. Since every rule decreases $|E(G)| + |E(\textsc{backbone}(G))|$, the preprocessing will terminate in a linearly bounded number of steps. Thus the preprocessing algorithm has polynomial time complexity. Observe also that every reduction rule yields again a connected simple graph. Now we prove the properties stated in Theorem 6.2 one by one.

1. (G, k) is equivalent to the reduced instance (G', k'): all reduction rules give an equivalent instance.
2. We show that for a reduced instance (G', k'), $\textsc{backbone}(G')$ is a simple graph. For a graph G, parallel edges in $\textsc{backbone}(G)$ correspond to c-paths P and P' in G that have the same end vertices u and v. If both P and P' contain at least one internal vertex, (G, k) is reducible (Lemma 6.13). If P consists of the single edge uv, then Lemmas 6.14-6.17 cover all cases (Proposition 6.8)), and show that (G, k) is reducible. It follows that for a reduced instance (G', k'), $\textsc{backbone}(G')$ has no parallel edges. Finally, Lemma 6.18 shows that $\textsc{backbone}(G')$ contains no loops.
3. All of our reduction rules do not increase k, so $k' \leq k$.
4. By Lemma 6.19, G' contains no cubic diamonds.
5. Proposition 6.5 shows that this property holds.
6. Proposition 6.8 shows that this property holds.

□

6.4 An enumerative procedure

Algorithm 6 is the enumerative procedure. This algorithm answers the decision problem by checking an exponential number of sets (subsets of the leaves), and for all of these sets finding a spanning tree that is optimal in some way in polynomial time.

In the proof of the next lemma we show how to implement the steps in the algorithm; how to decide if a spanning tree T with $L \subseteq L(T)$ exists, and how to find such a spanning tree such that $|L(T) \backslash S^{\geq 3}(G)|$ is maximized, all in polynomial time. In Lemma 6.21, we show that by checking all of these sets, Algorithm 6 finds the correct answer.

Lemma 6.20 *Let graph G satisfy the properties from Theorem 6.2. For every $L \subseteq S^{\geq 3}(G)$ we can decide in polynomial time if a spanning tree T of G exists with $L \subseteq L(T)$, and in that case we can find in polynomial time a spanning tree T of G with $L \subseteq L(T)$ that maximizes $|L(T) \backslash S^{\geq 3}(G)|$.*

Algorithm 6 the enumerative procedure

Input: a MaxLeaf instance (G, k) that satisfies the properties stated in Theorem 6.2, and $S^{\geq 3}(G) \neq \emptyset$.

```
for all subsets L ⊆ S^{≥3}(G) with |L| ≤ k do
  if a spanning tree T of G with L ⊆ L(T) exists then
    Find a spanning tree T of G with L ⊆ L(T), and |L(T)\S^{≥3}(G)| maximum.
    if |L(T)| ≥ k then
      return YES, stop
    end if
  end if
end for
return NO
```

Proof: The first part of this proof shows how to decide if such a spanning tree exists, and if it exists, how we construct T. In the second part of the proof, we analyze the constructed graph T: we prove that it is a spanning tree and that it maximizes $|L(T) \backslash S^{\geq 3}(G)|$.

Tree construction It is easy to see that a spanning tree T with $L \subseteq L(T)$ does not exist if $V(G) \backslash L$ is not a dominating set or does not induce a connected graph. In the other case, we will construct a spanning tree. Observe that in this case, c-paths between two vertices in L consist of a single edge, vertices in L are not adjacent to $S^{\emptyset}$-vertices, and in $\text{BACKBONE}(G)$, every vertex in L is adjacent to a vertex not in L. Also, since $S^{\geq 3}(G) \neq \emptyset$ and $\text{BACKBONE}(G)$ is simple (Property 2 from Theorem 6.2), note that every $S^{=2}$-vertex of G is part of a c-path. Figure 6.4 shows an example of our tree construction. Observe that the constructed tree T is not optimal: with a simple change vertex v can also be made into a leaf, without losing any other leaves. However, since $v \in S^{\geq 3}(G)$, v is not counted towards the number of leaves we want to maximize, so this does not matter.

Consider $H = \text{BACKBONE}(G) - L$. Assign weights w to the edges of H as follows: if the c-path corresponding to edge e contains an α-vertex, $w(e) = 1$, otherwise $w(e) = 0$. $w(H)$ denotes the sum of the weights w over all edges of H. These weights can be interpreted as the number of leaves we can gain on this c-path if we do not use all edges of the c-path in our spanning tree (end vertices of a c-path are not counted for the number of leaves). We construct a spanning tree T_H of H such that $w(T_H)$ is minimum. It is well-known that such a spanning tree can be found in polynomial time using a greedy algorithm.

Using this spanning tree T_H of H, we construct a spanning tree T of G: start with $T = G$, and remove edges as follows. For every edge $e \in E(H) \backslash E(T_H)$ that corresponds to c-path P in G:

- If $w(e) = 1$, then remove an edge of P incident to an α-vertex (this vertex becomes a leaf).

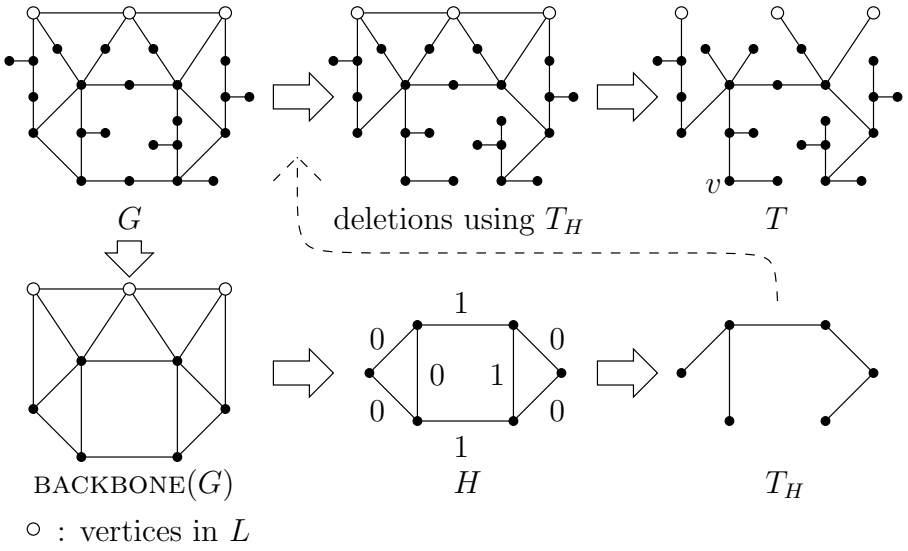


Figure 6.4: The construction of spanning tree T of G, with $L \subseteq L(T)$.

- If $w(e) = 0$, then remove an arbitrary edge of P.

For every $v \in L$:

- If v is adjacent to a β-vertex or an $S^{\geq 3}$-vertex not in L, then let u be this vertex. Otherwise, let u be an α-vertex adjacent to v. Note that in both cases, u is part of a c-path with one end vertex not in L. Remove all edges incident with v, except uv.

We call the resulting graph T.

Analysis of the constructed graph In order to prove that T is a spanning tree of G, we first show that for every pair $u, v \in V(G)$, a (u, v)-path exists in T. Consider the subgraph T_B of BACKBONE(G) that contains all edges that correspond to c-paths that are *used* in T: by this we mean the c-paths of which all edges are included in T. Observe that the edge set of T_B is exactly the edge set of T_H plus for every $v \in L$, one edge from v to a vertex $u \in V(T_H)$. So T_B is a spanning tree of BACKBONE(G). For every (u, v)-path in T_B, we obtain a (u, v)-path in T by replacing all edges with the corresponding c-paths. So in order to show that every pair of vertices in T is connected, we only need to show that for every vertex in T, we can find a path in T to an $S^{\geq 3}$-vertex of G. We removed at most one edge from every c-path: c-paths with two end vertices in L consist of a single edge, and for other c-paths the statement follows directly. We did not remove edges incident with $S^{\emptyset}$-vertices: no $S^{\emptyset}$-vertex is adjacent to a vertex in L. It follows that for every vertex there is a path in T to an $S^{\geq 3}$-vertex, and thus for every pair of vertices $u, v \in V(G)$, a (u, v)-path exists in T.

Next we show that T contains no cycles. c-paths together with their neighborhood form a tree, so if a cycle exists, it contains $S^{\geq 3}$-vertices. If $v_1, \ldots, v_k$ are these $S^{\geq 3}$-vertices numbered along the cycle, then $v_1, v_2, \ldots, v_k, v_1$ is a cycle in the aforementioned spanning tree T_B of BACKBONE(G), a contradiction.

Finally, we prove that our choice of T maximizes $|L(T) \backslash S^{\geq 3}(G)|$. So we only count leaves of T that are $S^{=2}$-vertices or $S^{\emptyset}$-vertices of G. Since every $S^{\emptyset}$-vertex is a leaf of G (Property 5 of Theorem 6.2), all $S^{\emptyset}$-vertices are leaves in every spanning tree of G. Therefore we only have to show that $|L(T) \cap S^{=2}(G)|$ is maximum. We do this by comparing T with an arbitrary spanning tree T' of G with $L \subseteq L(T')$.

First we consider c-paths between two vertices not in L. For every such c-path P, P contains a leaf of T if P contains an α-vertex, and P is not used in T. Such a path P corresponds to an edge e of H with $w(e) = 1$ and $e \notin E(T_H)$. We conclude that the number of this kind of leaves gained is $w(H) - w(T_H)$. Now we count the number of leaves of T' on these c-paths. For T' we define a subgraph T'_B of BACKBONE(G) in the same way as previously: $uv \in E(T'_B)$ if the c-path between u and v is used in T'. Observe that T'_B is a spanning tree of BACKBONE(G). If $u \in S^{\geq 3}(G)$ is a leaf of T', it is a leaf of T'_B, so $T'_H = T'_B - L$ is a spanning tree of H. If a c-path P is used in T', none of its internal vertices become leaves in T'. If P is not used in T', at most one edge e of P is not present in T'. Leaves are gained on this path only if end vertices of e are α-vertices. Since α-vertices on c-paths are not adjacent (Property 6 of Theorem 6.2), at most one vertex on an unused c-path can become a leaf in T', and none if the c-path contains no α-vertices. We see that the number of leaves of T' that are $S^{=2}$-vertices on c-paths between two vertices not in L is at most $w(H) - w(T'_H)$, where T'_H is a spanning tree of H. Since T_H was chosen to be a minimum weight spanning tree of H, this number is not greater than $w(H) - w(T_H)$, the number of leaves of T on these paths.

c-paths with two end vertices in L have no internal vertices and therefore do not count towards the number of $S^{=2}$-vertices that become leaves. Now we consider c-paths P with one end vertex $u \in L$ and one end vertex $v \notin L$. For every $u \in L$, there is exactly one c-path of this type incident with u that is used in T'. For all other incident c-paths P of this type, the edge incident with u is the only edge not present in T'. A leaf is gained if and only if the neighbor v of u on such an unused c-path is an α-vertex. It follows that our construction of T also maximizes the number of leaves on c-paths of this type.

We have considered all $S^{=2}$-vertices on all c-paths of G, and all $S^{=2}$-vertices of G are part of a c-path. Therefore, for the spanning tree T we constructed, $|L(T) \cap S^{=2}(G)| \geq |L(T') \cap S^{=2}(G)|$ holds for every spanning tree T' with $L \subseteq L(T')$. □

Lemma 6.21 *If a spanning tree T of G with $|L(T)| \geq k$ exists, Algorithm 6 returns YES, otherwise algorithm 6 returns NO.*

Proof: Clearly Algorithm 6 only returns YES if a spanning tree T with $|L(T)| \geq k$ exists. Now suppose a spanning tree T of G with $|L(T)| \geq k$ exists. Let

$L = L(T) \cap S^{\geq 3}(G)$. If $|L| \leq k$, the set L is considered in the algorithm, and a spanning tree T' is found that maximizes $|L(T') \backslash S^{\geq 3}(G)|$. So $|L(T')| \geq |L(T)|$, and the algorithm returns YES.

If $|L| > k$, then consider an arbitrary set $L' \subset L$ with $|L'| = k$. This set L' is considered Algorithm 6. Clearly, a spanning tree T' with $L' \subseteq L(T')$ exists, so the algorithm finds a spanning tree with at least k leaves, and returns YES. □

6.5 The FPT algorithm

Algorithm 7 is our FPT algorithm for MaxLeaf, which uses the preprocessing algorithm from Section 6.3 and enumerative procedure from Section 6.4. Correctness is proved in Lemma 6.22, and its time complexity is analyzed in Lemma 6.23.

Algorithm 7 the FPT algorithm

input: (G, k)

STEP 1: Apply preprocessing to (G, k), and call the resulting instance (G', k').

STEP 2: **if** G' contains no $S^{\geq 3}$-vertices **then** use a straightforward algorithm to find the answer, stop. (See the text for details.)

STEP 3: **if** $|S^{\geq 3}(G')| \geq 3.5k'$ **then return** YES, stop.

STEP 4: Use Algorithm 6 on (G', k').

First we show how to implement step 2. If G' has no $S^{\geq 3}$-vertices, then either G' is a tree or $\textsc{shave}(G')$ is a cycle. If G' is a tree, (G', k') is a YES-instance if and only if $|L(G')| \geq k'$. If $\textsc{shave}(G')$ is a cycle, a spanning tree of G' is obtained by deleting one edge. It is easy to find the best edge e to delete, and to find the correct answer.

Lemma 6.22 *Algorithm 7 returns YES if and only if (G, k) is a YES-instance for* MaxLeaf*.*

Proof: Theorem 6.2 shows that step 1 gives an equivalent instance. Above we showed how step 2 can be implemented such that the answer is correct. Lemma 6.21 shows that the enumerative procedure used in step 4 gives the correct output. Now we will show that if in step 3 a YES answer is given, this is correct; we will show that if $|S^{\geq 3}(G')| \geq 3.5k'$, then G' has a spanning tree with k' leaves, using Theorem 6.1 for $\textsc{backbone}(G')$.

When step 3 is entered, all properties stated in Theorem 6.2 hold for G', and $S^{\geq 3}(G') \neq \emptyset$, so $\textsc{backbone}(G')$ is a simple graph with minimum degree at least

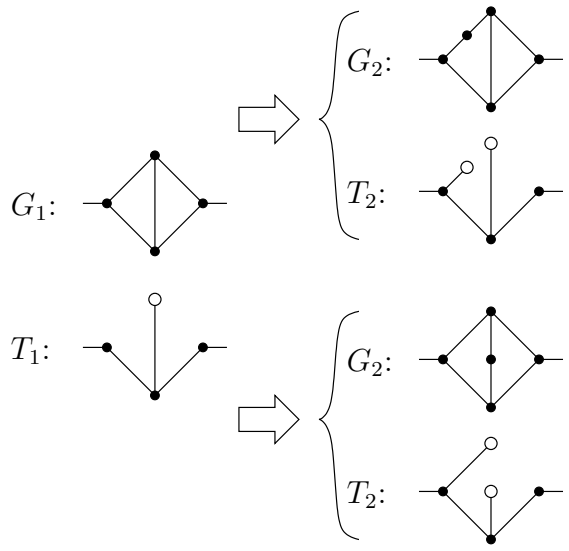


Figure 6.5: Changing the spanning tree when a diamond edge is subdivided

three. Since G' contains no cubic diamonds (Property 4 from Theorem 6.2), a cubic diamond in BACKBONE(G') does not correspond to a cubic diamond in G'. More formally this can be described as follows: G' can be constructed from BACKBONE(G') by applying edge subdivisions and adding leaves (introducing a vertex and connecting it to an existing vertex). If D is a cubic diamond in BACKBONE(G'), then in this construction of G', at least one edge of D is subdivided, or at least one leaf is added adjacent to a vertex of D. We will use this fact to show that if BACKBONE(G') has a spanning tree T with at least k leaves, then G' has a spanning tree with at least $k + d$ leaves, where d is the number of cubic diamonds of BACKBONE(G') that contain exactly one leaf of T.

Let T_1 be a spanning tree of G_1, and let D be a cubic diamond of G_1 that contains exactly one leaf of T_1. W.l.o.g., we may assume T_1 contains the edges shown in Figure 6.5. Apply an edge subdivision on an edge of D, and call the resulting graph G_2, and the subgraph corresponding to D we call D'. Now we can gain one leaf: remove the edges of D from $E(T_1)$, and add the appropriate edge set from Figure 6.5 or a symmetric edge set to $E(T_1)$. Note that since D is a cubic diamond, we can choose to use a different leaf than the original leaf in D. The resulting graph T_2 is is a spanning tree for G_2 with one more leaf. Now let G_2 be obtained from G_1 by adding a leaf adjacent to one of the vertices of D. In this case, we can also gain a leaf: in T_1, replace the edges of D with one of the edge sets shown in Figure 6.6, or a symmetric edge set.

We remark that we need to assume that D contains only one leaf of T_1: otherwise there are cases where no leaf can be gained after an edge subdivision or leaf addition.

This allows us to prove that if $|S^{\geq 3}(G')| \geq 3.5k'$, the instance is a YES-

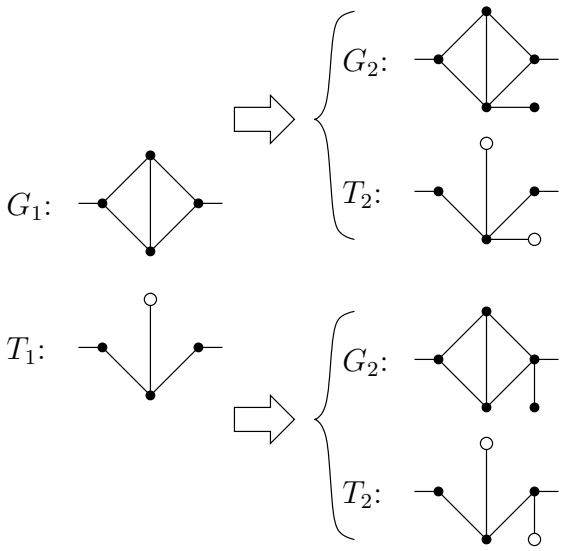


Figure 6.6: Changing the spanning tree when a leaf is added to a diamond

instance. Let $n = |S^{\geq 3}(G')| = |V(\text{BACKBONE}(G')|$. Now we can apply Theorem 6.1, which shows that BACKBONE(G') has a spanning tree T with $|L(T)| \geq \frac{2}{7}n + \frac{12}{7} - \frac{d}{7}$, where d is the number of cubic diamonds that contain exactly one leaf of T (we use the fact that BACKBONE(G') is simple: Property 2 from Theorem 6.2).

We construct G' from BACKBONE(G') by applying edge subdivisions and leaf additions. At every step we maintain a spanning tree T: if we subdivide an edge that is part of a cubic diamond D, or add a leaf adjacent to a vertex in a cubic diamond D, and D contains one leaf of T, we change T as described above. We gain one leaf in this case. Otherwise, we extend T in the straightforward way, which does not decrease the number of leaves. We observed that on every cubic diamond of BACKBONE(G'), at least one of these operations is applied. Therefore when G' is obtained, at least d leaves are gained. It follows that the constructed spanning tree of G' has at least $\frac{2}{7}n + \frac{12}{7} - \frac{d}{7} + d \geq \frac{2}{7}n + \frac{12}{7}$ leaves, so the instance is a YES-instance if $k' \leq \frac{2}{7}n + \frac{12}{7}$, or alternatively, if $n \geq \frac{7}{2}k' > \frac{7}{2}k' - 6$. This proves that the output of step 3 is correct. □

Lemma 6.23 *Algorithm 7 with input (G, k) has time complexity $g(m) + f(k)$, with $m = |V(G)|$, $g(m)$ a polynomial, and $f(k) \in O(8.12^k)$.*

Proof: The preprocessing step can be implemented such that the complexity is polynomial in m (Theorem 6.2). The same is true for step 2 and step 3. So the time complexity of step 1–3 is polynomial in m, where the degree of the polynomial depends on the implementation. Note however that this is a practical, small degree polynomial.

Now we analyze the complexity of step 4. Let $n = |S^{\geq 3}(G')|$. Step 4 is only done if $n < 3.5k'$. The enumerative procedure checks all subsets of size at most k' of the vertices of BACKBONE(G'). For every such subset, a polynomial time algorithm is applied (polynomial in n, so also in k'). The number of subsets is

$$\binom{n}{k'} + \binom{n}{k'-1} + \ldots + \binom{n}{0} \leq$$

$$\binom{3.5k'}{k'} + \binom{3.5k'}{k'-1} + \ldots + \binom{3.5k'}{0} \leq k' \binom{3.5k'}{k'} + 1.$$

Therefore there is a polynomial $g(k')$ such that the complexity of the last step is $O(g(k')\binom{3.5k'}{k'})$. In addition, $k' \leq k$ (Property 3 from Theorem 6.2). Now we use Stirling's approximation $x! \approx x^x e^{-x}\sqrt{2\pi x}$, or alternatively, $x! \in \Theta(x^x e^{-x}\sqrt{x})$. This gives

$$\binom{3.5k'}{k'} \leq \binom{3.5k}{k} = \frac{(3.5k)!}{(2.5k)!k!} \in$$

$$O\left(\frac{(3.5k)^{3.5k}\sqrt{3.5k}}{e^{3.5k}} \cdot \frac{e^{2.5k}}{(2.5k)^{2.5k}\sqrt{2.5k}} \cdot \frac{e^k}{k^k\sqrt{k}}\right) = O\left(\frac{3.5^{3.5k}}{2.5^{2.5k}\sqrt{k}}\right) \subset$$

$$O\left(\left(\frac{80.2118}{9.8821}\right)^k\right) \subset O(8.117^k).$$

For any polynomial $g(k)$, we have $g(k)8.117^k \in O(8.12^k)$, so we conclude that the parameter function of this algorithm is $f(k) \in O(8.12^k)$. □

The above lemmas can be summarized as follows:

Theorem 6.24 *Algorithm 7 is an FPT algorithm for* MAXLEAF *with parameter function* $f(k) \in O(8.12^k)$.

6.6 Discussion

We have found an FPT algorithm for MAXLEAF with parameter function $f(k) \in O(8.12^k)$. This is the best parameter function for an FPT algorithm for MAXLEAF yet. We remark that even though we did not completely optimize and analyze the polynomial algorithms that are used in our algorithm, these are all algorithms with reasonable complexities; polynomials with no degrees greater than three. It follows that considering the parameter function is indeed a good way to assess the speed of our algorithm.

There is another commonly used way of comparing the strength of certain types FPT algorithms. The first part of our algorithm is an example of a *kernelization* algorithm: in terms of MAXLEAF, this is a polynomial time algorithm that either answers the question, or translates one instance (G, k) into an equivalent instance (G', k'), such that $|V(G')|$ and k' are both bounded by

some polynomial in k. Observe that if $|V(G')| = n \leq g(k)$, then there is a trivial enumerative procedure that finds an optimal spanning tree for G' in time $f(n)2^n \leq f(g(k))2^{g(k)}$, where $f(n)$ is a polynomial, by trying all possible leaf sets. So any kernelization procedure automatically leads to an FPT algorithm. Different FPT algorithms that are based on a kernelization step can also be compared by looking at this function $g(k)$ that bounds $|V(G')|$.

In our case, the bound for k' is trivial: $k' \leq k$. To bound $|V(G')|$ in terms of k, we observe that after our preprocessing, for every c-path P, the number of internal vertices and $S^{\emptyset}$-vertices in its neighborhood is at most five (Proposition 6.5, Proposition 6.8). Every $S^{\geq 3}$-vertex of G' is adjacent to at most one $S^{\emptyset}$-vertex (Lemma 6.4). It follows that $|V(G')| \leq 2|V(\textsc{backbone}(G'))| + 5|E(\textsc{backbone}(G'))| < 2|S^{\geq 3}(G')| + \frac{5}{2}|S^{\geq 3}(G')|^2$. If $|S^{\geq 3}(G')| \geq 3.5k$, we may answer YES immediately (see Algorithm 7), and otherwise $|V(G')| \leq 2 \cdot 3.5k + \frac{5}{2}(3.5k)^2 = 7k + 30.625k^2$. So, even though we can bound $|V(\textsc{backbone}(G'))|$ by $3.5k$, which is all we need for our enumerative algorithm, for the reduced instance G' itself the bound is worse.

The recent algorithm by Estivill-Castro et al. [24] is also an example of a kernelization algorithm. The bound on their kernel (G', k') is much better than ours: $|V(G')| \leq 3.75k$. However, their method does not easily allow a fast enumerative procedure which has to check subsets of only a *part* of the reduced instance (in our case, $S^{\geq 3}(G')$). So we deduced the parameter function of their algorithm as follows: one can check all subsets L of $V(G')$ of size *exactly* k, and check in polynomial time if there is a spanning tree T of G' with $L \subseteq L(T)$. Using $|V(G')| \leq 3.75k$ and an analysis very similar to the proof of Lemma 6.23, we conclude that their parameter function is $f(k) \in O(8.80^k)$.

For practical purposes, our algorithm with parameter function $f(k) \in O(8.12^k)$ is still not very useful for worst case instances, even for relatively small values of k like $k = 15$. However, for instances with many vertices of degree one and two, the algorithm works well for much larger values of k. In this case, a lot of preprocessing can be done and/or the backbone graph is relatively small, as is therefore the running time of the enumerative procedure.

But even though this algorithm may not be practical for many instances, parts of the algorithm can be combined with other (heuristic) approaches. In particular, the preprocessing can be combined with any other approach, to yield fast, practical algorithms.

Our algorithm is formulated only for the decision problem. In practice the optimization version of MaxLeaf is more interesting. Fortunately, since the extremal result we used is constructive (see Chapter 5), our algorithm can also be translated to an optimization algorithm. For instance, for any constant C, we can give an algorithm and a number k such that for any instance G with $|V(G)| = n$, in time $C + g(n)$ (where $g(n)$ is a polynomial) we find

- An optimal spanning tree of G, or
- a spanning tree of G with at least k leaves.

The first case occurs when the preprocessing returns a small reduced instance,

which can be analyzed in time C, and the second case occurs when the reduced instance is large, and the algorithm from Chapter 5 gives a spanning tree with at least k leaves.

Finally, we discuss some ideas for further improvements. Though the extremal result in Theorem 6.1 is best possible for its class (graphs with $\delta(G) \geq 3$ without cubic diamonds), it is expected that stronger bounds can be obtained for more restricted graph classes (see Chapter 5). For instance, bounds of the form $|L(T)| \geq |V(G)|/3+C$ may exist for the class of graphs with $\delta(G) \geq 3$ that contain no diamonds at all, or just no diamond necklaces. Loosely speaking, a diamond necklace is a string of diamonds separated from the rest of the graph by a 2-edge cut, thus a generalization of a cubic diamond. If such a bound can be proved for graphs without diamond necklaces, this can almost immediately be combined with our algorithm to get a parameter function $f(k) \in O(6.75^k)$.

This is just one way in which stronger extremal results can be used in our algorithm: for any extremal result for a graph class $\mathcal{G}$, if we can extend our preprocessing to guarantee that BACKBONE(G') is part of $\mathcal{G}$, the bound can be used in our algorithm. Unfortunately, it does not seem easy to find preprocessing rules that raise the minimum degree of BACKBONE(G') or remove triangles from BACKBONE(G'), in which case there would be a number of extremal results ready to be used (see Section 6.1, and Chapter 5).

Bibliography

[1] N. Alon. Transversal numbers of uniform hypergraphs. *Graphs and Combinatorics*, 6(1):1–4, 1990.

[2] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[3] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 222–231, New York, 2004. ACM.

[4] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[5] A. Bellaachia and A. Youssef. Communication capabilities of product networks. *Telecommunication Systems*, 13:119–133, 2000.

[6] H. L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, 1993.

[7] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[8] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[9] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. American Elsevier Publishing Co., Inc., New York, 1976.

[10] P. S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. In *Graph-theoretic concepts in computer science (Elspeet, 2003)*, volume 2880 of *Lecture Notes in Computer Science*, pages 93–105, 2003.

[11] P. S. Bonsma. Sparsest cuts and concurrent flows in product graphs. *Discrete Applied Mathematics*, 136(2-3):173–182, 2004.

[12] P. S. Bonsma, T. Brueggemann, and G. J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In *Mathematical foundations of computer science (Bratislava, 2003)*, volume 2747 of *Lecture Notes in Computer Science*, pages 259–268. Springer, Berlin, 2003.

[13] I. Z. Bouwer and G. F. LeBlanc. A note on primitive graphs. *Canadian Mathematical Bulletin*, 15:437–440, 1972.

[14] A. Brandstädt, F. F. Dragan, V. B. Le, and T. Szymczak. On stable cutsets in graphs. *Discrete Applied Mathematics*, 105(1-3):39–50, 2000.

[15] Y. Caro, D. B. West, and R. Yuster. Connected domination and spanning trees with many leaves. *SIAM Journal on Discrete Mathematics*, 13(2):202–211, 2000.

[16] V. Chvátal. *Linear Programming*. Freeman, New York, 1983.

[17] V. Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.

[18] D. G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004.

[19] K. Day and A. E. Al-Ayyoub. The cross product of interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):109–118, 1997.

[20] R. Diestel. *Graph Theory*. Number 173 in Graduate Texts in Mathematics. Springer-Verlag, New York, 1997.

[21] G. Ding, T. Johnson, and P. Seymour. Spanning trees with many leaves. *Journal of Graph Theory*, 37(4):189–197, 2001.

[22] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *Feasible mathematics, II (Ithaca, NY, 1992)*, volume 13 of *Progress in Computer Science and Applied Logic*, pages 219–244. Birkhäuser Boston, Boston, MA, 1995.

[23] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999.

[24] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In *Algorithms and complexity in Durham 2005*, volume 4 of *Texts in algorithmics*, pages 1–41, London, 2005. King's College Publications.

[25] A. M. Farley and A. Proskurowski. Networks immune to isolated line failures. *Networks*, 12(4):393–403, 1982.

[26] A. M. Farley and A. Proskurowski. Extremal graphs with no disconnecting matching. In *Proceedings of the second West Coast conference on combinatorics, graph theory, and computing (Eugene, 1983)*, volume 41 of *Congressus Numerantium*, pages 153–165, 1984.

[27] M. R. Fellows and M. A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM Journal on Discrete Mathematics*, 5(1):117–126, 1992.

[28] M. R. Fellows, C. McCartin, F. A. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems. In *FST TCS 2000: Foundations of software technology and theoretical computer science (New Delhi)*, volume 1974 of *Lecture Notes in Computer Science*, pages 240–251. Springer, Berlin, 2000.

[29] G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Information Processing Letters*, 52(1):45–49, 1994.

[30] G. Galbiati, A. Morzenti, and F. Maffioli. On the approximability of some maximum spanning tree problems. *Theoretical Computer Science*, 181(1):107–118, 1997. Latin American Theoretical INformatics (Valparaíso, 1995).

[31] M. R. Garey and D. S. Johnson. *Computers and intractability.* Freeman, San Francisco, 1979.

[32] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified $\mathcal{NP}$-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[33] R. L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York academy of sciences*, 175:170–186, 1970.

[34] J. R. Griggs, D. J. Kleitman, and A. Shastri. Spanning trees with many leaves in cubic graphs. *Journal of Graph Theory*, 13(6):669–695, 1989.

[35] J. R. Griggs and M. Wu. Spanning trees in graphs of minimum degree 4 or 5. *Discrete Mathematics*, 104(2):167–183, 1992.

[36] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[37] W. Hsu and T. Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM Journal on Computing*, 28(3):1004–1020, 1999.

[38] D. J. Kleitman and D. B. West. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics*, 4(1):99–106, 1991.

[39] V. B. Le and B. Randerath. On stable cutsets in line graphs. *Theoretical Computer Science*, 301:463–475, 2003.

[40] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[41] N. Linial and D. G. Sturtevant. unpublished result. 1987.

[42] H. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *Journal of Algorithms*, 29(1):132–141, 1998.

[43] D. W. Maluta and F. Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27:113–123, 1990.

[44] D. W. Matula. Concurrent flow and concurrent connectivity in graphs. In *Graph Theory with Applications to Algorithms and Computer Science (Kalamazoo, 1984)*, pages 543–559. Wiley, New York, 1985.

[45] S. L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9(5):229–232, 1979.

[46] A. M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13(5):527–536, 1989.

[47] T. Nieberg. *Independent and dominating sets in wireless communication graphs*. PhD thesis, University of Twente, Enschede, 2006.

[48] M. Patrignani and M. Pizzonia. The complexity of the matching-cut problem. In *Graph-theoretic concepts in computer science (Boltenhagen, 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 284–295, 2001.

[49] D. J. Rose. On simple characterization of k-trees. *Discrete mathematics*, 7:317–322, 1974.

[50] I. Rusu and J. Spinrad. Forbidden subgraph decomposition. *Discrete Mathematics*, 247(1-3):159–168, 2002.

[51] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, 1990.

[52] F. Shahrokhi and L. A. Székely. Uniform concurrent multicommodity flow in product graphs. *Congressus Numerantium*, 122:67–89, 1996.

[53] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In *Algorithms—ESA '98 (Venice)*, volume 1461 of *Lecture Notes in Computer Science*, pages 441–452. Springer, Berlin, 1998.

[54] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

Index

Summary

In this thesis, three different graph concepts are studied. A *graph* (V, E) consists of a set of *vertices* V and a set of *edges* E. Graphs are often used as a model for telecommunication networks, where the nodes of the network are represented by the vertices, and an edge is present between two vertices if the corresponding nodes are joined by a direct connection in the network. The two vertices joined by an edge are called its *end vertices*, and these two vertices are *neighbors* of each other. The *degree* of a vertex is its number of neighbors. The problems in this thesis can be explained and motivated using applications in the area of network design and analysis, which is done in Chapter 1. This chapter also contains the basic definitions that will be used.

The first problem we study is the problem of finding sparsest cuts of a graph. For a non-empty proper subset of the vertices $S \subset V$, $[S, \overline{S}]$ denotes the set of edges with one end vertex in S, which is an *edge cut*. An edge cut $[S, \overline{S}]$ is a *sparsest cut* if its *density* $\frac{|[S,\overline{S}]|}{|S||\overline{S}|}$ is minimum, over all edge cuts of the graph. Sparsest cuts are closely related to all-to-all flows in networks, and we also discuss an application related to network reliability. In general, the problem of finding sparsest cuts or calculating their density is $\mathcal{NP}$-hard. However, for three classes of well-structured graphs we characterize the sparsest cuts in Chapter 2. These classes are Cartesian product graphs, unit interval graphs and complete bipartite graphs.

Secondly, matching-cuts are studied. A *matching* is a set of edges that pairwise have no end vertices in common, and a *matching-cut* is an edge cut that is also a matching. The problem of deciding whether a graph admits a matching-cut is $\mathcal{NP}$-complete. In Chapter 3 we show that the problem remains $\mathcal{NP}$-complete when restricted to planar graphs with maximum degree four. We show that the problem can be decided in polynomial time for a number of graph classes such as claw-free graphs, co-graphs, outerplanar graphs, and graph classes with bounded treewidth in general.

In Chapter 4, graphs without a matching-cut are studied, which are called *(matching) immune*. Farley and Proskurowski proved that for all immune graphs on n vertices with m edges, $m \geq \lceil 3(n-1)/2 \rceil$, and constructed a large class of immune graphs attaining this lower bound for every value of n, called *ABC graphs*. In Chapter 4, we prove their conjecture that all matching immune graphs with $m = \lceil 3(n-1)/2 \rceil$ are ABC graphs.

The third topic of this thesis is spanning trees with many leaves. A *spanning tree* of a graph is a connected subgraph that contains all vertices, with a minimum number of edges. A *leaf* of a spanning tree is a vertex with degree one. Finding spanning trees with many leaves is another problem that often occurs in network design. In Chapter 5 we show that connected graphs on n vertices, without triangles, with minimum degree at least three, have a spanning tree with at least $(n+4)/3$ leaves. In addition we present a more general but weaker result; we show that connected graphs on n vertices, with minimum degree at least three, have a spanning tree with at least $(2n - D + 12)/7$ leaves, where D is the number of diamonds in the graph induced by degree three vertices (a *diamond* is a K_4 minus an edge). Both results are best possible and constructive.

In Chapter 6 we give an algorithm for deciding whether a graph on n vertices has a spanning tree with at least k leaves, using the second result from Chapter 5. This problem is $\mathcal{NP}$-complete. The complexity of the algorithm is $g(n) + f(k)$, where $g(n)$ is a polynomial and $f(k) \in O(8.12^k)$. It follows that this is a *Fixed Parameter Tractable* (FPT) algorithm, when k is viewed as the parameter of the problem. This is the current fastest FPT algorithm for this problem.

Samenvatting

In dit proefschrift worden drie verschillende grafentheoretische concepten behandeld. Een *graaf* (V, E) bestaat uit een verzameling *punten* V, en een verzameling *lijnen* E. Grafen worden vaak gebruikt om telecommunicatienetwerken te modelleren; in dat geval komen de punten overeen met de knopen uit het netwerk, en zijn twee punten verbonden door een lijn als de bijbehorende knopen direct verbonden zijn in het netwerk. De twee punten die verbonden worden door een lijn heten de *eindpunten* van de lijn, en deze twee punten zijn *buren* van elkaar. De *graad* van een punt is het aantal buren. De onderwerpen uit dit proefschrift kunnen worden uitgelegd en gemotiveerd door middel van toepassingen op het gebied van netwerkontwerp en -analyse. Dit wordt gedaan in Hoofdstuk 1. Dit hoofdstuk behandelt tevens de basisdefinities die gebruikt worden.

Het eerste probleem dat behandeld wordt is het probleem van het vinden van lichtste snedes in een graaf. Voor een niet-lege, echte deelverzameling van de punten $S \subset V$, noteert $[S, \overline{S}]$ de verzameling lijnen met precies één eindpunt in S. Een verzameling van deze vorm heet een *lijnsnede*. Een lijnsnede $[S, \overline{S}]$ is een *lichtste snede* als de *dichtheid* $\frac{|[S,\overline{S}]|}{|S||\overline{S}|}$ minimaal is, over alle lijnsnedes van de graaf. Lichtste snedes zijn gerelateerd aan bepaalde stromen in netwerken. We schetsen tevens een toepassing gerelateerd aan de betrouwbaarheid van netwerken. In het algemeen is het $\mathcal{NP}$-moeilijk om lichtste snedes te vinden, of hun dichtheid te berekenen. Desalniettemin karakteriseren we in Hoofdstuk 2 voor enkele graafklassen de lichtste snedes. Deze graafklassen zijn Cartesisch product grafen, eenheids-interval grafen en volledige bipartiete grafen.

Als tweede worden matching-snedes behandeld. Een *matching* is een verzameling lijnen die paarsgewijs geen eindpunten gemeen hebben, en een *matching-snede* is een lijnsnede die eveneens een matching is. Beslissen of een graaf een matching-snede heeft is een $\mathcal{NP}$-volledig probleem. In Hoofdstuk 3 tonen we aan dat dit probleem $\mathcal{NP}$-volledig blijft wanneer we de invoer beperken tot planaire grafen met grootste graad vier. Het probleem kan beslist worden in polynomiale tijd voor grafen zonder geïnduceerde $K_{1,3}$, voor co-grafen, voor buitenplanaire grafen, en in het algemeen, voor graafklassen met begrensde boombreedte.

In Hoofdstuk 4 worden grafen zonder matching-snede bestudeerd. Deze grafen heten *(matching) immuun*. Farley and Proskurowski hebben bewezen dat voor alle immune grafen met n punten en m lijnen geldt dat $m \geq \lceil 3(n-1)/2 \rceil$, en

hebben een graafklasse geconstrueerd die bestaat uit immune grafen die precies aan deze ondergrens voldoen, genaamd *ABC grafen*. In Hoofdstuk 4 bewijzen we hun vermoeden dat alle immune grafen met $m = \lceil 3(n-1)/2 \rceil$ ABC grafen zijn.

Het derde onderwerp van dit proefschrift is dat van opspannende bomen met veel bladeren. Een *opspannende boom* van een graaf is een samenhangende deelgraaf die alle punten bevat, met een minimaal aantal lijnen. Een *blad* is een punt met graad één. Het vinden van opspannende bomen met veel bladeren is wederom een probleem dat zich vaak voordoet in het ontwerpen van netwerken. In Hoofdstuk 5 bewijzen we dat samenhangende grafen met n punten, zonder driehoeken, met kleinste graad minstens drie, een opspannende boom hebben met minstens $(n+4)/3$ bladeren. We bewijzen ook een algemener maar zwakker resultaat; we tonen aan dat samenhangende grafen met n punten, met kleinste graad minstens drie, een opspannende boom bevatten met minstens $(2n-D+12)/7$ bladeren. Hierin staat D voor het aantal diamanten in de graaf die geïnduceerd worden door punten met graad drie (een *diamant* is een K_4 minus een lijn). Beide resultaten zijn constructief, en de gegeven grenzen zijn optimaal.

In Hoofdstuk 6 geven we een algoritme om te beslissen of een graaf op n punten een opspannende boom bevat met minstens k bladeren, gebruikmakend van het tweede resultaat in Hoofdstuk 5. Dit is een $\mathcal{NP}$-volledig probleem. De complexiteit van het algoritme is $g(n) + f(n)$, waarbij $g(n)$ een polynoom is, en $f(k) \in O(8.12^k)$. Dit is dus een *Fixed Parameter Tractable* (FPT) algoritme, wanneer we k beschouwen als de parameter van het probleem. Dit is op dit moment het snelste FPT algoritme voor dit probleem.

About the Author

Paul Bonsma was born on March 17, 1976 in Heerenveen, the Netherlands. After finishing secondary school at the RSG Heerenveen in 1994, he began studying applied mathematics at the University of Twente, Enschede, the Netherlands. In 2000 he graduated. The topic of his master's thesis was 'Algorithms for Routing and Wavelength Assignment in All-Optical Wide Area Networks', which was written under the supervision of Hajo Broersma. After spending a short period teaching at the University of Twente, he started his Ph.D. research in 2001. This was done at the Discrete Mathematics and Mathematical Programming Group, again supervised by Hajo Broersma, and by Gerhard Woeginger. The research was on various topics in graph theory and algorithms, and resulted in this thesis in 2006.