GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT
HANNOVER

MASTERARBEIT

Graphenparameter

Autor:
Thiemo FISCHER
Matrikelnummer:
2841650

Erstprüfer:
Prof. Dr. Heribert Vollmer
Zweitprüfer und Betreuer:
Dr. Arne Meier

am

Institut für Theoretische Informatik

10. Dezember 2020

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Unterschrift:

_____

Ort, Datum:

_____

GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT HANNOVER

Zusammenfassung

Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Abschlussarbeit zum Erwerb des Titels „Master of Science“

Graphenparameter

von Thiemo FISCHER

Viele Graphenalgorithmen, die in der klassischen Komplexitätstheorie den **NP**-vollständigen Problemen zugeordnet werden müssen, sind parametrisiert durch gewisse Graphenparameter in **FPT** und damit in Reichweite für effiziente Implementierungen. Jedoch sind die Bestimmung der Parameterwerte zu einem Graphen und die Konstruktion gut ausnutzbarer Graphenzerlegungen selbst schwierige algorithmische Probleme. Diese Arbeit präsentiert eine Schnittmenge aus algorithmischer Graphentheorie und parametrisierter Komplexität, um die Parameter Baumweite, Pfadweite, Zweigweite, Baumtiefe, Degeneriertheit, Cliquenweite und modulare Weite zu erläutern, verschiedene Charakterisierungen anzugeben und diese an Beispielen zu verdeutlichen. Darüber hinaus werden exakte Werte oder obere Schranken der Parameter auf verschiedenen Graphfamilien bewiesen. Approximationsalgorithmen für Baumweite und zugehörige algorithmische Konzepte für Teilprozeduren, Sätze, und Beweise bilden den Hauptteil der Ausarbeitung. Wir explorieren außerdem Optimierungen, die in praktischen Implementierungen eingesetzt werden, erläutern die Zusammenhänge zwischen – und die Vorzüge der verschiedenen Graphenparameter und diskutieren die Ergebnisse und Perspektiven der aktuellen Forschung.

GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT HANNOVER

Abstract

Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Thesis submitted in support of candidature for the academic degree
"Master of Science"

Graph Parameters

by Thiemo FISCHER

Decision problems to many algorithms on graphs are **NP**-complete by classical complexity analysis. However in the framework of parameterized complexity many of those problems allow **FPT** algorithms for certain graph parameters. Therefore for those problems wie are able to – or close to constructing feasible implementations that give resonably fast running times in practice. However the computation of the parameter's value as well as the construction of corresponding graph decompositions is a tough algorithmic challenge on its own. This thesis presents the field from the perspective of algorithmic graph theory and parameterized complexity. We give a precise introduction to the graph parameters tree-width, path-width, branch-width, tree-depth, degeneracy, clique-width and modular-width, show their different characterizations and illustrate them by examples. Furthermore some families of graphs take a constant value or allow upper bounds of the graphparameter. Some of the respective proofs are given in detail. However approximination algorithms for tree-width, the realization of needed subprocedures, related theorems and proofs comprise the main body of this thesis. We shall round of this exploration by concidering different algorithmic improvements that are used in practial implementations and compare the graph parameters and their merit to each other. Finally we canvass the results and prospects of the current research.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Viele Entscheidungsprobleme auf Graphen sind **NP**-vollständig. Nach unserem besten Wissen existieren keine Algorithmen. die solche Probleme in deterministischer polynomieller Laufzeit entscheiden können. Da Fragestellungen über Graphen in vielen Anwendungsgebieten natürlich aufkommen, besteht ein großes Interesse diese algorithmisch zu lösen.

Instanzen für die eine Fragestellung über Graphen besonders schwierig zu beantworten ist, bewirken die **NP**-schwere des zugehörigen Entscheidungsproblems. Ein universeller Algorithmus, der auch diese Instanzen löst, hat in der Regel auch auf einfachen Instanzen eine schlechte Laufzeit. Unsere grundlegende Strategie ist daher Teilmengen der Instanzen in einer Weise zu beschreiben, die sich geschickt algorithmisch ausnutzen lässt, um die Laufzeit gegenüber dem universellen Algorithmus zu verbessern.

Die Graphentheorie liefert eine reiche Sammlung gut untersuchter Grapheigenschaften wie Zusammenhang, Planarität, transitive Orientierbarkeit, Perfektion und vielen mehr. Sie eignen sich, um Teilmengen der Instanzmenge zu bilden und das Entscheidungsproblem auf diesen neu zu betrachten. Die in der Informatik vielleicht wünschenswerteste Eigenschaft eines Graphen wäre ein Baum zu sein. Viele auf allgemeinen Graphen **NP**-schwere Probleme wären auf Bäumen in polynomieller oder sogar linearer Zeit lösbar. Wir werfen die Frage auf, ob die Einschränkung auf welche Graphenfamilien, zu Algorithmen mit polynomieller Laufzeit führt, oder ob das Entscheidungsproblem dennoch **NP**-schwer bleibt. Das ISGCI[1] führt aktuell über 1500 Graphenfamilien für die wir diese Frage je Entscheidungsproblem stellen könnten. Die Suche nach Gemeinsamkeiten solcher Graphfamilien, welche **P**-Algorithmen erlauben, führt uns auf numerische, sogenannte *Graphenparameter*. Neben numerischen Grapheigenschaften, können auch algebraische Eigenschaften der Adjazenzmatrix oder die „Weite“ zugeordneter Strukturen und Graphzerlegungen Grundlage der Definition von Graphenparametern sein. Wir werden Beispiele für all diese Möglichkeiten kennen lernen. Zur Lösung von Graphenproblemen können wir Algorithmen heranziehen, die zusätzlich zur Graphinstanz eine obere Schranke des Graphenparameters oder ein Zertifikat i.e. einen kodierten Beweis für diese obere Schranke erhalten. Für so parametrisierte Algorithmen lassen sich bei kleinen Werten des Graphenparameters bessere Garantien für die Laufzeit beweisen, als dies in der klassischen Laufzeitanalyse der Fall ist.

[1] Information System on Graph Classes and their Inclusions, graphclasses.org

2 Grundlagen

Diese Arbeit fällt unter die Fachgebiete der algorithmischen Graphentheorie und der parametrisierten Komplexität. Die Graphentheorie ist ein traditionsreiches Forschungsgebiet der Mathematik. Spätestens seit den Arbeiten von Golumbic, 1980 [1] sowie Christofides, 1976 [2] werden die algorithmischen Aspekte der Graphentheorie intensiv erforscht. Viele in anderen Bereichen bekannte, für diese Arbeit relevante Definitionen, fallen mit graphentheoretischen Beschreibungen zusammen, weshalb die algorithmische Graphentheorie hier als geeigneter Ausgangspunkt dient.

Die parametrisierte Komplexität wurde als eigenständiges Forschungsfeld neben der „klassischen“ Komplexitätstheorie, in dem gleichnamigen Buch von Downey und Fellows im Jahr 1999 [3] etabliert. Mit ihr können die in dieser Arbeit vorgestellten Algorithmen präzise eingeordnet werden.

2.1 Graphentheorie

2.1.1 Grundbegriffe

Zunächst erinnern wir die folgenden dem Leser wohlbekannten Grundbegriffe aus der Graphentheorie. Ausführliche Einführungen können in den Standardwerken von Diestel oder Krumke und Noltemeier nachgeschlagen werden[4], [5].

Ein *ungerichteter Graph* ist ein Paar $G = (V, E)$ mit einer endlichen *Knotenmenge* V und einer *Kantenmenge* $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Die sogenannten *Knoten* aus der Menge V werden untereinander durch *Kanten* aus E verbunden. Ein *gerichteter Graph* ist ein Paar $G = (V, E)$ mit einer endlichen Knotenmenge V und einer Kantenmenge $E \subseteq V \times V$.

ungerichtet
Knoten
Kanten
gerichtet

Zwei Knoten u, v heißen *adjazent* , falls $\{u, v\} \in E$ beziehungsweise für gerichtete Graphen falls $(u, v) \in E$ oder $(v, u) \in E$. Wir schreiben für eine Kante auch $uv \in E$, falls aus dem Kontext klar ist, ob eine gerichtete Kante von u nach v oder eine ungerichtete Kante zwischen u und v gemeint ist. Zwei Kanten u und v heißen je *inzident* zur Kante uv. Zwei Kanten heißen *inzident*, falls sie einen gemeinsamen Knoten enthalten. Wir bezeichnen mit n die Anzahl der Knoten eines Graphen $|V|$. Analog notieren wir mit m die Anzahl der Kanten $|E|$. Die adjazenten Knoten zu u heißen *Nachbarn* von u. Sie bilden die Menge $N(u)$.

adjazent
inzident
Nachbarn

Eine Kante $vv \in E$ heißt *Schleife* . Ein ungerichteter Graph G ohne Schleifen (und ohne Mehrfachkanten[1]) heißt *einfach* .

Schleife
einfach

[1] Sog. Multigraphen erlauben eine Multimenge an Kanten. Multigraphen sind für diese Arbeit nicht relevant.

Teilgraph induziert

Wir schreiben $H \sqsubseteq G$ für einen sogenannten *Teilgraph* $H = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$. Selbiges G heißt Obergraph von H. Der durch W *induzierte Teilgraph* von G ist $G[W] := (W, \{ v_1v_2 \in E \mid v_1, v_2 \in W \})$.

$G - \{ v_1, \ldots, v_k \}$ bezeichnet den induzierten Teilgraphen $G[V \backslash \{ v_1, \ldots, v_k \}]$. Außerdem definieren wir den Teilgraphen $G - \{ e_1, \ldots, e_k \} := (V, E'')$ wobei $E'' := E \backslash \{ e_1, \ldots, e_k \}$. Bei einelementigen Mengen können die geschweiften Klammern weggelassen werden.

Beispiel 1 (Priorität beim Weglassen von Klammern). $G - uv$ bezeichnet den Graphen G ohne die einelementige Menge an Kanten $\{ uv \}$. $G - \{u, v\}$ bezeichnet den Graphen G ohne die Knoten u und v und alle zu diesen inzidenten Kanten.

In dieser Arbeit und der entwickelten Software werden hauptsächlich endliche[2] einfache Graphen behandelt. Wir betrachten die „Struktur" dieser Graphen. Zwei

isomorph

Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ heißen *isomorph* , geschrieben $G_1 \simeq G_2$, wenn es eine Bijektion $\phi: V_1 \to V_2$ gibt mit $xy \in E_1 \Leftrightarrow \phi(x)\phi(y) \in E_2$. Unter der Struktur eines Graphen verstehen wir seine Eigenschaften bis auf Isomorphie, also der „Umbenennung" der Knoten. Das bedeutet die Menge V dient lediglich zur Referenzierung von Knoten und diese können oft gleichwertig als $V = [1, n]_{\mathbb{N}} := \{ 1, 2, \ldots, n \}$, statt $V = \{v_1, \ldots, v_n\}$ angenommen werden. Graphinvarianten sind genau solche über Graphen definierte Größen, die sich nicht unter Umbenennung der Knoten verändern. Im folgenden werden einige Definitionen weiterer Grundbegriffe und Graphinvarianten festgehalten. $G = (V, E)$ sei im Folgenden ein einfacher endlicher Graph.

Grad Maximalgrad Minimalgrad

Der *Grad* $d(v)$ eines Knoten v ist die Anzahl seiner adjazenten Knoten $|N(v)|$. Der größte Grad eines Knoten in G heißt *Maximalgrad* $\Delta(G)$ von G. $\delta(G)$ bezeichnet analog den *Minimalgrad* von G.

unabhängige Menge

Eine Teilmenge $U \subseteq V$ heißt *unabhängige Menge* in G, falls alle $v, w \in U$ mit $v \neq w$ nicht adjazent in G sind. Eine unabhängige Menge heißt *maximal unabhängig*, falls sie keine echte Teilmenge einer größeren unabhängigen Menge in G ist. Die

Unabhängigkeitszahl

Unabhängigkeitszahl $\alpha(G)$ bezeichnet die Größe einer maximal unabhängigen Menge in G.

Knotenfärbung chromatische Zahl

Eine Abbildung $f: V \to \{1, \ldots k\}$ heißt *Knotenfärbung* mit k Farben, falls gilt: $\{v, w\} \in E \Rightarrow f(v) \neq f(w)$. Die *chromatische Zahl* χ oder auch Knotenfärbungszahl von G gibt die kleinste Anzahl an Farben an, die benötigt werden, um G zu färben; also $\chi(G) := \min\{ k \mid \text{es gibt eine Knotenfärbung von } G \text{ mit } k \text{ Farben.} \}$

Kantenfärbung

Analog heißt eine Abbildung $h: E \to \{1, \ldots k\}$ *Kantenfärbung* mit k Farben, falls gilt: Sind zwei verschiedene Kanten e und f inzident in G, so ist $h(e) \neq h(f)$. $\chi'(G) := \min\{ k \mid \text{es gibt eine Kantenfärbung von } G \text{ mit } k \text{ Farben.} \}$ heißt *chromatischer Index* oder auch Kantenfärbungszahl von G.

chromatischer Index

Weitere Graphinvarianten sind über besondere Teilgraphen definiert, die auch je für sich selbst von zentraler Bedeutung für die Graphentheorie und Graphenalgorithmen sind. Ausgewählte davon werden hier zusammengestellt:

[2] Die Knotenmenge ist endlich.

U sei eine Teilmenge von V. U heißt *k-partit* , falls U in k disjunkte Teilmengen $U_1 \uplus \cdots \uplus U_k$ partitioniert werden kann, sodass für alle i gilt: $u, v \in U_i$ impliziert $\{u, v\} \notin E$. Falls $U = V$ heißt auch G k-partit. Ist zusätzlich für jede Menge U_i jeder Knoten v zu jedem Knoten aus $U_1 \uplus \cdots \uplus U_{i-1} \uplus U_{i+1} \uplus \cdots \uplus U_k$ adjazent, so heißt U *vollständig-k-partit* . Falls $U = V = [1, n]_{\mathbb{N}}$ mit $n = \sum_i n_i$ gilt, so heißt $K_{n_1,\ldots,n_k}$ der vollständig k-partite Graph, für eine geeignete Zerlegung mit $|U_i| = n_i$ für jedes i wie oben.

k-partit

vollständig-k-partit

Pfade und Wege werden benutzt, um zu beschreiben, wie Teilgraphen untereinander verbunden sind:

Eine Folge $(v_1, \ldots, v_{n+1})$ von Knoten aus G, mit $v_i v_{i+1} \in E$ für $1 \leq i \leq n$, heißt *Weg* in G [3]. Sei W ein solcher Weg. Die Länge von W ist n. Ein Weg der keinen Knoten mehrfach besucht, heißt *Pfad* . P_k bezeichnet den Graphen mit Knotenmenge $[1, k+1]_{\mathbb{N}}$, der aus genau einem Pfad der Länge k besteht.

Weg

Pfad

Eine Folge $(v_1, \ldots, v_{n+1} = v_1)$ mit $v_i \in V$, $v_i v_{i+1} \in E$ und $v_i \neq v_j$ für alle $1 \leq i \leq n$, $1 \leq j \leq n$ mit $i \neq j$, heißt *Kreis* in G. Sei C ein solcher Kreis. Die Länge des Kreises C ist n. Eine Kante $v_i v_k \in E$ zwischen zwei nicht aufeinander folgenden Knoten $v_i, v_k \in C$ heißt *Sehne* . C_k bezeichnet den Graphen auf der Knotenmenge $[1, k]_{\mathbb{N}}$, der aus genau einem Kreis der Länge k besteht.

Kreis

Sehne

Eine Knotenteilmenge $C \subseteq V$ heißt *Clique* , wenn je zwei Knoten $v, w \in C$ mit $v \neq w$ in G adjazent sind. Eine Clique heißt *maximal* , wenn sie keine echte Teilmenge einer größeren Clique in G ist. Gibt es neben einer Clique C keine andere Clique höherer Kadinalität in G, so ist C eine *größte Clique* . Die Kardinalität einer größten Clique $|C|$ heißt *Cliquenzahl* $\omega(G)$ von G. Der Graph mit Knotenmenge $[1, n]_{\mathbb{N}}$, in dem alle Knoten gemeinsam eine Clique bilden, heißt *vollständiger Graph* und wird mit K_n notiert.

Clique

maximal

größte Clique

Cliquenzahl

vollständiger Graph

Ein kreisfreier einfacher Graph heißt *Wald* . Ein zusammenhängender Wald heißt *Baum* . Zusammenhängende Teilgraphen eines Baums oder Waldes nennen wir kurz *Teilbaum* . Knoten u eines Waldes mit $d(u) = 1$ heißen *Blätter*. Gelegentlich stellen wir einen Knoten des Baumes gesondert als *Wurzel* heraus. Sei r Wurzel eines Baumes mit Knoten w und $p = (r, \ldots, v, w, x, \ldots)$ ein Pfad von r zu einem Blatt, dann heißt x ein *Kind-* und v der *Elternknoten* von w. Einen Wald mit ausgezeichneten Wurzelknoten je Baum nennen wir *gewurzelt*.

Wald

Baum

Teilbaum

Blatt

Wurzel

Kindknoten

Elternknoten

„Partition“ bezeichnet in dieser Arbeit eine Menge $\mathcal{P}$, die disjunkte Knoten- oder Kantenteilmengen eines Graphen G enthält. Der Begriff „Zerlegung“[4] wird für solche Definitionen reserviert, bei denen über den Elementen einer Menge von Knoten- oder Kantenteilmengen eine zusätzliche Struktur verlangt wird. Die Adjektive „partitioniert“ und „zerlegt“ werden entsprechend verwendet.

[3] Für allgemeine Graphen wird der Begriff „Weg“ über eine Folge von Knoten und Kanten definiert. Dies ist hier nicht nötig, da die Kanten zwischen Knoten in einfachen Graphen eindeutig sind.

[4] sowie auch der dem Englischen entlehnte Begriff „Dekomposition“

2.1.2 Chordale Graphen

Um die Struktur von Graphen zu erfassen, sind „kleine“ Obergraphen interessant, in die ein Graph eingebettet werden kann. Ein Ziel könnte sein, mittels einer Einbettung eine Obergrenze für eine der zuvor genannten Invarianten zu etablieren. Eine oft verwendete Methode einen Obergraphen zu konstruieren, ist den Ausgangsgraphen zu chordalisieren:

Definition 1 (chordaler Graph, simplizialer Knoten, Chordalisierung, minimale Chordalisierung)**.** Sei $G = (V, E)$ ein ungerichteter Graph.

chordal
- G heißt *chordal* , wenn jeder Kreis C in G der Länge $k \geq 4$ mindestens eine Sehne hat.

simplizial
- Ein Knoten $v \in V$ heißt *simplizial* , falls $N(v)$ eine Clique in G ist.

Chordalisierung, chordale Vervollständigung
- Eine *Chordalisierung* oder *chordale Vervollständigung* eines Graphen $G = (V, E)$ ist ein chordaler Graph $H = (V, E')$ mit $E \subseteq E'$.
- Eine Chordalisierung heißt *minimal*, falls keine Kantenmenge E'' mit $E \subseteq E'' \subset E$ existiert, sodass (V, E'') eine Chordalisierung ist.

minimale Chordalisierung

Mit anderen Worten: Ein Graph ist chordal, wenn er keinen Kreis der Länge mindestens 4 als induzierten Teilgraphen enthält. In der Literatur werden chordale Graphen auch oft als triangulierte Graphen eingeführt. Dieser Begriff ist jedoch mehrdeutig, da maximal planare Graphen ebenfalls als trianguliert bezeichnet werden.

Jeder induzierte Teilgraph eines chordalen Graphen ist chordal. Um einen Graphen zu chordalisieren, können sukzessiv zusätzliche Kanten eingefügt werden, bis die Eigenschaft hergestellt ist. Es gibt im Allgemeinen sehr viele unterschiedliche Chordalisierungen. Die Konstruktion wird daher meißt vorbehaltlich eines weiteren Optimierungsziels durchgeführt, das durch eine der folgenden Fragen motiviert ist: Wie viele Kanten müssen wenigstens eingefügt werden, um den Graphen zu chordalisieren? Kann die Cliquenzahl der Chordalisierung möglichst klein gehalten werden? Erstaunlicherweise führt die Beantwortung dieser Fragen genau auf äqivalente Charakterisierungen von im nächsten Kapitel beschriebenen Graphenparametern.

Satz 1 (Dirac [6, Theorem 4])**.** *Jeder chordale Graph G enthält mindestens einen simplizialen Knoten. Falls G keine Clique ist, dann enthält G mindestens zwei nicht adjazente simpliziale Knoten.*

Um einen simplizialen Knoten v zu identifizieren, müssen lediglich die Adjazenzlisten der Nachbarn von v durchlaufen werden. Ein einfacher Algorithmus findet einen simplizialen Knoten in Zeit $\mathcal{O}(n^2 \cdot m)$. Zusammen mit der folgenden Definition und dem zugehörigen Satz erhalten wir eine einfache Charakterisierung für chordale Graphen.

Definition 2 (Perfektes Eliminationsschema)**.** Sei G ein ungerichteter Graph mit $|V| = n$. Ein perfektes Eliminationsschema für G ist eine bijektive Abbildung $\sigma: V \to \{1, \dots n\}$, sodass $\sigma^{-1}(i)$ simplizial in $G[\{\sigma^{-1}(i), \dots, \sigma^{-1}(n)\}]$ für alle $1 \leq i \leq n$ ist.

Satz 2 (Golumbic [1, Theorem 4.1.])**.**

- *Ein Graph G ist chordal, gdw. G besitzt ein perfektes Eleminationsschema.*
- *Falls ein Graph G chordal ist, so gibt es zu jedem simplizialen Knoten v von G ein perfektes Eliminationsschema für G, dass mit v startet.*

Mit einer lexikographischen Breitensuche kann ein Eliminationsschema in Zeit $\mathcal{O}(n+m)$ konstruiert werden [5]. Jeder Graph kann also in linearer Zeit auf Chordalität getestet werden.

Bemerkung 1. Außerdem erhalten wir mit einem perfekten Eliminationsschema für einen chordalen Graphen G auch dessen maximale Cliquen. Sei C eine maximale Clique in G und v_k das erste Auftreten eines Knotens von C im perfekten Eliminationsschema zu G. v_k ist simplizial in $G[\{v_{k+1}, \dots, v_n\}]$. Wir erhalten also alle weiteren Knoten der Clique im Schnitt $N(k) \cap \{v_{k+1}, \dots, v_n\}$. C hat die Form $C = \{v_k\} \cup (N(k) \cap \{v_{k+1}, \dots, v_n\})$. C ist beliebig gewählt, somit erhalten wir mit dem oben genannten Algorithmus auch jede maximale Clique und $\omega(G)$ in linearer Zeit.

Wir halten aus den bisherigen Beobachtungen fest, dass jeder Graph G chordalisiert werden kann und es einfach ist Cliquen in einer zugehörigen Chordalisierung zu finden. Des weiteren stellt sich heraus, dass neben dem Cliqueproblem, auch das Färbungs- und das Stabilitätsproblem für chordale Graphen in polynomieller Zeit lösbar sind [5]. Wie die minimalen Separatoren (siehe Abschnitt 2.1.4) einer chordalen Vervollständigung H eines Graphen G mit dessen minimalen Separatoren in Verbindung gebracht werden können, werden wir in Abschnitt 4.3 genauer untersuchen.

2.1.3 k-Zusammenhang

Um Aussagen über die Struktur eines Graphen zu treffen, interessieren uns neben möglichen Obergraphen auch besondere Teilgraphen. Ein erster einfacher Begriff drückt aus wie „schwierig“ es ist, in einem Graphen mögliche Pfade zwischen zwei beliebigen Knoten zu unterbrechen.

Definition 3 (zusammenhängend, Zusammenhangskomponente, k-zusammenhängend, Zusammenhangszahl)**.**

(1) Ein ungerichteter Graph $G = (V, E)$ heißt *zusammenhängend* , falls zu allen $v, w \in V$ ein Pfad zwischen v und w in G existiert. zusammenhängend

(2) Einen Teilgraphen $G[W]$ nennen wir *Zusammenhangskomponente* von G, falls zu W kein Knoten v hinzugefügt werden kann, so dass $G[W \cup v]$ noch zusammenhängend ist. Wir kürzen den Begriff mit *ZHK* ab. Zusammenhangskomponente

k-zusammenhängend

(3) Ein ungerichteter Graph G heißt *k-zusammenhängend* , falls für jede $(k-1)$-elementige Knotenteilmenge $X \subseteq V$ der Graph $G - X$ zusammenhängend ist. Wir definieren die k-zusammenhängenden Komponenten von G analog zu ZHKn.

Zusammenhangszahl

(4) Die *(Knoten-)Zusammenhangszahl* $\kappa(G)$ ist die größte natürliche Zahl k, für die G k-zusammenhängend ist.

Die Zusammenhangskomponenten eines Graphen $G = (V, E)$ lassen sich in linearer Zeit $\mathcal{O}(|V| + |E|)$ durch Tiefendurchlauf bestimmen [5].

Die Kontraposition von (3) lautet: Falls es für einen Graphen G eine Teilmenge X der Größe $k-1$ gibt, sodass $G - X$ in mehrere ZHK zerfällt, so ist G nicht k-zusammenhängend. Außerdem folgt aus der Prämisse: Es gibt einen Pfad in G, welcher in $G - X$ unterbrochen ist, sodass Start- und Endknoten dieses Pfades in unterschiedlichen ZHK von $G - X$ liegen. Die Menge X der „entfernten" Knoten respektive $G - X$ möchten wir nun benennen und gesondert untersuchen.

2.1.4 Knotenseparatoren

Die algorithmische Bestimmung von Grapheigenschaften wie der Cliquenzahl, der Unabhängigkeitszahl, oder der Knotenfärbungszahl in absehbarer Laufzeit ist eine Herausforderung. Die zugehörigen Probleme sind **NP**-vollständig und es ist nicht offensichtlich, wie sich die Probleminstanz auf kleinere unabhängige Teilprobleme zurückführen ließe. In der Graphentheorie kommen sogenannte Knotenseparatoren bzw. Trenner oder Kantenseparatoren bzw. Schnitte zum Einsatz, um Graphen in mehrere kleinere Graphen zu teilen. Nicht jeder Separator wird sich gleichermaßen für die Untersuchung der obigen Grapheigenschaften als hilfreich erweisen. Wie wir bestimmte Separatoren algorithmisch nutzen können, erfordert einer genauere Erörterung mit zugehörigen Lemmata und Sätzen. Sie folgt in Abschnitt 4.2. Der einfachste Begriff eines Separators lautet wie folgt:

Definition 4 ((a,b)-Separator, separierende Menge)**.**

(a,b)-Separator

- Eine Knotenmenge $S \subseteq V$ heißt (a,b)-*Separator* (auch (a,b)-Trenner oder Knotenschnitt), falls a und b in $G' = G - S$ in unterschiedlichen Zusammenhangskomponenten liegen. S separiert (trennt) die Knoten a und b. Die Größe des Seperators ist $|S|$.

separierende Knotenmenge

- Falls M für ein beliebiges Knotenpaar x, y ein x, y-Separator ist, heißt M auch *separierende Knotenmenge* [5].

Beispiel 2. In einem Graphen G mit $\kappa(G) = 1$ existiert ein Knoten v sodass $G - v$ nicht zusammenhängend ist. Ein solcher Knoten ist als Artikulationspunkt oder Gelenkpunkt bekannt. $\{v\}$ ist eine separierende Knotenmenge der Größe 1.

[5] „(a,b)-Separatoren" und „separierende Knotenmengen" beschreiben grundsätzlich die selbe Familie von Mengen. Die kanonische Definition von Minimalität führt jedoch zu unterschiedlichen optimalen Mengen.

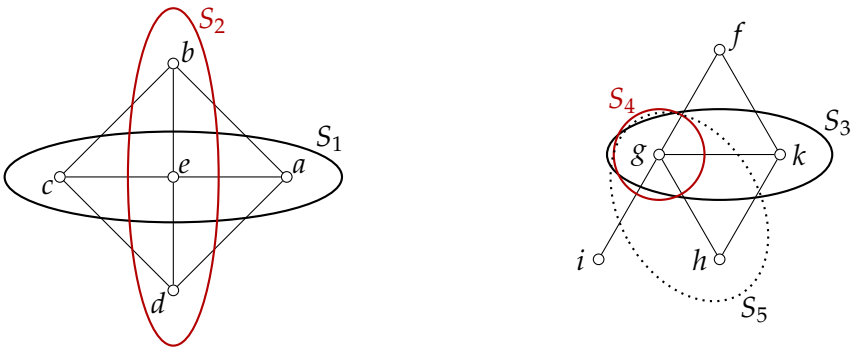
Ein Graph G ist k-zusammenhängend genau dann, wenn für ihn keine separierende Knotenmenge der Größe $k-1$ existiert. Adjazente Knoten können nicht separiert werden. a und b wie in der Definition oben, liegen in unterschiedlichen Zusammenhangskomponenten. Die Knotenmengen dieser Zusammenhangskomponenten sein A und B, mit $a \in A$ und $b \in B$. Ein (a,b)-Separator trennt nicht nur a und b, sondern jedes beliebige Knotenpaar aus $A \times B$. Wir können das selbe Objekt daher mit der Notation (A,B)-Separator benennen. Durch die Entfernung von S können auch weitere Zusammenhangskomponenten $G[C], G[D], \ldots$ entstehen. V kann in der Form $S \uplus A \uplus B \uplus C \uplus D \uplus \ldots$ ausgedrückt werden.

Definition 5 (minimaler (a,b)-Separator, minimale separierende Menge)**.**

- Ein (a,b)-Separator heißt *minimal* , falls keine echte Teilmenge ebenfalls ein (a,b)-Separator ist. (a,b) kann in der Notation entfallen. Δ_G sei die Menge aller minimalen Separatoren eines Graphen G [6]. minimaler Separator
- Ein minimaler Separator heißt *minimal* , falls keine echte Teilmenge ebenfalls ein minimaler Separator ist. minimale separierende Menge

Beispiel 3. In Abbildung 2.1 ist S_1 ein minimaler (b,d)-Separator, S_2 ist ein minimaler (a,c)-Separator. S_3 ist ein (f,i)-Separator und ein minimaler (f,h)-Separator. S_4 ist eine minimale separierende Menge und echte Teilmenge des minimalen Separators S_3. Ein Beispiel für einen nicht minimalen Separator ist S_5.

ABBILDUNG 2.1: Separatoren, separierende Mengen und Minimalität



Beachte, dass ein minimaler (a,b)-Separator eine echte Teilmenge eines anderen minimalen c,d-Separators sein kann. Das Paar (c,d) enthält notwendigerweise mindestens einen Knoten, der von a oder b verschieden ist.

Beispiel 4.

- Sei G_1 ein k-zusammenhängender Graph und X eine k-elementige Knotenteilmenge, sodass $G - X$ mindestens zwei ZHKn hat. X ist eine minimale separierende Menge. Alle minimalen Separatoren von G enthalten mindestens k Knoten.
- Sei G_2 ein Graph und *jede* minimale separierende Menge von G_2 habe mindestens k Knoten. Es folgt, G_2 ist k-zusammenhängend.

[6] Δ_G, nicht zu verwechseln mit $\Delta(G)$ dem Maximalgrad von G

Oft werden zusätzliche Eigenschaften von Separatoren gefordert oder gegeneinander abgewogen, wenn sie nicht gleichzeitig herzustellen sind: Es ist algorithmisch vorteilhaft Separatoren so zu wählen, dass die entstehenden Zusammenhangskomponenten ähnlich groß sind. Dazu werden Maße definiert, welche angeben wie „balanciert“ ein Separator ist. In rekursiven Algorithmen führen balancierte Separatoren heuristisch zu einer geringeren Rekursionstiefe. Auch wenn Minimalität nicht garantiert werden kann, sollen Separatoren in der Regel möglichst klein sein. Folgende Auswahl an Eigenschaften sind für diese Arbeit relevant.

Definition 6 (Eigenschaften von Separatoren)**.** Sei $G = (V, E)$ ein Graph und S ein a, b-Separator, der den Graphen in Knotenteilmengen A und B mit $a \in A$ und $b \in B$ trennt.

- S hat Größe höchstens k.
- (exakt) balanciert: S heißt *(exakt) balanciert* , falls jede ZHK von $G - S$ höchstens $\left\lceil \frac{|V|-|S|}{2} \right\rceil$ Knoten enthält [7].
- α-balanciert: S heißt *α-balanciert* , falls gilt: $V = A \uplus S \uplus B$ und $\max\{|A|, |B|\} \leq \alpha|V|$.
- α-balanciert in W: S heißt *α-balanciert in W* , falls gilt: $V = A \uplus S \uplus B$ und $\max\{|A \cap W|, |B \cap W|\} \leq \alpha|W|$.

Wir notieren diese Eigenschaften als Präfix „α, k, W-“ zu dem dies betreffenden Separator.

Beispiel 5 ($\frac{2}{3}, W$-Separator). Die in unserem Kontext am häufigsten betrachteten Separatoren sind $\frac{2}{3}, W$-Separatoren.

Einen α, k-Separator kann in Zeit $n^{\mathcal{O}(1)}\binom{n}{k}$ bestimmt werden. Diese Laufzeit wird erzielt, indem alle k-elementigen Knotenteilmengen durchsucht und anschließend die Balance überprüft wird. Falls k als konstant modelliert werden kann, lässt sich diese Laufzeit durch ein Polynom abschätzen. Im Allgemeinen gehört k jedoch zur Eingabe.

Das Ziel exakt balancierte Separatoren zu finden, muss in vielen prakischen Algorithmen aufgegeben werden [8]. Das Problem in einem gegebenen Graphen für $\frac{1}{2} \leq \alpha < 1$ einen α-Separator minimaler Größe zu finden, ist **NP**-schwer [9]. Dies gilt ebenso für α, k-Separatoren, über eine Reduktion ausgehend von k-CLIQUE [10] und unter üblichen Annahmen über **NP** auch für $k \in \mathcal{O}(\log n)$ [11]. Falls ein Separator der minimalen Größe k^* existiert, so kann ein angemessen kleiner Knotenseparator (mit Größe als Produkt eines polynomiellen Faktors in k^* und eines polylogarithmischen Faktors in n) in fast linearer Zeit approximiert werden [12]. Knoten-Separatoren sind essenziell für zahlreiche Graphenalgorithmen und viele praktische Anwendungen, wie die Lokalisierung von Flaschenhälsen in Netzwerken, Verteilung von Arbeitspaketen auf Rechnercluster, sowie die im nächsten Kapitel betrachteten Graphzerlegungen

[7] Manche Autoren verlangen, dass im Fall der exakten Balance die ganzzahlige Rundung $\lceil \cdot \rceil$ entfällt. Dies ändert die Größe eines exakt balancierten Separators um höchstens 1 [7]

[12]. Daher gibt es viel Forschungsarbeit zu Separatoren, die über die Komplexitäts- und Graphentheorie im engeren Sinne hinausgeht. In Approximationsalgorithmen sind die zu erzielende Größe und Balance des Separators die wesentlichen „Stellschrauben“ für die Laufzeit.

Zum Abschluss dieses Abschnittes führen wir einige der graphentheoretischen Grundlagen in einem kleinen gefälligen Satz über chordale Graphen zusammen und verwenden im Beweis die wiederholten Definitionen.

Satz 3. *Ein Graph G ist genau dann chordal, wenn jede seiner 2-zusammenhängenden Komponenten chordal ist.*

Beweis. $\Rightarrow$ Nach Definition ist jede 2-zusammenhängende Komponente Z, ein induzierter Teilgraph maximaler Größe von G, sodass kein 1-Separator aka. Artikulationspunkt in Z existiert. Jeder Kreis aus Z ist auch in G enthalten und hat eine Sehne, da G chordal ist und Z induzierter Teilgraph.

$\Leftarrow$ Alle 2-zusammenhängenden Komponenten Z_i der Größe mindestens 3 von G sind chordal. Aus der Definition wissen wir, dass genau die Artikulationspunkte von G nicht durch die Z_i abgedeckt werden, wenn wir den trivialen Fall einer 2-zusammenhängenden Kompoenente isomorph zu P_2 auszuschließen. Wie sehen maximale zusammenhängende induzierte Teilgraphen von G aus, welche nur aus Artikulationspunkten bestehen? Dies sind nach Definition genau Bäume. Alle Z_i sind höchstens durch induzierte Teilbäume von G untereinander kreisfrei verbunden. Andernfalls hätten wir einen Widerspruch zur Maximalität der Z_i. Wir folgern dass jeder Kreis in G in einem der Z_i enthalten sein muss. Da jeder Kreis in den Z_i eine Sehne besitzt, gilt dies für jeden Kreis von G.

□

2.2 Parametrisierte Komplexität

Viele relevante Entscheidungsprobleme sind **NP**-schwer. Falls **P** ≠ **NP** gilt, sind diese Probleme nicht effizient lösbar. Das bedeutet, falls **P** ≠ **NP**, so existiert zu **NP**-schweren Problemen kein Algorithmus, der *alle* Instanzen des Problems entscheidet und in polynomieller Zeit läuft.

Es besteht jedoch die Möglichkeit, dass in der Praxis relevante Instanzen in Teilmengen des allgemeinen Problems fallen und diese Teilmengen effizient gelöst werden können. Wenn beispielsweise alle Probleminstanzen einer Graphfamilie angehören, ist es möglich, dass für diese Familie bessere Algorithmen konstruiert werden könnten.

Dieser Ansatz kann so modelliert werden, dass der Algorithmus mit der Probleminstanz einen weiteren Parameter erhält. Solche Parametrisierungen ermöglichen unter anderem eine feinere Betrachtung klassisch **NP**-schwerer Probleme.

parametrisiertes Problem

Definition 7. Σ sei ein endliches Alphabet. Ein *parametrisiertes Problem* ist eine Menge $L \subseteq \Sigma^* \times \mathbb{N}$. Für eine Instanz $(x, k) \in \Sigma^* \times \mathbb{N}$ heißt k der Parameter[8]. N sei die Größe der Eingabe $|x|$.

Laufzeit und Platzbedarf können mit dieser Definition in Abhängigkeit von N und k ausgedrückt werden. In Anwendungen treffen wir häufig auf sehr große Instanzen. Für Exponentialzeitalgorithmen erreichen wir schnell Laufzeiten, die auf heutiger Hardware die Lebensdauer unseres Sonnensystems überschreiten. Diese Beobachtung motiviert die Definition von **FPT**.

FPT

Definition 8. Ein parametrisiertes Problem liegt in der Komplexitätsklasse **FPT** (engl. fixed-parameter tractable) , falls ein deterministischer Algorithmus M, eine berechenbare Funktion f und ein Polynom p existieren, sodass die Laufzeit von M in $f(k) \cdot n^{\mathcal{O}(1)}$ liegt und M das parametrisierte Problem entscheidet.

Die Definition erfasst folgende Intuition: Wir versuchen die Abhängigkeit der „Komplexität“ des Problems, gemessen an der Laufzeit eines zugehörigen Algorithmus, von der Eingabeinstanz in den Parameter zu verschieben. Wenn ein geeigneter Parameter k gefunden wird, der für Instanzen aus praktischen Anwendungen sehr klein gehalten werden kann, so ist der Algorithmus für diese Anwendung brauchbar. Zu beachten ist, dass ein Problem auf verschiedene Arten parametrisiert werden kann. Für den einen Parameter kann das parametrisierte Problem zu **FPT** und für einen anderen Parameter nicht zu **FPT** gehören.

Über einen Reduktionsbegriff zu **FPT** und eine spezielle Version des Erfüllbarkeitsproblems, kann eine Hierarchie an Komplexitätsklassen **FPT** = $\mathbf{W}[0] \subseteq \mathbf{W}[1] \subseteq \mathbf{W}[2]$ definiert werden. Einige **NP**-schwere Probleme für die keine **FPT**-Mitgliedschaft gezeigt werden kann, fallen mit geeigneter Parametrisierung in die niederigen Level der **W**-Hierarchie.

[8] Die Definition kann auf Parameter über einem beliebigen Alphabet ausgeweitet werden. Wir benötigen als Domäne nur die natürlichen Zahlen

Die Komplexitätsklasse **XP** enthält alle parametrisierten Sprachen L, sodass $L_k \in \mathbf{P}$ für jedes k. Im Vergleich zu **FPT** können sich die Algorithmen, welche die Mitgliedschaft $L_k \in \mathbf{P}$ zeigen, für jedes k unterscheiden. Es gilt $\mathbf{FPT} \subsetneq \mathbf{XP}$ [13, Proposition 27.1.1]. **XP**

Zu wünschenswerten Eigenschaften eines Parameters zählen, dass er auf möglichst vielen Eingaben kleine Werte annimmt, gut algorithmisch ausnutzbar ist und sich gut im Kontext der Anwendung interpretieren lässt. Aus einem klassischen Problem ein parametrisiertes Problem zu gewinnen, für den sich eine **FPT**-Laufzeit zeigen lässt, ist eine Aufgabe, die sowohl Kreativität als auch Kenntnis des Anwendungsfeldes erfordert. Für Probleme, deren Eingabeinstanzen Graphen sind, lassen sich geeignete Parameter oft an der Struktur der Graphen oder an Eigenschaften einer Lösung festmachen. Erstere nennen wir auch Strukturparameter. Die vorliegende Arbeit thematisiert Strukturparameter für Graphenprobleme, kurz *Graphenparameter* . Graphenparameter

Für **FPT**-Algorithmen ist f in der Regel[9] eine schnell wachsende Funktion, weshalb der Parameter k für praktische Belange möglichst klein sein sollte. Für ein festes k erhalten wir auf der so beschränkten Instanzmenge einen **P**-Algorithmus. Für Graphenfamilien mit beschränkter Baumweite können zu einigen klassisch **NP**-schweren Problemen parametrisierte **FPT**-Algorithmen gefunden werden [15].

Die klassischen Probleme CLIQUE, INDEPENDENTSET, CLIQUECOVER, und VERTEXCOLORABILITY[10] sind **NP**-vollständig [16]. Zugehörige Optimierungsprobleme führen auf die Graphinvarianten. ω, α, $\bar{\chi}$ und χ. Parametrisiert in der „Größe“ einer Lösung sind die Probleme CLIQUE, INDEPENDENTSET und CLIQUECOVER in **XP**, z.B. [17]. Für mindestens drei Farben k ist das Problem k-VERTEXCOLORABILITY bekannter Weise auf allgemeinen Graphen **NP**-schwer [18]. Daher ist die in k parametrisierte Variante von VERTEXCOLORABILITY weder in **XP** noch in **FPT**, falls $\mathbf{P} \neq \mathbf{NP}$.

[9] Falls die Exponentialzeithypothese[14] gilt, ergibt sich für die Parametrisierungen der meisten klassischen **NP**-schweren Probleme notwendigerweise ein mindestens exponentiell wachsendes f. Ausgenommen wenige **NP**-vollständige Probleme die nicht unter die Exponentialzeithypothese fallen e.g. Maximum Indipendent Set für Planare Graphen.

[10] Karp führt dieses Problem unter dem Namen CHROMATICNUMBER auf. Da die chromatische Zahl die Lösung zum kanonischne Minimierungsproblem MINVERTEXCOLORABILITY ist, verwenden wir hier den Begiff VERTEXCOLORABILITY für das Entscheidungsproblem.

3 Parameter und Zerlegungen auf ungerichteten Graphen

Im Grundlagenteil haben wir einige Eigenschaften von Graphen aufgeführt, die bekannter weise zu **NP**-vollständigen Problemen führen [16], jedoch zum Teil auf speziellen Obergraphen, den chordalen Vervollständigungen, in **P** gelöst werden.

Die Zusammenhangskomponenten eines nicht zusammenhängenden Graphen erhalten wir genau als die Komponenten zum Separator $\varnothing$ und in linearer Zeit. Für alle weiteren kleinen oder balancierten Separatoren müssen wir zum jetzigen Stand Approximationsalgorithmen hinnehmen, die je nach Größe und Balance bestenfalls in quasi-linearer Zeit und schlimmstenfalls in exponentieller Zeit bestimmt werden.

Im Verlauf dieses Kapitels werden wir erkennen, wie die grundlegenden Begriffe „Zusammenhang“ und „Separatoren“ in Graphzerlegungen eingesetzt oder sublimiert werden. Es werden verschiedene Zerlegungen vorgestellt, sowie zugehörige Graphenparameter, welche Aspekte der jeweiligen Zerlegung beschreiben, jedoch oft auch ohne diese definiert werden können. Chordale Obergraphen werden uns in verschiedensten Zusammenhängen begegnen.

3.1 Baum- und Pfadzerlegung

Baum- und Pfadweite sind die zur Zeit am besten untersuchten Graphenparameter. Wir führen die Begriffe zunächst über Baum- und Pfadzerlegungen ein. Im Anschluss werden wir einige äquivalente Definitionen zur Baum- und Pfadweite eines Graphen kennenlernen.

Definition 9 (Baumzerlegung)**.** Eine Baumzerlegung eines Graphen $G = (V, E)$ ist ein Paar $(\{X_i | i \in I\}, T = (I, F))$, wobei T ein Baum ist. $\{X_i | i \in I\}$ ist eine Menge von Teilmengen von V; eine für jeden Knoten von T, sodass gilt:

(i) $\bigcup_{i \in I} X_i = V$.

(ii) zu jeder Kante $(v, w) \in E$ gibt es ein $i \in I$, sodass $v \in X_i$ und $w \in X_i$.

(iii) zu allen $i, j, k \in I$: Falls j auf dem Pfad von i nach k liegt, dann gilt $X_i \cap X_k \subseteq X_j$.

Wir nennen die X_i Beutel (engl. bags). Letztere Forderung ist äquivalent zu:

(iii)′ Für alle $v \in G$ gilt: Die i zu solchen Behältern X_i, welche v enthalten, bilden gemeinsam einen Teilbaum T_v von T.

Nahe beieinander liegende Knoten (umgangssprachlich Cluster) eines Graphen, werden gemeinsam einem Beutel zugeordnet, sodass insbesondere jede Kante aus E in mindestens einem Beutel X enthalten ist[1]. Ein Knoten wird im Allgemeinen mehreren Beuteln zugehörig sein. Alle Beutel, die einen gemeinsamen Knoten enthalten, sind via der Relation F miteinander verbunden. Offensichtlich lassen sich alle Forderungen der Definition trivial erfüllen, indem ein Beutel der Größe $|V|$ gewählt wird, dem alle Knoten zugeordnet sind. Wir interessieren uns jedoch für Beutel die „gerade groß genug“ sind, sodass F eine nützliche und nicht triviale Struktur aufweist.

Definition 10 (Baumweite)**.**

- Die Weite einer Baumzerlegung $(\{X_i | i \in I\}, T = (I, F))$ ist $\max_{i \in I} |X_i| - 1$
- Die Baumweite $\mathbf{tw}(G)$ eines Graphen G ist die kleinste Baumweite über alle Baumzerlegungen von G.

Die Baumweite misst die „Ähnlichkeit“ eines beliebigen Graphen zu einem Baum. Graphen mit Baumweite 1 sind genau Wälder mit mindestens einer Kante. Wobei wir zumeist zusammenhängende Graphen betrachten und so Graphen der Baumweite 1 als Bäume erkennen.

Bemerkung 2. Die Weite einer Baumdekomposition zu G ist eine obere Schranke für $\mathbf{tw}(G)$. Für jeden Graphen G gilt $\mathbf{tw}(G) \leq n - 1$, da $\{\{X_1 = V\}, (\{1\}, \varnothing)\}$ eine gültige Baumzerlegung für G ist.

ABBILDUNG 3.1: Eine Baumzerlegung mit optimaler Weite $\mathbf{tw}(G_\triangle) = 2$

Ein Beispielgraph $G_\triangle$ (links) und eine Baumzerlegung zu $G_\triangle$ (rechts).

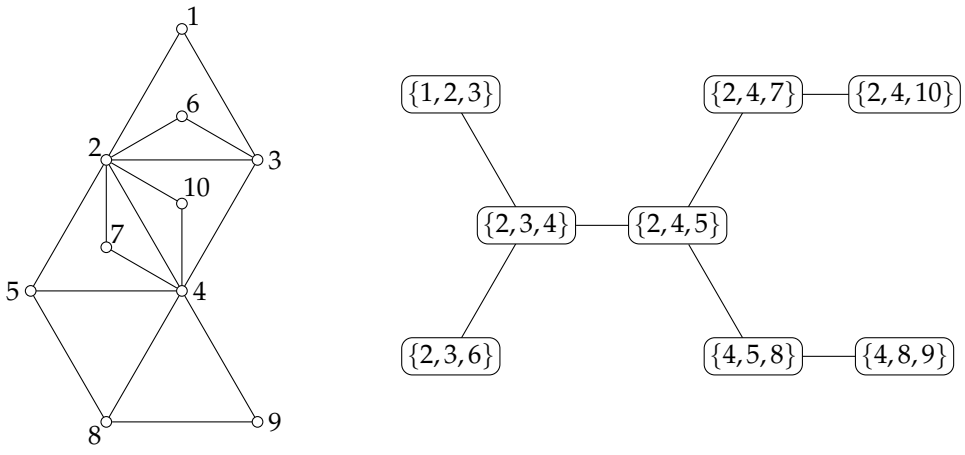
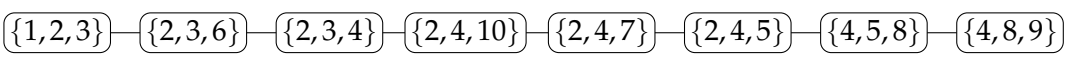


ABBILDUNG 3.2: Eine Pfadzerlegung mit optimaler Weite $\mathbf{pw}(G_\triangle) = 2$



[1] genauer in dem induzierten Teilgraphen $G[X]$

Natürlich können wir die Struktur der Zerlegung weiter spezialisieren: Die Einschränkung, dass die Teilbäume T_v in (iii)' insbesondere Pfade sein, wurde z.B. von Renz untersucht [19]. Verlangen wir darüber hinaus, dass die Beutel $\{X_i | i \in I\}$ sequenziell angeordnet werden können, i.e. der Baum T ein Pfad sei, erhalten wir folgende Definition:

Definition 11 (Pfadzerlegung)**.** Eine Pfadzerlegung eines Graphen $G = (V, E)$ ist eine Sequenz von Knoten-Teilmengen $(X_1, X_2, ..., X_r)$, sodass gilt:

- $\bigcup_{1 \leq i \leq r} X_i = V$.
- zu jeder Kante $(v, w) \in E$ gibt es ein $i, 1 \leq i \leq r$, mit $v \in X_i$ und $w \in X_i$.
- zu allen $i, j, k \in I$: Falls $i \leq j \leq k$, dann $X_i \cap X_k \subseteq X_j$.

Definition 12 (Pfadweite)**.**

- Die Weite einer Pfadzerlegung $(X_1, X_2, \ldots, X_r)$ ist $\max_{1 \leq i \leq r} |X_i| - 1$.
- Die Pfadweite $\mathbf{pw}(G)$ eines Graphen G ist die kleinste Weite über alle Pfadzerlegungen von G.

Bemerkung 3. Für jeden Graphen G ist seine Pfadweite eine obere Schranke seiner Baumweite, da jede Pfadzerlegung von G ebenfalls eine gültige Baumzerlegung ist.

$$\mathbf{tw}(G) \leq \mathbf{pw}(G)$$

Baumweite spielt eine zentrale Rolle in der Graphminorentheorie und wurde in diesem Zusammenhang durch Robertson und Seymour erstmalig eingeführt [20]. Viele graphentheoretische Konzepte sind entweder zu Baumweite oder Pfadweite ähnlich. Die unabhängige Forschung in den 1980er Jahren zu Baum-, Pfadweite und äquivalenten Definitionen wurde erst im Laufe der Zeit deutlich [15].

Beispiel 6. Aus jeder Baumzerlegung $(\mathcal{X}, (I, F))$ der Weite d lässt sich sehr einfach eine Pfadzerlegung konstruieren. Wähle einen Pfad p entlang $i_1 \ldots i_k$ zwischen zwei Blättern der Baumzerlegung so, dass die Menge der nicht durch p abgedeckten Knoten $V \backslash W$, mit $W = X_{i_1} \cup \ldots \cup X_{i_k}$, minimale Größe hat. Füge alle Knoten $V \backslash W$ aus Beuteln abseits des Pfades zu jedem Beutel auf dem Pfad p hinzu und nenne diese $X_i{}'$, dann ist $(X_i{}')_{i=i_1 \ldots i_k}$ eine Pfadzerlegung der Weite höchstens $d + |V \backslash W|$.

Andersherum ist jede Pfadzerlegung auch eine gültige Baumzerlegung.

Wir möchten nun untersuchen, ob und wie Cliquen, Pfade und Zusammenhangskomponenten von Graphen durch Baumdekomposition auf die Beutel $\mathcal{X}$ einer Baumzerlegung verteilt werden.

Lemma 1.

(i) Zu jeder Clique C eines Graphen G existiert ein Beutel X_{i_m} in jeder Baumzerlegung $(\{ X_i \mid i \in I \}, T = (I, F))$ des Graphen G, der C vollständig enthält.

(ii) Für die Cliquenzahl $\omega(G)$ von G gilt: $\omega(G) \leq \boldsymbol{tw}(G) + 1$.

Beweis.

(i) Für 1- und 2-Cliquen folgt die Aussage direkt aus Definition 9, (i) & (ii). Für Cliquen $C = (v_1, \ldots, v_k)$ der Größe $k \geq 3$ zeigen wir die Aussage durch vollständige Induktion: Nach Induktionsbehauptung existieren Beutel $X_{i_1}, \ldots X_{i_k}$, welche je die $k-1$-Cliquen $C\backslash\{v_j\}$ mit $1 \leq j \leq k$ enthalten.

Wir betrachten zwei verschiedene dieser Beutel X_{i_a} und X_{i_b}. Wir haben $C\backslash\{v_a\} \subseteq X_{i_a}$ und $C\backslash\{v_b\} \subseteq X_{i_b}$. Demnach $C\backslash\{v_a, v_b\} \subseteq X_{i_a}, X_{i_b}$.

Es folgt: X_{i_a} und X_{i_b} haben mindestens $k-2 \geq 1$ gemeinsame Knoten. Nach Def. 9 (iii) muss ein Pfad p_{i_a,i_b} zwischen i_a und i_b existieren. Auf dem Pfad p_{i_a,i_b} muss jeder Beutel mindestens die $k-2$ Knoten aus $C\backslash\{v_a, v_b\}$ enthalten.

Wählen wir einen weiteren Beutel X_{i_c} verschieden von X_{i_a} und X_{i_b}, so gilt selbiges für p_{i_a,i_c} und p_{i_b,i_c}. Die Pfade p_{i_a,i_b}, p_{i_a,i_c} und p_{i_b,i_c} haben darüber hinaus mindestens einen gemeinsamen Knoten i_m in der Baumzerlegung. Andernfalls könnten wir einen Kreis in T angeben; im Widerspruch dazu, dass T ein Baum ist. Für den zugehörigen Beutel X_{i_m} gilt

$$C \subseteq C\backslash\{v_a, v_b\} \cup C\backslash\{v_a, v_c\} \cup C\backslash\{v_b, v_c\} \subseteq X_{i_m}$$

(ii) Folgt für eine maximale Clique direkt aus *(i)*.

□

Wir können Lemma 1 als Verallgemeinerung der zweiten definierenden Eigenschaft von Baum- und Pfadzerlegungen verstehen, welche verlangt dass alle 2-Cliquen in einem Beutel X_i enthalten sind.

Lemma 2. *Sei $(\{X_i \mid i \in I\}, T = (I, F))$ eine Baumzerlegung des Graphen $G = (V, E)$. Zu $i_1, i_q \in I$ gelte für Knoten $v, w \in V$, dass $v \in X_{i_1}$ und $w \in X_{i_q}$. Für jeden Beutel X_{i_m} entlang des $i_1 i_q$-Pfades Ψ in T gilt: X_{i_m} enthält mindestens einen Knoten aus jedem vw-Pfad in G.*

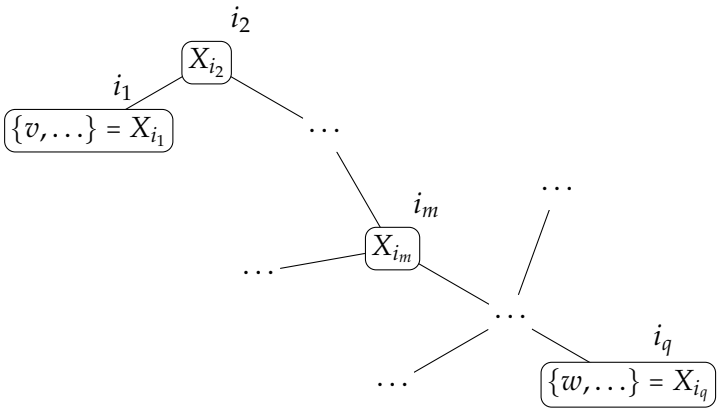
Beweis. Falls v und w aus unterschiedlichen ZHK gewählt sind, gilt die Aussage trivialer Weise.

Im Folgenden sein v und w aus der selben ZHK. Wir betrachten einen beliebigen vw-Pfad $p = (v = u_1, u_2, \ldots u_{q-1}, u_r = w)$ aus G. Sei $\mathcal{T}$ die Menge aller Teilbäume $T_v, T_{u_2}, \ldots T_{u_{r-1}}, T_w \subseteq T$ wie in Def. 9 (iii)' zu den Knoten aus p. Jeder Teilbaum $T_u \in \mathcal{T}$ enthält genau alle Beutel, die wiederum u enthalten. Im Pfad p aufeinander folgende Knoten u_k, u_{k+1} sind adjazent. Es folgt mit Def. 9 (ii): Die Teilbäume T_{u_k}, T_{u_k+1} haben einen nichtleeren Schnitt, welcher genau aus solchen Beuteln besteht, die beide Knoten u_k und u_{k+1} enthalten. Außerdem gilt nach Voraussetzung $i_1 \in T_v$ und $i_q \in T_w$.

Insgesamt gilt: Die Vereinigung aller Teilbäume aus $\mathcal{T}$ enthält einen $i_1 i_q$-Pfad. Da Pfade in Bäumen eindeutig sind, handelt es sich hierbei genau um den Pfad Ψ aus T. Demnach enthält jeder Beutel entlang des Pfades Ψ mindestens einen Knoten aus dem vw-Pfad p. Dies gilt für jeden vw-Pfad, da wir p beliebig gewählt haben. □

ABBILDUNG 3.3: Pfad $\Psi = (i_1 \dots i_q)$ in einer Baumzerlegung $(\{ X_i \mid i \in I \}, T = (I, F))$

Benennung aller Knoten und Beutel wie in Lemma 2. Jeder Knoten i aus T kann beliebige weitere Nachbarn außerhalb des Pfades Ψ besitzen. Für triviale Pfade der Länge 0 oder 1 gilt $i_m = i_1 \vee i_m = i_q$.



Für nicht triviale Pfade erhalten wir folgendes Lemma.

Korollar 1. *Sei $v, w \notin X_{i_m}$ zusätzlich zu den Voraussetzungen von Lemma 2. Dann ist X_{i_m} ein (v, w)-Separator von G.*

Beachte, dass nicht jeder *insbesondere minimale* (v, w)-Separtor aus Knoten der Vereinigung der Beutel entlang eines solchen Pfades besteht.

Lemma 3. *Der $x \times y$-Gittergraph hat Baumweite und Pfadweite* $\min(x, y)$

Beweis. Wir zeigen hier die Existenz einer passenden Zerlegung und verweisen auf [21] für die Optimalität dieser Weite. Wir definieren die Knoten des Gittergraphen als v_{ij} mit $1 \le i \le x$ und $1 \le j \le y$ wobei $v_{ij}v_{i(j+1)}$ und $v_{ij}v_{(i+1)j}$ durch Kanten verbunden sind[2]. Sei o.B.d.A. $y \le x$. Konstruiere die Pfadzerlegung $\mathcal{X} = (X_k)$ wie folgt:

$$
\begin{aligned}
X_1 &= \{ v_{11}, v_{12}, \dots, v_{1(y-1)}, v_{1y}, v_{21} \} \in \mathcal{X} \\
X_2 &= \{ v_{12}, v_{13}, \dots, v_{1y}, v_{21}, v_{22} \} \in \mathcal{X} \\
X_3 &= \{ v_{13}, v_{14}, \dots, v_{21}, v_{22}, v_{23} \} \in \mathcal{X} \\
X_4 &= \{ v_{14}, v_{15}, \dots, v_{22}, v_{23}, v_{24} \} \in \mathcal{X} \\
&\dots \\
X_{x\cdot(y-1)-1} &= \{ v_{(x-1)y}, v_{x1}, \dots, v_{x(y-2)}, v_{x(y-1)}, v_{xy} \} \in \mathcal{X}
\end{aligned}
$$

Wir orientieren den Gittergraphen wie eine Matrix. Beachte, dass alle vertikalen Kanten des Gittergraphen durch je einen der Beutel X_i abgedeckt sind. Offensichtlich sind alle Knoten und alle horizontalen Kanten in der Pfadzerlegung enthalten. Die Pfadzerlegung eingeschränkt auf Beutel die einen bestimmten Knoten enthalten sind zusammenhängend. $\mathcal{X}$ hat Weite $y = \min(x, y)$ □

[2] wähle die passenden Schranken

Tatsächlich zeigen Robertson, Seymour und Thomas darüber hinaus, dass jeder planare Graph mit Baumweite k einen $k \times k$-Gittergraph als Minor enthält [21].

3.1.1 TREEWIDTH und verwandte Entscheidungsprobleme

Für einen gegebenen Graphen $G = (V, E)$ und eine gegebene Konstante k ist das klassische Entscheidungsproblem, ob G Baumweite höchstens k hat, **NP**-vollständig [22]. Selbiges gilt für die Pfadweite [22]. Daraus ergaben sich Forschungsansätze, die mit verschiedenen Einschränkungen des Problems und Approximationsverfahren befasst sind. Natürlich werden auch beide Herangehensweisen in Kombination betrachtet.

Definition 13. Eine Graphenfamilie $\mathcal{C}$ hat beschränkte Baumweite bzw. beschränkte Pfadweite, falls ein k existiert, sodass für alle $G \in \mathcal{C}$ gilt: $\mathbf{tw}(G) \leq k$ bzw. $\mathbf{pw}(G) \leq k$.

Viele gut erforschte Graphfamilien haben beschränkte Baum- und Pfadweite: Bäume haben eine Baumweite von höchstens 1, Serien-Parallele-Graphen und außenplanare Graphen haben eine Baumweite von höchstens 2. Die stärkere Aussage, dass Serien-Parallele-Graphen genau die Graphen mit Baumweite höchstens 2 sind, gilt nicht: Der Graph $K_{1,3}$ hat Baumweite 1, ist jedoch nicht serien-parallel. Die Baumweite von planaren Graphen ist beschränkt durch $\alpha\sqrt{n}$ für eine Konstante $\alpha < 3{,}182$ [23]. Für weitere Graphfamilien beschränkter Baumweite siehe [24] oder [25, Tabelle 1]. Wir erwähnen auch die Einschränkung des Baumweiteproblems auf planare Graphen. Die Komplexität des Problems PLANARTREEWIDTH ist bis heute nicht bekannt:

> PLANARTREEWIDTH
> Eingabe: Ein ungerichteter planarer Graph $G = (V, E)$ und eine natürliche Zahl k.
> Frage: Hat G Baumweite höchstens k?

In Anwendungsfeldern wie Straßennetzwerken oder dem Schaltkreisentwurf ist die Baumweite oft klein genug oder die Instanz ähnlich genug zu einer Graphenfamilie beschränkter Baumweite, dass Baumzerlegungen geringer Weite mit exakten Algorithmen auf kleinen Instanzen, oder mit Approximationsalgorithmen, in annehmbaren Laufzeiten gefunden werden.

In der Entwicklung von Approximationsalgorithmen muss in der Regel eine Abwägung zwischen Laufzeit und Güte der Lösung getroffen werden. Wählt man ein beliebiges festes k, so lässt sich zeigen, dass ein $\mathcal{O}(n \log n)$-Algorithmus existiert, der auf Eingabe $G = (V, E)$ entweder feststellt, dass die Baumweite $\mathbf{tw}(G)$ größer als k ist oder eine Baumzerlegung von G mit Baumweite höchstens $3k + 2$ ausgibt, mit $k \leq \mathbf{tw}(G)$ [25]. Für viele der Algorithmen, die in theoretischen Arbeiten konstruiert werden, um Laufzeitschranken für die Bestimmung der Baum- und Pfadweite zu zeigen, sind weitere Verbesserungen nötig, um diese praktisch Einzusetzen [25], [26].

An der theoretischen Forschungsfront bewies Bodlaender 1996 die Mitgliedschaft des zugehörigen parametrisierten Problems in **FPT** durch einen Algorithmus mit linearer Laufzeit.

TREEWIDTH
Eingabe: Ein ungerichteter Graph $G = (V, E)$.
Parameter: Eine natürliche Zahl k.
Frage: Hat G Baumweite höchstens k?

Satz 4 (Satz von Bodlaender [27])**.** *Für jedes k existiert ein deterministischer Algorithmus welcher in linearer Zeit arbeitet und Graphen der Baumweite höchstens k erkennt. Falls der gegebene Graph Baumweite k hat, produziert der Algorithmus eine Baumzerlegung der Weite k.* TREEWIDTH *ist linear in* **FPT**.

Obwohl Bodlaender's Algorithmus in linearer Zeit arbeitet, ist er für die Praxis aufgrund sehr großer konstanter Faktoren, die sich in der Laufzeitfunktion verbergen, unbrauchbar. In der parametrisierten Komplexitätstheorie können auch exakte Algorithmen in ihrer Laufzeitabhängigkeit von gewissen Parametern verglichen werden. Es werden exakte parametrisierte **FPT**-Algorithmen gesucht, in denen $f(k)$ möglichst langsam wächst.

Downey und Fellows befürworten einen Begriff der *parametrisierten Approximation*, welcher die wissenschaftliche Methodologie für Baumweite und verwandte Graphenparameter weiter verfeinert. „parameterized approximation [is] a relatively new, topic of intense interest. It was introduced independently by three papers presented at the same conference“ [13].

3.1.2 Zwei Algorithmen zur Baumweiteapproximation durch Separatoren

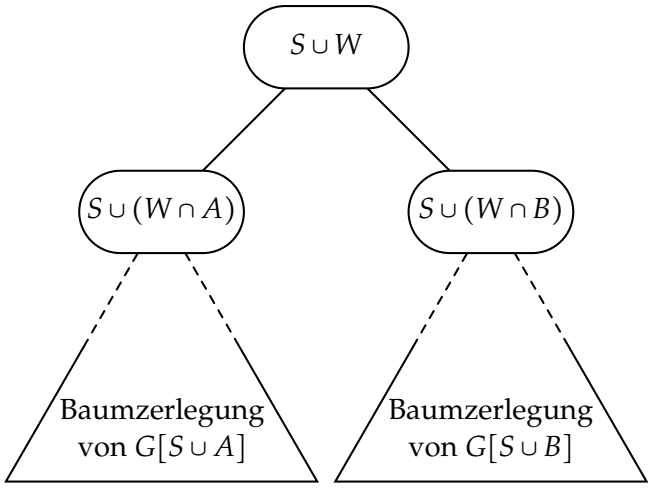
Wir möchten nun einen einfachen Algorithmus angeben, welcher eine Baumzerlegung für ungerichtete Graphen G durch „Separation in Komponenten“ findet. Der Algorithmus sucht rekursiv Separatoren, die über eine „Verbindungsmenge“ W zu einer Baumzerlegung zusammengesetzt werden. Die Existenz und Konstruktion dieser Separatoren zeigten Robertson und Seymour in [28]. Wir betrachten die Formulierungen des Algorithmus in [29], [30] und Vervollständigen die dort verkürzt dargestellten Korrektheitsbeweise. Der hier dargestellte Ansatz wurde in vielen weiteren Algorithmen auf äquivalenten oder ähnlichen Modellen für Baumweite erfolgreich eingesetzt [31]–[34].

Algorithmus 1 (BUILDTREEDECOMPOSITION)**.**

Eingabe: Graph $G = (V, E)$, Knotenteilmenge W der Größe höchstens $3k + 3$, natürliche Zahl k

Ausgabe: Baumdekomposition $(\mathcal{X}, (I, F))$ der Weite höchstens $4k + 3$ oder Ablehnen, falls $\mathbf{tw}(G) > k$

ABBILDUNG 3.4: Baumdekomposition zu Algorithmus BUILDTREE-DECOMPOSITION



1: **if** $V = W$ **then return** Baumdekomposition aus einem Beutel, der alle Knoten enthält.
2: $(S, A, B) \leftarrow$ FINDBALANCEDPARTITION(G, W, k)
3: **if** kein solcher Separator gefunden **then**
4: **Ablehnen**, „Die Baumweite von G ist größer als k“.
5: **if** ($A = \varnothing$ oder $B = \varnothing$) und $S \subseteq W$ **then**
6: Füge einen Knoten aus $V \backslash W$ zu S hinzu.
7: $(\mathcal{X}_A, (I_A, F_A)) \leftarrow$ BUILDTREEDECOMPOSITION$(G[S \cup A], S \cup (W \cap A), k)$.
8: $(\mathcal{X}_B, (I_B, F_B)) \leftarrow$ BUILDTREEDECOMPOSITION$(G[S \cup B], S \cup (W \cap B), k)$.
9: **if** einer der beiden rekursiven Aufrufe lehnt ab. **then**
10: **Ablehnen**, „Die Baumweite von G ist größer als k“.
11: $(\mathcal{X}, T = (I, F)) \leftarrow (\mathcal{X}_A \uplus \mathcal{X}_B, (I_A \uplus I_B, F_A \uplus F_B))$
12: Füge dem Baum (I, F) einen neuen Knoten r mit Kanten zu den Wurzelknoten von (I_A, F_A) und (I_B, F_B) hinzu.
13: Füge einen neuen Beutel $X \leftarrow S \cup W$ zu $\mathcal{X}$ mit Bezeichner r hinzu. **return** $(\mathcal{X}, T)$

Der Algorithmus FINDBALANCEDPARTITION findet einen balancierten W-Separator und erzeugt zwei voneinander getrennte Knotenmengen A und B. Wir können verschiedene Implementierungen benutzen, um diese Berechnung auszuführen. Einen Überblick erhalten wir in Abschnitt 4.2.

Voraussetzung 1. Der Algorithmus FINDBALANCEDPARTITION(G, W, k) berechnet einen $\frac{2}{3}, k+1, W$-Separator in Zeit $\mathcal{O}(\,3^{|W|} \cdot k^{\mathcal{O}}(\,1\,) \cdot (n+m)\,)$, falls G Baumweite höchstens k hat und lehnt sonst ab.

Wir beweisen die Korrektheit und Laufzeit des Algorithmus in drei Lemmata:

Lemma 4. *Sei $G = (V, E)$ ein Graph mit Baumweite höchstens k. Der Algorithmus* BUILDTREEDECOMPOSITION *berechnet eine gewurzelte Baumdekomposition von G der Weite höchstens $4k + 3$. Die Menge W ist im Beutel zum Wurzelknoten enthalten.*

Beweis. Falls $V = W$, so hält der Algorithmus. Nach Voraussetzung des Satzes und Voraussetzung 1 erreichen wir Schritt 4: Seien $A \neq \varnothing \neq B$. Dann reduziert sich die Problemgröße der rekursiven Aufrufe mit $G[S \cup B]$ und $G[S \cup A]$. Falls $A = \varnothing$ und $S \subseteq W$ so gilt nach Schritt 4: $S' \leftarrow S \cup \{v\}$, $S' \cup A = S'$, $S' \cup B = V$ und $|S' \cup (W \cap A)| = |S' \cup \varnothing| = |S \cup \{v\}|$ sowie $|S' \cup (W \cap B)| = |S' \cup W| = |W \cup \{v\}|$ für ein Knoten v. Für den rekursiven Auruf in Schritt 7 erhalten wir im ersten Argument $G[S' \cup B] = G$ und die Größe des zweiten Argumentes steigt um 1. Schließlich reduziert sich in einem rekursiven Aufruf die Größe von G oder er terminiert in Schritt 1. Analog für $B = \varnothing$. Falls $S \notin W$, so steigt ebenfalls die Größe des 2. Argumentes und der Algorithmus terminiert analog. BUILDTREEDECOMPOSITION terminiert.

Wir zeigen die Aussage mit vollständiger Induktion:

IA. Falls die verbleibende Rekursionstiefe 0 beträgt, so terminiert BUILDTREEDECOMPOSITION in Schritt 1. Die Ausgabe mit Beutel X_1 ist offensichtlich eine Baumdekomposition (Bemerkung 2) und hat die geforderte Weite: $|X_1| - 1 = |V| - 1 = |W| - 1 =\leq 3k + 3 \leq 4k + 3$.

IS. Die verbleibende Rekursionstiefe beträgt mindestens 1. Es folgt $W \neq V$. Nach Voraussetzung des Satzes und Voraussetzung 1 gilt $|W \cap A| \leq \left\lfloor \frac{2}{3}|W| \right\rfloor \leq \left\lfloor \frac{2}{3}(3k+4) \right\rfloor = 2k + 2$. Sei zunächst die Bedingung in Schritt 4 nicht erfüllt. Aus $|S| \leq k + 1$ folgt $|S \cup (W \cap A)| \leq 3k + 3$. Analog $|S \cup (W \cap B)| \leq 3k + 3$. Andernfalls gilt nach Schritt 4: $S \subseteq W$. Aufgrund $V \neq W$ haben wir entweder $A = \varnothing$ oder $B = \varnothing$. Wir bezeichnen den durch Schritt 5 modifizierten Separator mit S'. Es folgt $S' \leftarrow S \cup \{v\}$ mit $v \notin S \cup W$ und o.B.d.A. $v \in A$. Daraus folgt $v \notin W \cap A$. Wir können die Größe des zweiten Arguments der rekursiven Aufrufe wie folgt abschätzen: $|S' \cup (W \cap A)| \leq |S \cup (W \cap A)| \leq |S \cup W| = |W| \leq 3k + 3$ und $|S' \cup (W \cap B)| \leq |S' \cup \varnothing| = |S| + 1 \leq k + 2$.

Nach IV. erhalten wir Baumdekompositionen $(\mathcal{X}_A, T_A = (I_A, F_A))$ von $G[S \cup A]$ und $(\mathcal{X}_B, T_B = (I_B, F_B))$ von $G[S \cup B]$ der Weite höchstens $4k + 3$. Die Beutel zu den Wurzelknoten r_A und r_B enthalten $S \cup (W \cap A)$ und $S \cup (W \cap B)$, siehe auch Abbildung 3.4.

Sei $vw \in E$. Da keine Kanten zwischen A und B existieren, gilt entweder $v, w \in A \cup S$ oder $v, w \in B \cup S$. Sei auch $u \in V$. Da $\mathcal{X}$ die disjunkte Vereinigung von Beuteln der gültigen Baumdekompositionen zu $G[S \cup A]$ und $G[S \cup B]$ enthält und $A \cup S' \cup B = V$ gilt, ergibt sich: u ist in einem Beutel aus $\mathcal{X}$ enthalten und v, w ist in einem Beutel aus $\mathcal{X}$ enthalten.

Sei $u \in V$. Falls $u \notin S$ und u nur in $G[S \cup A]$ vorkommt, so ist T_u Teilbaum von T_A, also auch Teilbaum in T. Analog für $G[S \cup B]$.

Nun gelte $u \in S$. Es gilt $T_u \cap T_A \sqsubseteq T$ und $T_u \cap T_B \sqsubseteq T$ unabhängig davon ob u tatsächlich in den zugehörigen Baumdekompositionen enthalten ist. Wir erhalten: T_u ist ein Teilbaum von $T = (I_A \uplus I_B \uplus \{r\}, F_A \uplus F_B \uplus \{rr_A, rr_B\})$.

Insgesamt gilt: $(\mathcal{X}, (I, F))$ ist eine Baumdekomposition der Weite höchstens $4k+3$ und der Beutel zu r enthält W als Teilmenge. □

Lemma 5. *Falls der Algorithmus* BUILDTREEDECOMPOSITION *die Ausgabe* $\boldsymbol{tw}(G) \geq k$ *produziert, dann ist diese Aussage korrekt.*

Beweis. In rekursiven Aufrufen betrachten wir stets induzierte Teilgraphen H von G. Falls BUILDTREEDECOMPOSITION in Schritt 3 oder 8 ablehnt, so wurde für H in Schritt 2 kein Separator gefunden. Es gilt $\mathbf{tw}(H) \geq k$ nach Voraussetzung 1 und somit $\mathbf{tw}(G) \geq \mathbf{tw}(H) \geq k$. □

Lemma 6. *Der Algorithmus* BUILDTREEDECOMPOSITION *arbeitet in* **FPT***-Laufzeit.*

Beweis. FINDBALANCEDPARTITION in Schritt 2 läuft in Zeit $f(k) \cdot n^{\mathcal{O}(1)}$ für ein berechenbares f, nach Voraussetzung 1. Bis auf die rekursiven Aufrufe arbeiten alle weiteren Schritte in Zeit polynomiell in $n+m$. Die Anzahl rekursiver Aufrufe beträgt höchstens n^2, da mit allen rekursiven Aufrufen, in denen o.B.d.A. $|S \cup A|$ nicht strikt fällt, das zweite Argument bis höchstens n anwächst: $|S \cup (W \cap A)| \leq n$. Insgesamt erhalten wir eine Laufzeit in $f(k) \cdot n^{\mathcal{O}(1)}$. □

Eine genauere Analyse liefert:

Lemma 7 (Bodlaender, Drange, Dregi u. a. [30])**.** *Der Algorithmus* BUILDTREEDECOMPOSITION *arbeitet in Zeit* $\mathcal{O}(\, 3^{3k} k^{\mathcal{O}(1)} n(n+m)\,)$.

Beweisidee. Die Laufzeit von FINDBALANCEDPARTITION sei $T(n,k)$. Aus den rekursiven Aufrufen in Schritt 6 & 7 erhalten wir folgende rekursive Formel:

$$T(n,k) \in \mathcal{O}(\, 3^{|W|} k^{\mathcal{O}(1)} (n+m)\,) + T(|S \cup A|, k) + T(|S \cup A|, k)$$

Die Funktionen der Klasse $\mathcal{O}(\, 3^{3k} k^{\mathcal{O}(1)} n(n+m)\,)$ lösen die Formel. □

Korollar 2. BUILDTREEDECOMPOSITION *ist ein 4-Approximationsalgorithmus mit Laufzeit* $\mathcal{O}(\, 3^{3k} n^2\,)$.

Um nun eine Baumdekomposition kleiner Weite zu approximieren, führen wir für ansteigende $k = 1, 2, 3 \ldots$ den Algorithmus BUILDTREEDECOMPOSITION$(G, \varnothing, k)$ aus, bis dieser schließlich eine Baumdekomposition zurückgibt.

Beispiel 7. Sei G der Graph in Abbildung 3.5. Wir wählen $k \leftarrow 1$.

- BUILDTREEDECOMPOSITION$(G, \varnothing, 1)$
  Im ersten Aufruf erhalten wir in Schritt 2, wie dargestellt in Abbildung 3.5, die Partition $(S, A, B) \leftarrow$ FINDBALANCEDPARTITION$(G, \varnothing, 1)$.

  Als neu erzeugten Beutel der Zerlegung erhalten wir $X \leftarrow S \cup W = \{1, 5\}$ in Schritt 11. Wir betrachten nun die Unteraufrufe entlang des linken Zweiges im Rekursionsbaum:

- BUILDTREEDECOMPOSITION$(G[\{1,\dots,6\}],\{1,5\},1)$

  $W = \{1,5\}$. Als Partitionen erhalten wir in Schritt 4:
  $(S,A,B) \leftarrow$ FINDBALANCEDPARTITION$(G[\{1,\dots,6\}],\{1,5\},1)$.
  $S = \{3,6\}$, $A = \{1,2\}$, $B = \{4,5\}$
  Weiter gilt in Schritt 6: $S \cup A = \{1,2,3,6\}$, $S \cup (W \cap A) = \{1,3,6\}$
  Und in Schritt 7: $S \cup B = \{3,4,5,6\}$, $S \cup (W \cap B) = \{3,5,6\}$
  Wir erzeugen den neuen Beutel $X \leftarrow \{1,3,5,6\}$ in Schritt 11.

- BUILDTREEDECOMPOSITION$(G[\{1,2,3,6\}],\{1,3,6\},1)$

  $W = \{1,3,6\}$, $S \leftarrow \{2,6\}$, $A \leftarrow \{3\}$, $B \leftarrow \{1\}$
  $S \cup A = \{2,3,6\}$, $S \cup (W \cap A) = \{2,3,6\}$
  $S \cup B = \{1,2,6\}$, $S \cup (W \cap B) = \{1,2,6\}$
  $X \leftarrow \{1,2,3,6\}$

- BUILDTREEDECOMPOSITION$(G[\{2,3,6\}],\{2,3,6\},1)$
  Gibt in Schritt 1 die Baumzerlegung $(\{X_1 = \{2,3,6\}\},(\{1\}),\varnothing)$ zurück.

Als Ausgabe erhalten wir die in Abbildung 3.8 dargestellte Baumdekomposition $(\mathcal{X},T)$ der Weite 3.

Bemerkung 4. Sei G ein ungerichteter Graph und k eine natürliche Zahl. BUILDTREEDECOMPOSITION$(G,\varnothing,k)$ produziere eine Baumzerlegung $(\mathcal{X},(I,F))$. Im Allgemeinen können wir nicht folgern, dass $\mathbf{tw}(G) \leq k$.

Beweis. Für den Graphen in Abbildung 3.5 gilt:

$$\underbrace{2 \leq tw(G)}_{\text{Lemma 1}} \leq \text{Weite von } (\mathcal{X},(I,F)) = 3 \text{ und } k = 1.$$

□

ABBILDUNG 3.5: BUILDTREEDECOMPOSITION$(G,\varnothing,1)$. Siehe auch Beispiel 7 und Bemerkung 4.

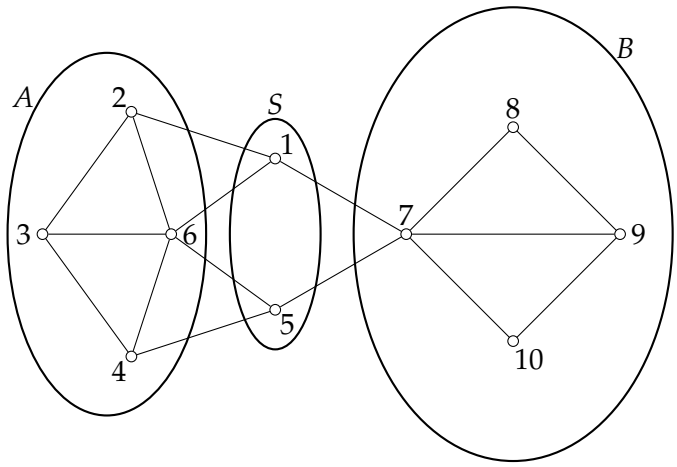


ABBILDUNG 3.6: BUILDTREEDECOMPOSITION($G[\{1,\ldots,6\}], \{1,5\}, 1$)

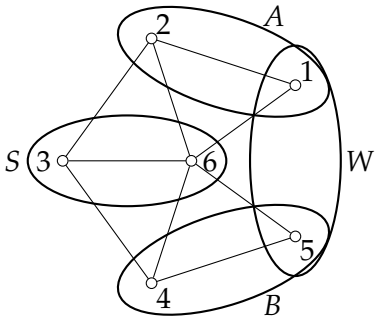
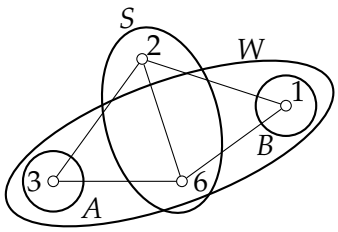


ABBILDUNG 3.7: BUILDTREEDECOMPOSITION($G[\{1,2,3,6\}], \{1,3,6\}, 1$)

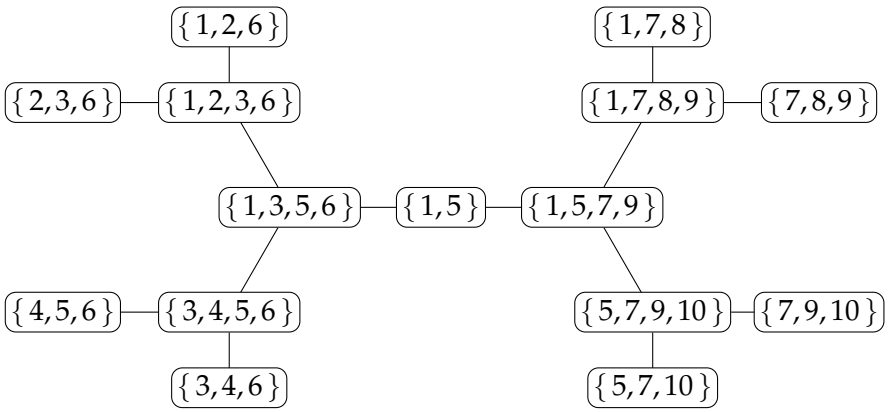


Wir ergänzen einen zweiten Algorithmus zur Approximation der Baumweite mit einer anderen algorithmischen Idee.

Arie Koster befasste sich in seiner Doktorarbeit mit Frequenzzuteilungsproblemen in Funknetzen [35]. Er entwickelte eine andere Heuristik „MSVS“, um die Weite einer gegebenen Baumzerlegung $(\mathcal{X}, (I, F))$ zu reduzieren. Sei G der Graph zu dieser Zerlegung. Der Algorithmus nutzt minimal separierende Knotenmengen (engl. minimal separating vertex sets, MSVS), um einzelne Beutel durch eine Teilzerlegung mit Beuteln geringerer Größe zu ersetzen. Wir benötigen einen Algorithmus um minimal separierende Mengen zu berechnen.

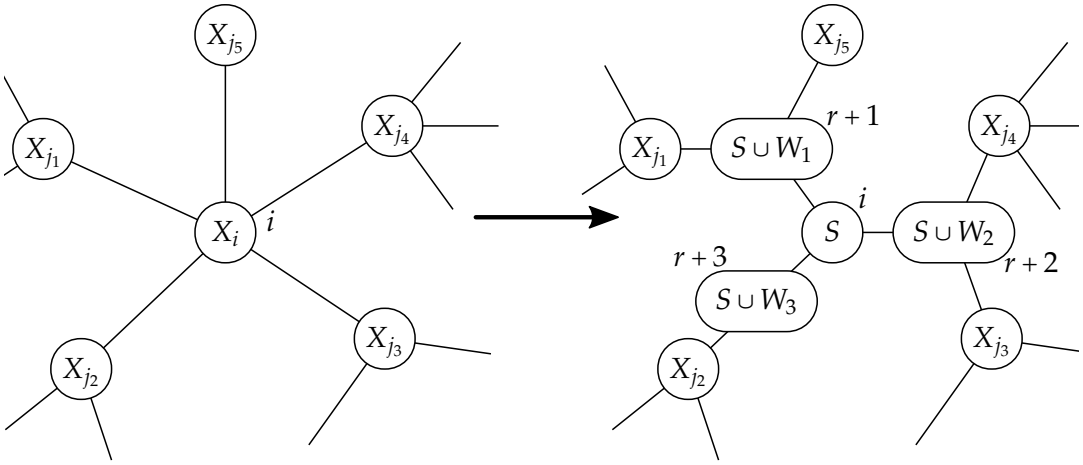
Voraussetzung 2. Der Algorithmus FINDMINIMALSEPERATINGVERTEXSET(G) findet eine minimal separierende Menge der unbekannten Größe k^* in Zeit $\mathcal{O}(\max\{{k^*}^3 \cdot m, k^* \cdot m \cdot n\})$.

ABBILDUNG 3.8: $(\mathcal{X}, T) \leftarrow$ BUILDTREEDECOMPOSITION($G, \varnothing, 1$) auf dem Graphen G aus Abbildung 3.5



Zu einem Beutel X_i der keine Clique in G induziert, konstruiert RefineTreeDecomposition gewissen Hilfsgraphen H_i über den Knoten von X_i. Sei S minimal separierende Knotenmenge in H_i und W_1 bis W_s die durch S getrennten Komponenten. Wir ersetzen X_i durch $S, S \cup W_1, \dots S \cup W_s$ und ermitteln anschließend, wie die zuvor zu X_i benachbarten Beutel mit den neuen Beuteln verbunden werden müssen. Als Vorlage des hier abgedruckten Pseudocode dienen [36, 2.2. Construction of a Tree Decomposition, FIG.3] und [29]. Um im Pseudocode doppelte Indizes zu vermeiden, gehen o.B.d.A davon aus, dass I als ein Intervall natürlicher Zahlen gegeben ist.

Abbildung 3.9: Verfeinerung einer Baumdekomposition durch Algorithmus RefineTreeDecomposition



Algorithmus 2 (RefineTreeDecomposition)**.**

Eingabe: einfacher Graph $G = (V, E)$, Baumdekomposition $(\mathcal{X}, (I = [1, r]_{\mathbb{N}}, F))$

Ausgabe: Baumdekomposition $(\mathcal{X}', (I', F'))$

1: **while** es gibt ein nicht markiertes $i \in I$, sodass $|X_i|$ maximal ist **do**

2: $H_i \leftarrow (X_i, \{ vw \mid v, w \in X_i \wedge (vw \in E \vee \exists j \neq i : v, w \in X_j) \}$.

3: **if** H_i ist vollständig **then**

4: Markiere i und beginne den nächsten Durchlauf der Schleife in Schritt 1.

5: $S \leftarrow$ FindMinimalSeperatingVertexSet(H_i).

6: $W_1, \dots, W_s \leftarrow$ ZHKn von $H_i[X_i \backslash S]$.

7: $I \leftarrow I \cup \{ r+1, \dots, r+s \}$.

8: $\mathcal{X} \leftarrow \{ X'_j = X_j \mid i \neq j \}$
$\cup \{ X'_i = S \}$
$\cup \{ X'_{r+q} = W_q \cup S \mid q \in \{ 1, \dots, s \} \}$

9: $F \leftarrow F \setminus \{ ij \mid j \in N_T(i) \}$
$\cup \{ ik \mid k \in \{ r+1, \dots, r+s \} \}$
$\cup \{ kj \mid j \in N_T(i) \wedge q \in \{ 1, \dots, s \} \wedge (\exists k = r+q) \rightarrow X_i \cap X_j \subseteq W_q \cup S \}$

10: $r \leftarrow r + s$

11: **return** $(\mathcal{X}, (I, F))$

Lemma 8. *Sei $(\mathcal{X}', (I', F'))$ die durch einen Durchlauf der* while*-Schleife konstruierte Verfeinerung einer Baumdekomposition $(\mathcal{X}, (I, F))$ zu $G = (V, E)$ und dem Beutel X_i. Dann ist $(\mathcal{X}', (I', F'))$ eine Baumdekomposition und für die neuen Beutel gilt: $|X'_i|, |X'_{r+1}|, \ldots |X'_{r+s}| < |X_i|$.*

Beweis. Da G ein einfacher Graph ist gilt $|S| \geq 1$. Es folgt $|X'_i| = |S| < |X_i|$ und $|W_q| \geq 1 \Rightarrow |X'_{r+q}| = |S \cup W_q| < |X_i|, \forall q$. Zu zeigen sind die einzelnen Eigenschaften der Baumdekomposition.

- Für jeden Knoten $v \in V$ gilt nach Schritt 8: $v \in X'_j = X_j \in \mathcal{X}$ für $X'_j \in \mathcal{X}'$ mit $i \neq j \leq r$ oder $\exists k{:}\, r+1 \leq k \leq r+s \rightarrow v \in X'_k \in \mathcal{X}'$, denn

$$\bigcup_{r+1 \leq k \leq r+s} X'_k = X_i \in \mathcal{X}.$$

- Sei $vw \in E$. Falls $v, w \in X_j$ für $i \neq j$, so gilt durch Schritt 8 auch $v, w \in X'_j \in \mathcal{X}'$. Seien also $v, w \in X_i$. Wir berechnen in den Schritten 5 & 6 die Partition $X_i = S \cup W_1 \cup \ldots \cup W_s$. Falls $v, w \in S$, dann gilt $v, w \in X'_i \in \mathcal{X}'$. Sei o.B.d.A. $v \in W_k$, so gilt $w \in W_k \cup S$, denn S ist minimal separierende Menge für X_i und enthält demnach alle Nachbarn in $G[X_i]$ von Knoten in W_k. Mit $W_k \cup S = X_{r+k} \in \mathcal{X}'$ folgt die zweite Eigenschaft für beliebige k.

- Der Wald $T \backslash \{ i \}$ ist identisch zu $T' - \{ i, r+1, \ldots, r+s \}$ aufgrund der Schritte 7, 8 & 9. $T'[\{ i, r+1, \ldots, r+s \}]$ ist offensichtlich ein Baum. Wir betrachten einen Nachbarbeutel X_j zu X_i. Nach Schritt 4 gilt: Die Knotenteilmenge $X_j \cap X_i$ ist eine Clique in H_i. Kein Knotenseparator S kann die Knoten einer Clique über mehrere ZHK von $H_i - S$ verteilen. Daher gilt $X_j \cap X_i \subset S \cup W_q$ für genau ein q. Für alle j enthält die Kantenmenge $\{ kj \mid j \in N_T(i) \wedge q \in \{1, \ldots, s\} \wedge (\exists k = r+q) \rightarrow X_i \cup X_j \subseteq W_q \cup S \}$ genau eine solche Kante zwischen j und einem Knoten aus $\{ r+1, \ldots, r+s \}$. T' ist zusammenhängend und kreisfrei. Also ist T' ein Baum.

- Zu zeigen ist $s \geq 1$, sodass mindestens W_1 existiert (siehe auch Bemerkung 5). Andernfalls wäre die Kantenrelation F nach Schritt 9 nicht zusammenhängend. Aufgrund von Schritt 4 ist $H_i = (V_H, E_H)$ nicht vollständig. Es existieren zwei Knoten $v, w \in V_H$ mit $vw \notin E_H$. Es folgt $d(v) \leq |V_H| - 2$ und $d(w) \leq |V_H| - 2$. Also ist $V_H \backslash \{ v, w \}$ ein (v, w)-Separator. Dies impliziert, dass eine minimal separierende Menge existiert.

- Wir zeigen $T'_v = \{ j \in I' \mid v \in X'_j \in \mathcal{X}' \}$ ist ein Teilbaum von T'. Seien X_j die ursprünglichen Nachbarbeutel zu X_i ($ij \in F$): Falls $v \notin X_i$, so ist $T'_v = T_v$ und die Eigenschaft vererbt sich direkt. Falls $v \in S \subset X_i$, so ist v in allen neuen Beuteln von $\mathcal{X}'$ enthalten. Daher sind alle X_j die v enthalten auch in T'_v verbunden. Sei nun $v \in W_q \subset X_i$. Es folgt $v \notin W_p$ für $p \neq q$, da $(S, W_1, \ldots W_s)$ eine Partition von X_i ist. Es gilt $s \geq 1$. Für alle X_j die v enthalten gilt: $X_j \cap X_i \subset S \cup W_q$.

Also sind diese $X_j = X'_j \in \mathcal{X}$ über $X_{r+q} \in \mathcal{X}$ verbunden. Insgesamt folgt: T'_v ist zusammenhängend und damit Teilbaum von T'.

□

Korollar 3. *Der Algorithmus* RefineTreeDecomposition *ist korrekt.*

Beweis. Jeder betrachtete Beutel X_i wird entweder als bearbeitet markiert oder durch Beutel strikt kleinerer Größe ersetzt. RefineTreeDecomposition terminiert, sobald schließlich alle zu Beuteln konstruierte Hilfsgraphen H_i vollständig sind. Die Korrektheit folgt mit vorigem Lemma. □

Wir können den Algorithmus RefineTreeDecomposition verwenden, um das Ergebnis eines anderen Approximationsverfahrens, zum Beispiel BuildTreeDecomposition zu verbessern. Falls noch keine Baumzerlegung zu G bekannt ist und G eine kleine Baumweite hat, können wir eine neue Baumdekomposition durch RefineTreeDecomposition$(G = (V, E), (\{ X_1 = V \}, (\{ 1 \}, \varnothing)))$ berechnen.

Bemerkung 5. Der hier dargestellte Algorithmus weicht von der Vorlage in [29, Algorithm 5] in der Schleifenbedingung ab. Die Autoren verlangen dort „$G[X_i]$ does not induce a clique“, um zu garantieren, dass die Suche nach einer minimal separierenden Menge terminiert. Jedoch genügt dies nicht, falls X_i unerwartet in H_i zu einer Clique vervollständigt wird. Ein entsprechendes Gegenbeispiel ist die Baumzerlegung $(\{ X_1 = \{ 1,2,3 \}, X_2 = 1,3 \}, (\{ 1,2 \}, \{ 12 \}))$ zum Graphen $G = (\{ 1,2,3 \}, \{ 12,23 \})$. Daher findet in Algorithmus 2 die Prüfung auf eine Clique über X_i erst nach Schritt 2 statt.

3.1.3 Repräsentation durch Intervalle, topologische Bäume, kleine, schöne und gewurzelte Baumzerlegungen

Wir lernen äquivalente Modelle zu Pfad- und Baumzerlegung kennen, die uns in Abschnitt 3.1.6 helfen werden, die Beziehung zwischen Graphen beschränkter Pfadweite und Intervallgraphen sowie zwischen Graphen beschränkter Baumweite und chordalen Graphen anzugeben. Jede Pfadzerlegung von G korrespondiert mit einem Intervallmodell.

Definition 14. Sei $\mathcal{X} = (X_i)_{i \in I}$ eine Pfadzerlegung und $\mathcal{I} = \{I_v\}_{v \in V}$ eine Familie von endlichen Intervallen aus $\mathbb{R}$, sodass gilt $I_v \cap I_w \neq \varnothing \Leftrightarrow \exists i \colon v, w \in X_i$. Die Familie $\mathcal{I}$ heißt *Intervallmodell* zur Pfadzerlegung $\mathcal{X}$ und vice versa. Intervallmodell

Abbildung 3.10 zeigt ein Intervallmodell. Die korrespondierende Pfadzerlegung lässt sich konstruieren, indem $\mathbb{R}$ von $-\infty$ nach $+\infty$ durchlaufen wird. Wenn sich die im Durchlauf getroffenen Intervalle ändern, fügen wir einen neuen Beutel am Ende von $\mathcal{X}$ an, der die Indizes der getroffenen Intervalle enthält.

Bemerkung 6. Die Weite der korrespondierenden Pfadzerlegung $\mathcal{X}$ ist die maximale Anzahl minus 1 sich überlappender Intervalle.

Intervallmodelle, die durch „lokale Streckung und Stauchung“ des Zahlenstrahls aus anderen Intervallmodellen hervorgehen führen auf die selbe Pfadzerlegung.

Homöomorphismus

Eine solche Abbildung heißt *Homöomorphismus* $\phi\colon \mathbb{R} \to \mathbb{R}$ und lässt sich über die Eigenschaften charakterisieren, dass ϕ und ϕ^{-1} stetig und bijektiv seien.

Bemerkung 7. Die Korrespondenz zwischen Pfadzerlegungen und Intervallmodellen ist eindeutig bis auf Streckungen und Stauchungen der Intervalle in $\mathcal{I}$, unter einem Homöomorphismus auf $\mathbb{R}$. Insbesondere können wir in Darstellungen des Intervallmodells die Achseneinteilung weglassen.

Beweis. Wir können zu jeder Pfaddekomposition $\mathcal{X}$ ein Intervallmodell finden: Wähle für jedes $v \in V$ das rechts-offene Intervall $I_v = [a, b+1[_{\mathbb{R}}$ mit $a = \min\{\, k \mid v \in X_k \in \mathcal{X} \,\}$ und $b = \max\{\, k \mid v \in X_k \in \mathcal{X} \,\}$. Obiges Verfahren rekonstruiert die Pfaddekomposition.

Seien nun $\mathcal{I}$ und $\mathcal{J}$ zwei Intervallmodelle zur selben Pfadzerlegung $\mathcal{X}$ eines Graphen $G = (V, E)$. Es gilt: $\mathcal{I}$ und $\mathcal{J}$ enthalten die selbe Anzahl durch V indizierter Intervalle. Sei $\phi\colon \mathbb{R} \to \mathbb{R}$ nach Voraussetzung ein Homöomorphismus welcher die Intervalle aus $\mathcal{I}$ auf die Intervalle aus $\mathcal{J}$ abbildet. Da Spiegelungen in 0 homöomorph sind, können wir ϕ insbesondere so wählen, dass gilt: $lim_{r\to+\infty}\ \phi(r) = +\infty$. Die Intervallfamilie $\phi(\mathcal{I}) := \{\, \phi(I) \mid I \in \mathcal{I} \,\}$ ist offensichtlich ebenfalls ein Intervallmodell zur Pfadzerlegung $\mathcal{X}$. Betreten oder Verlassen wir in einem Durchlauf von $-\infty$ nach $+\infty$ das Intervall $I_v \in \mathcal{I}$ wie in obigem Verfahren, so existiert ein w, sodass $\phi(I_v) = J_w \in \mathcal{J}$. Da $\mathcal{I}$ und $\mathcal{J}$ die selbe Pfadzerlegung haben, folgt $v = w$ und somit $\mathcal{J} = \phi(\mathcal{I})$. □

ABBILDUNG 3.10: Beispiel eines Intervallmodells

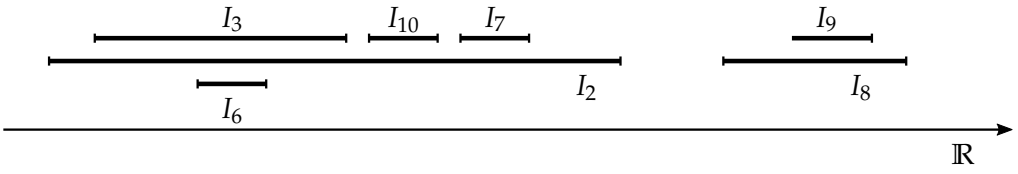
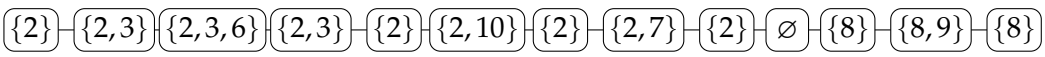


ABBILDUNG 3.11: Pfadzerlegung zum Intervallmodell in Abb. 3.10



Ein ähnliches Modell können wir für Baumzerlegungen angeben. Unter einem topologischen Baum verstehen wir hier (stark spezialisierend) eine Linienzeichnung im $\mathbb{R}^2$ der Kantenrelation zu einem gewöhnlichen Baum, wie sie auch im Zusammenhang mit planaren Graphen definiert wird. Ein topologischer Teilbaum sei eine zusammenhängende Teilmenge eines topologischen Baumes.

Definition 15. Sei $(\{X_i | i \in I\}, T = (I, F))$ eine Baumzerlegung und $\mathcal{T} = \{T_v\}_{v\in V}$ eine Familie von topologischen Teilbäumen eines topologischen Baums $Y \subset \mathbb{R}^2$, sodass gilt $T_v \cap T_w \neq \varnothing \Leftrightarrow \exists i\colon v, w \in X_i$. Die Familie $\mathcal{T}$ ist eine *Repräsentation durch topologische Teilbäume* zur Baumzerlegung $(\{X_i | i \in I\}, T = (I, F))$ und vice versa.

Repr. d. topologische Teilbäume

Wir verändern und erweitern folgendes Beispiel aus der Veröffentlichung [37] von Gavril, 1974:

Beispiel 8. Sei $G_{3.1.3}$ der einfache Graph in Abbildung 3.12 mit einer gültigen Baumdekomposition in Abbildung 3.14. Die oben definierte alternative Repräsentation ist in Abbildung 3.13 dargestellt. Sie zeigt einen Baum Y (rot) mit topologischen Teilbäumen, die nach den Knoten von $G_{3.1.3}$ benannt sind (schwarz). Die Knoten $1,\dots,7$ von $G_{3.1.3}$ entsprechen Teilbäumen $T_1,\dots,T_7$ von Y. Falls zwei Teilbäume einen nicht-leeren Schnitt besitzen, $T_v \cap T_w \neq \emptyset$, so sind v und w im selben Beutel einer Baumzerlegung enthalten.

Die Beutel der Baumzerlegung entsprechen den verschiedenen Kombinationen von Teilbäumen, welche nicht-leere Schnitte ergeben. Die Weite der Baumzerlegung ist die maximale Anzahl sich überlappender Teilbäume von Y.

Offensichtlich hat die Baumdekomposition in Abbildung 3.14 mehr Beutel als notwendig. Wir kontrahieren die Kante zwischen zwei benachbarten Knoten i, j mit Beuteln X_i und X_j, falls gilt $X_i \subseteq X_j$. Wie leicht zu überprüfen ist, führt diese Operation auf eine neue gültige Baumdekomposition gleicher Weite. Wir wiederholen diesen Schritt bis keine derartigen Kantenkontraktionen mehr möglich sind und erhalten so den Graphen in Abbildung 3.15

Der Beutel dieser Zerlegung induziert eine 4-Clique $\{1,4,6,7\}$ in $G_{3.1.3}$. Zusammen mit Lemma 1 erhalten wir, dass die Baumweite von $G_{3.1.3}$ nach unten durch 3 beschränkt ist. Beide hier angegebenen Baumzerlegungen haben Weite 3 und sind daher optimal.

Bemerkung 8. Die Weite der korrespondierenden Baumzerlegung $\mathcal{X}$ ist die maximale Anzahl minus 1 sich überlappender Teilbäume.

Die letzte Konstruktion des vorigen Beispiels erfassen wir in folgender Definition. Sie verkleinert die Kodierung und kann als eine Normalform angesehen werden.

Definition 16.

- Eine Baumzerlegung $(\{X_i | i \in I\}, T = (I, F))$ heißt *kleine Baumzerlegung* , wenn für alle $i, j \in I$ mit $i \neq j$ gilt: $X_i \not\subseteq X_j$. kleine Baumzerlegung
- Eine Baumzerlegung $(\{X_i | i \in I\}, T = (I, F))$ heißt *kleine Pfadzerlegung* , wenn für alle $i, j \in I$ mit $|i - j| = 1$ gilt: $X_i \not\subseteq X_j$. kleine Pfadzerlegung

Wir können die Baumstruktur (I, F) einer Zerlegung verwenden, um eine Halb-ordnung auf den Beuteln einer Baumzerlegung und somit eine implizierte „Abarbeitungsreihenfolge“ auf den Knoten und Kanten des Graphen G festzulegen. Alle Beutel sind kreisfrei verbunden. Sei $i_r \in I$ als Wurzelknoten festgelegt. Wir können leicht überprüfen, dass sich durch F eine eindeutige strenge Halbordnung $(I, \prec_r)$ über I mit $\nexists k : k \prec_r i_r$ impliziert wird. Jeder Kindbeutel X_j hat höchstens einen Elternbeutel X_i mit $i \prec_r j$ als unmittelbaren Vorgänger in $(I, \prec_r)$.

ABBILDUNG 3.12: Einfacher Graph $G_{3.1.3}$ mit 4-Clique $\{1,4,6,7\}$ und 4-Kreis $(3,5,4,6,3)$ ohne Sehne

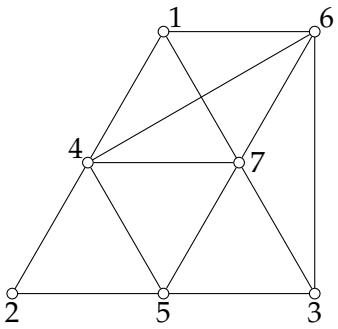


ABBILDUNG 3.13: Baum Y und Teilbäume zum Graphen $G_{3.1.3}$

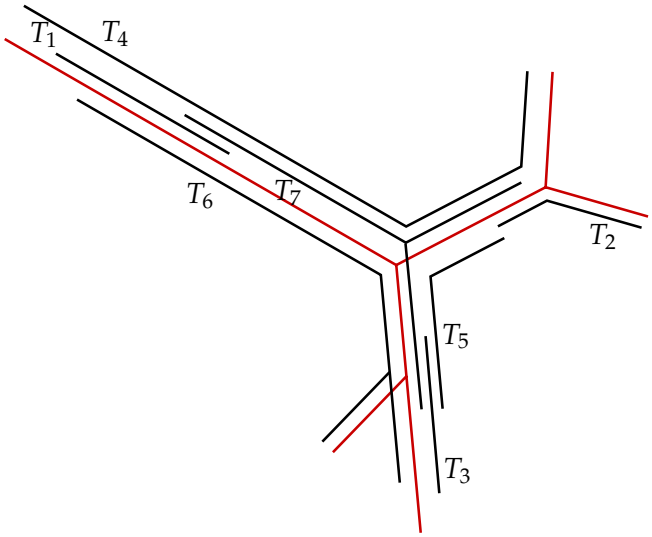


ABBILDUNG 3.14: Zu Baum Y korrespondierende Baumzerlegung von $G_{3.1.3}$

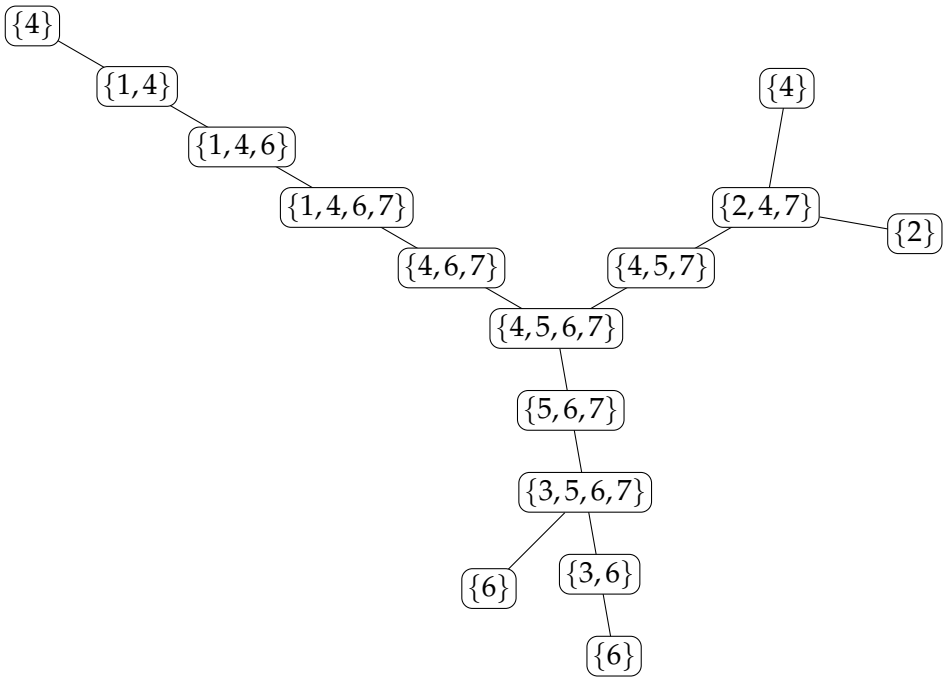
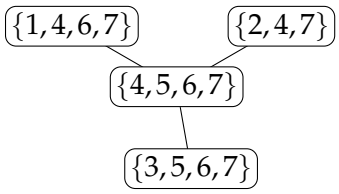


ABBILDUNG 3.15: Kleine Baumzerlegung von $G_{3.1.3}$



Definition 17. Sei $(\mathcal{X}, T = (I, F))$ eine Baumzerlegung. Für einen beliebig gewählten Knoten $r \in I$ ist $(I, \prec_r)$ die implizierte strenge Halbordnung auf T. Wir setzen $R_i = (\{ j \mid \exists j \colon i \prec_r j \}, \{ (i,j) \mid i \prec_r j \wedge \nexists k \colon i \prec_r k \prec_r j \})$ und $R_r = R$. Das Paar $(\mathcal{X}, R)$ heißt *gewurzelte Baumzerlegung* zu G. Das Paar $(\{ X_j \in \mathcal{X} \mid \exists j \colon i \prec_r j \}, R_i)$ heißt gewurzelte Baumzerlegung des induzierten Teilgraphen $G_i = G[\{ X_j \in \mathcal{X} \mid \exists j \colon i \prec_r j \}] \sqsubseteq G$.

gewurzelte Baumzerlegung

In gewöhnlichen, gewurzelten sowie kleinen Baum- und Pfaddekompositionen können sich die Beutel in einer Nachbarschaft X_{j_1}, X_{j_k} von X stark unterscheiden. Gehen wir von Nachbarn $\{X_j\}_{j \in \mathcal{P}([1,k]_\mathbb{N})}$ auf X über, so fallen i.A. je unterschiedliche Knoten aus den X_j weg (Verkleinerung), andere bleiben erhalten, und je unterschiedliche neue Knoten aus X kommen hinzu (Erweiterung). In vielen Algorithmen würde dies zu tief verschachtelten Fallunterscheidungen führen.

Um den Algorithmusentwurf zu erleichtern und die angedeutete Fallunterscheidung zu kapseln, definieren wir schöne Baumzerlegungen, in denen die Beutel X_i nur in vier verschiedenen Arten vorkommen:

Definition 18. Sei G ein Graph mit Baumweite k. Dann existiert eine Baumzerlegung $(\{X_i | i \in I\}, T = (I, F))$, mit einem in r gewurzelten Binärbaum T, es gilt $|I| \in \mathcal{O}(k \cdot n)$ und Beutel kommen nur in folgenden Arten vor:

schöne Baumzerlegung

- Falls ein Knoten $i \in I$ zwei Kindknoten j_1 und j_2 hat, so gilt $X_i = X_{j_1} = X_{j_2}$ (Verzweigungsbeutel).
- Falls ein Knoten $i \in I$ einen Kindknoten j hat, so existiert ein $v \in V$ sodass entweder $v \uplus X_i = X_j$ (Verkleinerungsbeutel) oder $v \uplus X_j = X_i$ (Erweiterungsbeutel).
- Falls ein Knoten $i \in I$ ein Blatt ist, dann gilt $X_i = 1$ (Blattbeutel).

Eine solche Baumzerlegung heißt *schön.*

In theoretischen Betrachtungen kann die Verwendung einer kleinen Baumzerlegung vorzugswert sein, während schöne Baumzerlegungen oft algorithmisch leichter handhabbar sind. Es ist sichergestellt, dass sowohl schöne als auch kleine Baumzerlegungen schnell aus allgemeinen Baumzerlegungen und mit der selben Weite berechnet werden können:

Satz 5.

- *Ein Graph G hat Baumweite k genau dann, wenn zu G eine gewurzelte / schöne / kleine Baumdekomposition der Weite höchstens k existiert.*

- *Zu jeder Baumzerlegung lässt sich in polynomieller Zeit eine gewurzelte / schöne / kleine Baumzerlegung der selben Weite konstruieren.*

Beweis. Zu jeder Wahl einer Wurzel kann die Kantenrelation eines Baumes in linearer Zeit entsprechend eines Wurzelknotens orientiert werden. Für die Aussagen über schöne Baumzerlegungen siehe z.B. [5, Satz 14.13 & Korollar 14.14]. Es verbleiben die Aussagen zu kleinen Baumzerlegungen: Zu jeder Baumzerlegung prüfen wir für alle Kanten mit Teilmengentest, ob sich diese kontrahieren lassen und führen die Kontraktion gegebenenfalls aus. Dies benötigt polynomielle Laufzeit. Die Beutel der Zerlegung können sich nicht vergrößern.

Jede dieser Konstruktionen erhält die größten Beutel unverändert und fügt keine größeren Beutel ein. Zu einer optimalen Baumzerlegung erhalten wir eine gewurzelte / schöne / kleine Baumzerlegung optimaler Weite. □

3.1.4 Charakterisierung von Graphen geringer Baumweite

Durch den Satz von Robertson-Seymour haben wir ein mächtiges Instrument, um Graphfamilien, welche abgeschlossen bzgl. der Bildung von Minoren sind, durch eine endliche Liste verbotener Minoren zu charakterisieren. Wir geben eine solche Charakterisierung für Graphen kleiner Baumweite k an.

Minor **Definition 19.** Ein Graph $H = (V', E')$ heißt *Minor* des Graphen $G = (V, E)$, falls H durch eine beliebige Folge der nachstehenden Operationen aus G gewonnen werden kann.

(1) Löschung einer Kante.

(2) Löschung eines Knotens und aller inzidenten Kanten.

Kontraktion (3) *Kontraktion* einer Kante vw. Das bedeutet wir entfernen vw und fassen v und w zu einem neuen Knoten zusammen, welcher die Nachbarschaft $[N(v) \cup N(w)] \setminus \{u, v\}$ hat [3].

Falls die Knotennamen im Minor erhalten werden sollen, definieren wir für den erhaltene Knotennamen *Minor mit erhaltenen Knotennamen* (engl. labelled minor) die dritte Operation als:

(3) Kontraktion einer Kante $\{v, w\}$, wobei der entstehende Knoten mit v oder w benannt wird.

Falls ein Minor existiert, so auch einer mit erhaltenen Knotennamen. Ebenso wie für Teilgraphen gilt auch für die Minoren eines Graphen G, dass ihre Baumweite die Baumweite $\mathbf{tw}(G)$ des Graphen nicht übersteigt:

Lemma 9 (Bodlaender und Koster [38, Lemma 10])**.** *Sei H ein Minor von G. Dann gilt $\boldsymbol{tw}(H) \leq \boldsymbol{tw}(G)$.*

[3] beachte, dass wir Graphen ohne Schleifen betrachten.

Falls Minoren des Graphen bekannt sind, zum Beispiel aus Eigenschaften der betrachteten Graphfamilie, erhalten wir so eine weitere Möglichkeit, um die Baumweite von G nach unten abzuschätzen.

Sei G ein Teilgraph von H: Dann ist jede Baumzerlegung [Pfadzerlegung] von H, eingeschränkt auf die Knoten von G, auch eine Baumzerlegung [Pfadzerlegung] von G. Folglich gilt auch $\mathbf{tw}(G) \leq \mathbf{tw}(H)$ und $\mathbf{pw}(G) \leq \mathbf{pw}(H)$.

Satz 6 (Bodlaender, Arnborg, Ramachandramurthi & weitere [39]–[41])**.**

- *Ein Graph G hat Baumweite höchstens* 1 *genau dann, wenn er nicht K_3 als Minor enthält.*
- *Ein Graph G hat Baumweite höchstens* 2 *genau dann, wenn er nicht K_4 als Minor enthält.*
- *Ein Graph G hat Baumweite höchstens* 3 *genau dann, wenn er nicht den vollständigen Graphen K_5, den Wagner-Graphen M_8, den Graphen zum regulären Oktaeder, oder den Graphen zum fünfeckigen Prisma als Minoren enthält.*
- *Für Graphen mit höherer Baumweite k steigt die Anzahl der verbotenen Minoren sehr schnell an.*

Für die Pfadweite gelten ähnliche Resultate. Im Allgemeinen ist das Entscheidungsproblem, ob G einen Minor M enthält **NP**-vollständig. Falls hingegen die Liste der verbotenen Minoren konstant gehalten wird, ist ein Test auf genau diese Minoren für G theoretisch in quadratischer Laufzeit möglich [42]. Jedoch ist kein praktisch implementierbarer Algorithmus bekannt, der diese Aufgabe ohne hohe Kontstanten in der Laufzeitfunktion bewältigt.

3.1.5 Algorithmen auf Graphen beschränkter Baumweite

Wir möchten nun exemplarisch untersuchen, wie sich eine gegebene Baumzerlegung einsetzen lässt und verdeutlichen dies am Beispiel WEIGHTEDINDEPENDENTSET auf Graphen beschränkter Baumweite.

Algorithmus 3 (Dynamic-Programming-Pseudocode für Algorithmen auf Graphen beschränkter Baumweite)**.**

Eingabe: Ein ungerichteter Graph G, mit Baumzerlegung $(\mathcal{X}, T)$
Ausgabe: DP-Tabelle C_r für einen beliebigen Knoten r aus T.

1: Sei R_r der in r gewurzelte Baum T und $\prec$ die induzierte Halbordnung.
2: **for each** Beutel X_i in Post-Order-Reihenfolge von R_r **do**
3: **if** i ist Blatt in T **then**
4: Finde Lösungskandidaten zu $G[X_i]$ durch vollständige Exploration.
5: Kodiere diese in der DP-Tabelle C_i.
6: **else**
7: Seien $\{C_j\}_{j \succ i}$ die DP-Tabellen zu Teilgraphen G_j.

8: Kombiniere $\{C_j\}_{j>i}$ und X_i zu Lösungskandidaten über G_i.
9: Kodiere die Lösungskandidaten in der DP-Tabelle C_i.
10: Gib den Speicher für die DP-Tabellen $\{C_j\}_{j>i}$ frei.

Wir lösen das Graphenproblem *bottom-up*, indem aus Lösungskandidaten zu G_j mit $i <_r j$ und dem Beutel X_i Lösungskandidaten auf G_j konstruiert werden. Schritt 8 ist das Herzstück des Algorithmus 3. Für klassisch **NP**-schwere Graphenprobleme ist hier in der Regel eine vollständige Exploration von Kombinationen aus Lösungskandidaten zu G_j und Teilmengen von X_i notwendig. Um die Laufzeit zu bändigen, streben wir beim Algorithmusentwurf folgende Eigenschaften an: (1) Die Anzahl von Kombinationen über X_i lässt sich durch eine Funktion beschränken, die *nur von k* abhängt. (2) Um eine polynomielle Laufzeit in der Größe von T zu erzielen, tabellieren wir Lösungskandidaten auf G_i. Dieses weit verbreitete Meta-Verfahren heißt *Dynamic Programming*. Wir versuchen die Größe dieser DP-Tabellen C_i[4] möglichst klein zu halten.

Beispiel 9. Wir sind nun in der Lage Algorithmus 3 zu einem Algorithmus für MAX-WEIGHTEDINDEPENDENTSET zu vervollständigen. Als Vorlage dient ein Beispiel aus [5, S.365-369]. Das zugehörige parametrisierte Entscheidungsproblem lautet:

> WEIGHTEDINDEPENDENTSET mit Parameter Baumweite
> Eingabe: Ein ungerichteter Graph $G = (V, E)$, eine Gewichtsfunktion $w\colon V \to \mathbb{R}$, eine Zahl $s \in \mathbb{R}$.
> Parameter: Eine natürliche Zahl k.
> Frage: Hat G beschränkter Baumweite k eine unabhängige Knotenteilmenge M, sodass $\Sigma_{v\in M} w(v) \geq s$?

Wir berechnen zunächst einen exakte Baumzerlegung der Weite k oder eine Approximation. Anschließend wandeln wir diese in eine schöne gewurzelte Baumzerlegungung $(\mathcal{X}, R = (I, F))$ um. Sei $V_{i\leq}$ die Menge aller Knoten die im Beutel zu i oder Nachfahren von i vorkommen. Der Graph $G_{i\leq} = G[V_{i\leq}]$ bezeichne den Zugehörigen induzierten Teilgraphen.

Wir verwenden nun Algorithmus 3. In der DP-Tabelle C_i kodieren wir für jede Teilmenge $S \subseteq X_i$ des Beutels zu i, das maximale Gewicht einer unabhängigen Teilmenge von $G_{i\leq}$, welche die Knoten S und keine weiteren aus X_i enthält. Formal gelte:

$$C_i(S) = \begin{cases} -\infty, \text{ falls keine unabhängige Teilmenge in } G_{i\leq} \text{ existiert,} \\ \max\{ \sum_{v\in M} w(v) \mid M \text{ ist unabhängig in } G_{i\leq} \text{ und } M \cup X_i = S \subseteq X_i \}, \text{ sonst.} \end{cases}$$

Da k konstant ist erhalten wir auch konstant viele Tabelleneinträge. Für Blattbeutel X_i der schönen Baumzerlegung $(\mathcal{X}, R = (I, F))$ erreichen wir Schritt 4. Es gilt $G_{i\leq} =$

[4] C steht für *dynamic programming* ***c****ache*

$(X_i = \{ v \}, \varnothing)$ für einen Knoten $v \in V$. die maximal unabhängigen Knotenteilmengen dieses Graphen sind $\varnothing$ und $\{ v \}$ wir kodieren die Tabelleneinträge $C_{i\leq}(\{ v \}) = w(v)$ und $C_{i\leq}(\varnothing) = 0$ in Schritt 5.

In Schritt 8 führen wir je nach Art des Beutels X_i eine der folgenden Operationen aus:

- X_i ist ein Erweiterungsbeutel: Für den Beutel X_j zum Kind j von i gilt: $X_i = X_j\{ v \}$, für einen gewissen Knoten $v \in V$, der X_j erweitert. Wir bestimmen nun die unabhängigen Mengen in $G_{i\leq}$ mit Hilfe der Tabelle $C_{j\leq}$ und kodieren die zugehörigen Gewichte.

$$C_{i\leq}(S) = \begin{cases} -\infty & \text{, falls } \exists w \in S : v \in S \wedge vw \in E \\ C_{j\leq}(S) + w(v) & \text{, falls } \forall w \in S : v \in S \wedge vw \notin E \\ C_{j\leq}(S) & \text{, falls } v \notin S. \end{cases}$$

- X_i ist ein Verkleinerungsbeutel: Für den Beutel X_j zum Kind j von i gilt: $X_i = X_j\backslash\{ v \}$, für einen gewissen Knoten $v \in V$, der in X_i nicht mehr vorkommt. Um die neue DP-Tabelle $C_{i\leq}$ zu bestimmen, müssen wir lediglich die maximalen Gewichte zu unabhängigen Mengen, die entweder v enthalten oder nicht, in der Tabelle $C_{j\leq}$ nachschlagen.

$$C_{i\leq}(S) = \max\{ C_{j\leq}(S), C_{j\leq}(S) \}$$

- X_i ist ein Verzweigungsknoten: i hat genau zwei Kinder j und k und es gilt $X_i = X_j = X_k$ Die DP-Tabellen $C_{j\leq}$ und $C_{k\leq}$ enthalten genau die maximalen Gewichte zu allen unabhängigen Teilmengen $S = M \cap V_{j\leq}$ in $G_{j\leq}$ und $S = N \cap V_{k\leq}$ in $G_{k\leq}$. Für alle S werden in der Summe $C_{j\leq}(S) + C_{k\leq}(S)$ genau die Gewichte von Knoten in S doppelt gezählt. Wir erhalten das neue maximale Gewicht einer unabhängigen Menge zu $G_{i\leq}$ welche die Knoten S aber keine weiteren Knoten aus X_i enthält, indem wir die Summe um das zu viel gezählte Gewicht reduzieren.

$$C_{i\leq}(S) = C_{j\leq}(S) + C_{k\leq}(S) - \sum_{v\in S} w(v)$$

Wir berechnen schließlich das maximale Gewicht einer unabhängigen Menge M in G aus der DP-Tabelle zum Wurzelbeutel X_r zu

$$\sum_{v\in M} w(v) = \max\{ C_{r\leq}(S) \mid S \subseteq X_i \}$$

Satz 7 (Krumke und Noltemeier [5])**.** *Der Algorithmus in Beispiel 9 ist korrekt und arbeitet in Zeit* $\mathcal{O}((2^{\mathcal{O}(k^3)} + k2^{k+1})n)$.

Korollar 4. WEIGHTEDINDEPENDENTSET *mit Parameter Baumweite ist in* **FPT**.

Falls wir eine Baumzerlegung zu einem Graphen $G = (V, E)$ kennen, dessen Weite durch eine feste Konstante k beschränkt ist, so werden viele weitere Probleme, die für allgemeine Graphen **NP**-vollständig sind, in polynomieller oder sogar linearer Zeit lösbar [25].

In einem DP-Algorithmus auf einer Baumzerlegung $D = ((X_i), (I, F))$ zu einem Graphen G wird der Umstand ausgenutzt, dass die meisten Beutel $i \in I$ selbst Separatoren in einem Graphen $G[M]$ sind. M bezeichnet dabei die vereinigte Knotenmenge aller Beutel unter i. Sei dazu $i_r \in I$ der Wurzelknoten von D. Die Knoten im Baum (I, F) sind dadurch über eine Eltern-Kind-Relation geordnet. Der DP Algorithmus arbeitet die Knoten in einem Post-Order-Durchlauf der Baumzerlegung, das heißt von den Blättern zur Wurzel, ab. Für jeden Knoten der Baumzerlegung wird eine Tabelle angelegt, welche die Informationen über Lösungen des betrachteten Problems auf dem Knoten und all seinen Kindern enthält.

Die dritte Eigenschaft der Definition von Baumzerlegungen verlangt, dass zu einem Knoten v, die Beutel, welche v enthalten, ein Teilbaum der Baumzerlegung bilden. Diese Eigenschaft kommt uns bei DP-Algorithmen auf Baumzerlegungen zu gute. Denn wenn der Knoten v in einem Teilbaum unter i vorkommt, jedoch in X_i nicht mehr enthalten ist, so ist v auch im Rest der Zerlegung oberhalb von i und in anderen Zweigen der Zerlegung nicht enthalten. Alle Informationen zu v sind „lokal“ zum Teilbaum unter i, das bedeutet in DP-Tabellen oberhalb von i muss v nicht mehr geführt werden.

Da ein Knoten $v \in V$ nur in untereinander verbundene Beutel einer Baumzerlegung fallen kann, ergeben sich oft Möglichkeiten, das Problem zu verkleinern. Beutel sind eine sinnvolle Einheit für die Abarbeitung in einem Algorithmus.

Die DP-Tabellen der Kinder von i zusammenzuführen, um eine Problemlösung für den gesamten Teilbaum unter i zu erhalten, ist im Allgemeinen sehr schwierig [43]. Oft muss der Lösungsraum kombinatorisch exploriert werden, wodurch sich ein exponentieller Faktor für die Laufzeit ergibt. Da jedoch die Anzahl der Knoten in X_i durch die Baumweite k beschränkt ist, können Algorithmen gefunden werden, in denen der exponentielle Anteil der Laufzeit nur von k abhängt.

In DP-Algorithmen werden schöne Baumzerlegungen verwendet. Dies hat den Vorteil, dass nur vier Fälle für die Behälter unterschieden werden müssen. Für Blätter und Verkleinerungsknoten im Zerlegungsbaum ist die Berechnung der DP-Tabelle oft sehr einfach. Die Algorithmen für Erweiterungsknoten und Verzweigungsknoten sind vergleichsweise umfangreich. Abschließend wird die Lösung aus der DP-Tabelle des Wurzelknotens abgeleitet.

Oft kann in der Praxis auf die Suche nach optimalen Baumzerlegungen verzichtet werden, da auch Baumzerlegungen mit kleinem $k \geq \mathbf{tw}(G)$ ausreichend sind, um das vorliegende Problem mit vertretbarer Laufzeit zu lösen.

Für einige Probleme existieren Algorithmen mit polynomieller Laufzeit, obwohl die gegebene Baumzerlegung eine Weite beschränkt in lediglich $\mathcal{O}(\log(n))$ hat. [25].

Einige weitere Beispiele für parametrisierte Entscheidungsprobleme die Linearzeit **FPT**-Algorithmen erlauben sind: 3-COLOURABILITY [25], WEIGHTEDINDEPENDENTSET [44], MAXIMUMCUT [45], HAMILTONIANCYCLE [25] und DOMINATINGSET [44].

3.1.6 Baumzerlegungen und chordale Graphen

Durch folgenden Satz von Gavril über Teilbaumschnittgraphen und Intervallschnittgraphen lernen wir, wie eng Baumzerlegungen mit chordalen Graphen in Beziehung stehen. Unser Ziel ist es, die reichhaltige Theorie zu chordalen Graphen zur Lösung von Baumweiteproblemen heranzuziehen.

Definition 20.

- Zu einer Familie von topologischen Teilbäumen $\{T_v\}_{v\in V}$ eines topologischen Baums $Y \subset \mathbb{R}^2$ sei $S = \{ uv \mid T_v \cap T_w \neq \varnothing \text{ und } u \neq v \}$. Der ungerichtete Graph $H = (V, S)$ heißt *Teilbaumschnittgraph* . Teilbaumschnittgraph
- Zu einem Intervallmodell $\mathcal{I} = \{I_v\}_{v\in V}$ sei $S = \{ uv \mid I_v \cap I_w \neq \varnothing \text{ und } u \neq v \}$. Der ungerichtete Graph $H = (V, S)$ heißt *Intervallschnittgraph* oder verkürzt *Intervallgraph* . Intervallgraph

Satz 8 (Gavril, 1974 [37, Theorem 3])**.** *Ein Graph G ist genau dann ein Teilbaumschnittgraph, wenn G ein chordaler Graph ist.*

Korollar 5. *Intervallgraphen sind chordal.*

Wir zeigen nun, dass die Modelle „Baumzerlegung“ und „Teilbaumschnittgraph“ gegeneinander austauschbar sind:

Lemma 10. *Sei $(\{X_i | i \in I\}, T = (I, F))$ eine Baumzerlegung zu G, dann existiert ein Teilbaumschnittgraph H, welcher chordaler Obergraph von G ist und die selbe Baumzerlegung hat.*

Beweis. Sei die Familie $\mathcal{T}$ eine Repräsentation durch topologische Teilbäume von $(\{X_i | i \in I\}, T = (I, F))$. Sei $H = (V, S)$ der Teilbaumschnittgraph zu $\mathcal{T}$, dann gilt:

$$\begin{aligned} S &\overset{\text{Def. } 20}{=} \{ uv \mid T_v \cap T_w \neq \varnothing, u \neq v \text{ und } T_v, T_w \in \mathcal{T} \} \\ &\overset{\text{Def. } 15}{=} \{ uv \mid \exists i \colon v, w \in X_i \text{ und } u \neq v \} \\ &\overset{\text{Def. } 9}{\supseteq} E \end{aligned}$$

Es folgt $G \sqsubseteq H$. H ist eine Chordalisierung von G und hat die Baumzerlegung $(\{X_i | i \in I\}, T)$. Wir bezeichnen H als die durch $(\{X_i | i \in I\}, T)$ *implizierte Chordalisierung*. □

Definition 21. Wir erhalten die implizierte Chordalisierung H aus G, indem wir alle Knoten je Beutel $\{X_i | i \in I\}$ zu Cliquen in G vervollständigen. implizierte Chordalisierung

Chordale Graphen mit der Struktur einer implizierten Chordalisierung heißen im Fachgebiet des maschinellen Lernens engl. „junction tree“. Um das Konzept der Baumweite in die Theorie der chordalen Graphen zu übertragen, müssen wir untersuchen, wie optimale Baumzerlegungen von G durch eine Aussage über chordale Obergraphen von G charakterisiert werden können.

Lemma 11. *Sei $(\{X_i | i \in I\}, T = (I, F))$ eine optimale Baumzerlegung zu G und die implizierte Chordalisierung $H \supseteq G$. Es gilt $\omega(H) - 1 = tw(G)$.*

Beweis. Die gemeinsame Baumzerlegung existiert nach Lemma 10. Wenn $(\{X_i | i \in I\}, T)$ optimal für G ist, folgt mit $\mathbf{tw}(G) \leq tw(H) =$ Weite von $(\mathcal{X}, T) - 1 = tw(G)$ die Optimalität für H. Für jeden Beutel $X_i \in \mathcal{X}$ gilt: X_i ist eine Clique in H.

Behauptung: $\{X_i | i \in I\}$ enthält die maximalen Cliquen von H. Angenommen es gäbe eine Clique C in H, die nicht Teilmenge eines Beutels X_i ist, dann gilt für alle i: $|C| > |X_i|$ im Widerspruch zu Lemma 1 und der Optimalität der Baumzerlegung.

Insgesamt gilt:

$$\begin{aligned} tw(G) &= tw(H) \\ &= \max \text{Weite von } (\{X_i | i \in I\}, T) - 1 \\ &= \max\{\, |X_i| \mid X_i \in X \,\} - 1 \\ &= \max\{\, |C| \mid C \text{ ist Clique in } H \,\} - 1 \\ &= \omega(H) - 1 \end{aligned}$$

□

Also sind unter allen Chordalisierungen die chordalen Obergraphen zu optimalen Baumzerlegungen enthalten.

Gavril gibt bereits 1974 einen Algorithmus an, um zu jedem chordalen Graphen eine Repräsentation durch Teilbäume $\{T_v\}_{v \in V}$ eines passenden Baumes Y zu konstruieren. Wir können diesen Algorithmus verwenden um Baumzerlegungen zu Chordalisierungen von G anzugeben. Diese sind auch Baumzerlegungen von G. Insbesondere können wir so alle optimalen Baumzerlegungen finden. Die Konstruktion ist auch ohne Umweg über Schnittgraphen in linearer Zeit möglich, siehe z.B. [15], [46]. Wir erhalten folgende Korollare.

Korollar 6 (z.B. [15], [46])**.** *Sei $G = (V, E)$ ein ungerichteter Graph. Es gilt:*

(i) Die Baumweite $\mathrm{tw}(G)$ *ist das Minimum von $\omega(H) - 1$, über alle Chordalisierungen H von G.*

(ii) Sei H' eine Chordalisierung des Graphen G dann kann eine Baumzerlegung von G mit Weite $\omega(H') - 1$ in linearer Zeit berechnet werden.

(iii) Sei $(\{X_i | i \in I\}, T = (I, F))$ eine Baumzerlegung von G der Weite genau k. Für den chordalen Graphen $H'' = (V, E'')$ mit $E'' := \{\, \{v, w\} \mid v \neq w, \exists i \in I : v, w \in X_i \,\}$ folgt $\omega(H'') = k + 1$

Beweis. Zu zeigen bleibt die lineare Laufzeit für (ii). Siehe hierzu [15] □

Korollar 7. *Seien $G = (V, E)$ ein ungerichteter Graph und π Eliminationsschemata:*

$$tw(G) = \min_{\pi=(v_1 \ldots v_n)} \left[\max_i |G[v_i, \ldots, v_n] \cap N(v_i)| \right]$$

Beweis. Wir erhalten diese Aussage aus Korrolar 6 (i), indem wir die Charakterisierung in Satz 2 verwenden, um die maximalen Cliquen aufzulisten (siehe auch Bemerkung 1). □

Wir können Korollar 6 auch benutzen, um unser Wissen um Baumzerlegungen mit vollständig k-partiten Teilgraphen zu erweitern. Wir notieren $\tilde{K}_{n_1,\ldots,n_r}$ für einen Graphen, der isomorph zu $K_{n_1,\ldots,n_r}$ ist.

Lemma 12. *Sei G ein unterichteter Graph. Zu jedem vollständig bipartiten induzierten Teilgraphen $\tilde{K}_{n_1,n_2}$ mit Knotenpartition (A, B), und für alle $a \in A, b \in B$, existieren in jeder Baumdekomposition $(\{ X_i \mid i \in I \}, (I, F))$ Beutel X_i, sodass $A \cup \{ b \} \subseteq X_i$ oder $B \cup \{ a \} \subseteq X_i$. Es gilt $\boldsymbol{tw}(G) \geq \min\{ |A|, |B| \}$. Falls $G = \tilde{K}_{n_1,n_2}$, dann gilt $\boldsymbol{tw}(G) = \min\{ |A|, |B| \}$.*

Beweis. Beobachte, dass in jeder Chordalisierung H von G mindestens eine der Mengen A oder B eine Clique in H induziert. Angenommen dies wäre nicht der Fall, dann existieren mindestens 2 Knoten in je A und B. Seien diese Knoten $a_1, a_2 \in A$ und $b_1, b_2 \in B$. Es folgt $(a_1, b_1, a_2, b_2, a_1)$ ist ein Kreis der Länge 4 in H ohne Sehne. Widerspruch. Also induziert o.B.d.A. A die gesuchte Clique in H. Weil $b \in B$ mit allen Knoten aus A verbunden ist, erhalten wir auch $A \cup \{ b \}$ ist Clique in H für alle b. Es gilt $|A \cup \{ b \}| = |A| + 1 \leq \omega(H) \leq \mathbf{tw}(G) + 1$, durch Lemma 1 und Korrolar 6. Außerdem ist $(\{ X_i = A \cup \{ b_i \} \mid b_i \in B \}, P_{n_2})$ eine korrekte Baumzerlegung[5], falls $G = \tilde{K}_{n_1,n_2}$, andernfalls sind diese Beutel in Teilmengen einer Baumzerlegung für $G \supset \tilde{K}_{n_1,n_2}$ enthalten. Analog für B. Da **tw** insgesamt minimal gewählt wird, gilt $\min\{ |A|, |B| \} \leq \mathbf{tw}(G)$ und Gleichheit, falls $\min\{ |A|, |B| \} + 1 = \omega(G)$. □

Lemma 13. *Sei $G = V, E$ unterichteter Graph. Für jeden vollständig k-partiten induzierten Teilgraphen $\tilde{K}_{n_1,n_2,\ldots n_r} \sqsubseteq G$ mit $\eta = \sum_i n_i$ und Knotenpartition $(F_1, F_2, \ldots, F_r)$ gilt:*

(i) Sei $S = F_{i_1} \uplus \ldots \uplus F_{i_s}$ mit $i_1, \ldots, i_s \in [1, r]_{\mathbb{N}} \wedge s < r$ und sei $T = V \backslash S$. Definiere: $s = |F_i|$, $t = |V \backslash S|$ und $\tilde{K}_{s,t}$ sei vollständig bipartit mit Partition (S, T). Dann gilt: $\tilde{K}_{s,t} \sqsubseteq \tilde{K}_{n_1,n_2,\ldots n_r} \sqsubseteq G$. Für alle $a \in S, b \in T$, existieren in jeder Baumdekomposition $(\{ X_j \mid j \in I \}, (I, F))$ Beutel X_j, sodass $S \cup \{ b \} \subseteq X_j$ oder $T \cup \{ a \} \subseteq X_j$.

(ii) Die Baumweite von G beträgt mindestens:

$$\boldsymbol{tw}(G) \geq \eta - \max_{\substack{\tilde{K}_{n_1,n_2,\ldots,n_r} \text{ mit Partitions-} \\ \text{klassen } (F_j) \text{ und } i \in [1,r]_{\mathbb{N}}}} |F_i|$$

[5] genauer auch Pfadzerlegung

(iii) *Falls* $\tilde{K}_{n_1,n_2,\ldots n_r} = G$ *dann gilt:*

$$\boldsymbol{tw}(G) = n - \max_{\substack{\tilde{K}_{n_1,n_2,\ldots,n_r} \text{mit Partitions-} \\ \text{klassen } (F_j) \text{ und } i \in [1,r]_{\mathbb{N}}}} |F_i|$$

Beweis. (i) Alle Knoten in einer Partionsklasse F von $\tilde{K}_{n_1,n_2,\ldots n_r}$ sind je zu allen Knoten in von F verschiedenen Partitionsklassen verbunden. Um $\tilde{K}_{s,t}$ zu konstruieren, verwerfen wir also lediglich alle Kanten in S und $V \backslash S$, erhalten aber alle Kanten zwischen diesen Mengen. Offensichtlich ist $(S, V \backslash S)$ eine Knotenpartition von V zu $\tilde{K}_{s,t}$. Mit vorigem Lemma 12 gilt die Aussage.

(ii) Wir zeigen mit vollständiger Induktion über r:

IA. $r = 2$ impliziert $\eta = n_1 + n_2$. Es folgt:

$$\mathbf{tw}(G) \geq \min\{\, |F_1|, |F_2| \,\} = \eta - \max\{\, |F_1|, |F_2| \,\}$$

In jeder Chordalisierung H von G existiert eine Clique C über F_1 oder F_2 und einem weiteren Knoten w_2 bzw. w_1 aus der je anderen Partitionsklasse der Größe mindestens $\eta - \min\{\, n_1, n_2 \,\} + 1$ (Lemma 12).

IS. $r - 1 \to r$: Nach Induktion haben wir eine Partitionsklasse F_j zu $\tilde{K}_{n_1,n_2,\ldots n_{r-1}}$ die nicht notwendigerweise eine Clique ist und eine Clique C der Größe mindestens $\left(\sum_{i=1}^{r-1} n_i\right) - n_j + 1$. Verbinde nun $\tilde{K}_{n_1,n_2,\ldots n_{r-1}}$ vollständig bipartit mit der Menge F_r. Wir erhalten $\tilde{K}_{n_1,n_2,\ldots n_r}$. Nach Definition enthält $G[F_j \cup F_r]$ eine vollständig bipartite Teilmenge mit Partition (F_j, F_r). Nach Lemma 12 ist F_j oder F_r Clique in einer Chordalisierung H. Wähle einen weiteren Knoten w_j bzw. w_r aus der Menge, die nicht notwendigerweise Clique ist. Somit ist auch $C \cup F_j \cup \{\, w_r \,\}$ oder $C \cup F_r \cup \{\, w_j \,\}$ eine Clique der Größe mindestens $\left(\sum_{i=1}^{r} n_i\right) - n_j + 1$.

Insgesamt gilt für eine optimale Chordalisierung H^*:

$$\begin{aligned}
\mathbf{tw}(G) \;= \omega(H^*) - 1 \quad &\geq \left[\left(\sum_{i=1}^{r} n_i\right) - n_j + 1\right] - 1, \forall \tilde{K}_{n_1,n_2,\ldots,n_r} \exists j : 1 \leq j \leq r \\
\Rightarrow \omega(H^*) - 1 \quad &\geq \max_{\substack{\tilde{K}_{n_1,n_2,\ldots,n_r} \text{mit Partitions-} \\ \text{klassen}(F_i) \text{ und } j \in [1,r]_{\mathbb{N}}}} \left[\eta - n_j + 1\right] - 1 \\
&= \eta - \max_{\substack{\tilde{K}_{n_1,n_2,\ldots,n_r} \text{mit Partitions-} \\ \text{klassen}(F_i) \text{ und } j \in [1,r]_{\mathbb{N}}}} |F_j|
\end{aligned}$$

(iii) Es gilt $\tilde{K}_{n_1,n_2,\ldots n_r} = G$: Die Baumzerlegung zum optimalen F_j^* in (ii) konstruiert wie in Lemma 12 hat die Weite $\eta - |F_j^*|$. Mit $\eta = n$ gilt auch $\mathbf{tw}(G) \leq n - |F_j^*|$. Zusammen mit (ii) folgt Gleichheit.

□

Wir nennen Lemma 12 (i) separat, da diese Aussage auch die Lage von Cliquen in Approximationen von Baumzerlegungen beschreibt. Während die kleinstmöglich gewählte Clique in (ii) & (iii) möglicherweise in Approximationen nicht induziert wird.

Wir fahren mit weitern Konsequenzen aus Korrolar 6 fort. Die Baumweite eines Graphen G zu berechnen, ist äquivalent dazu, eine Chordalisierung von G mit minimaler Cliquezahl ω zu bestimmen. Wir können für das Problem TREEWIDTH äquivalent formulieren:

> TREEWIDTH
> Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k.
> Frage: Hat G eine Chordalisierung mit Cliquenzahl höchstens $k + 1$?

Wir erinnern aus Abschnitt 2.1.2, dass die Cliquenzahl eines chordalen Graphen in linearer Zeit bestimmt werden kann. Diese Erkenntnis wirft die Frage auf, welche Kanten wir zu G hinzufügen können, um G zu chordalisieren, ohne dass sich die Baumweite erhöht. So könnten wir das Baumweiteproblem exakt lösen.

Ein mögliches Verfahren, um Graphen zu Chordalisieren, erhalten wir direkt aus Satz 2, der chordale Graphen über perfekte Eliminationsschemata charakterisiert, indem wir die Forderung, dass ein zu eliminierender Knoten simplizial sei, fallen lassen.

Definition 22. Sei G ein ungerichteter Graph mit $|V| = n$.

- Ein *Eliminationsschema* für G ist eine beliebige bijektive Abbildung $\pi: V \to \{1, \dots n\}$. — Eliminationsschema

- Folgendes Verfahren ELIMINATIONGAME produziert aus Ausgabe den sogenannten *Eliminationsgraphen* G^+_π zu G und π. — Eliminationsgraph

  Eingabe: Graph $G = (V, E)$, Eliminationsschema $\pi: V \to \{1, \dots n\}$
  Ausgabe: Eliminationsgraph G^+_π — ELIMINATIONGAME

  1: $V' \leftarrow \varnothing, E' \leftarrow \varnothing$
  2: **for** $i \leftarrow 1, \dots, n$ **do**
  3: $\quad v \leftarrow \pi^{-1}(i)$
  4: $\quad V' \leftarrow V' \cup \{ v \}$
  5: $\quad C \leftarrow \{ uw \mid \exists u, w: u, w \in N_G(v) \}$
  6: $\quad E' \leftarrow E' \cup \{ vw \mid \exists w: w \in N_G(v) \} \cup C$
  7: $\quad G \leftarrow G - \{ v \}$
  8: **return** (V', E')

- Betrachte einen einzelnen Durchlauf der **for**-Schleife. Für jeden ausgewählten Knoten v wird die Nachbarschaft der noch nicht zuvor eliminierten Knoten zu einer Clique C in G^+_π vervollständigt. Die gegenüber $G - V'$ neu hinzugefügten

Füllkanten

Kanten heißen *Füllkanten* (engl. fill-in edges) zu π, v. Wir bezeichnen die Anzahl der Füllkanten mit $fill^+_G(v)$

Lemma 14. *Für jeden Graphen G und jedes Eliminationsschema π gilt: Der Eliminationsgraph G^+_π ist chordal, Obergraph von G und hat π als perfektes Eliminationsschema.*

Beweis. V' enthält stets die bisher eliminierten Konten von G. Da π eine Bijektion ist, gilt schließlich $V = V'$. Falls in Schritt 6 eine Nachbarschaft gegenüber der Relation E unvollständig ist, so wurde diese Kante bereits in einem vorigen Schritt 6 hinzugefügt. Es gilt $E \subseteq E'$ und somit $G \sqsubseteq G^+_\pi$. Jeder Knoten v ist nach Schritt 5 & 6 simplizial in $G^+_\pi - (V' \backslash \{v\})$. Da in folgenden Durchläufen keine weiteren zu v inzidenten Kanten erzeugt, aber alle in C betrachteten Knoten noch nachfolgend eliminiert werden, ist v auch simplizial in $G^+[\{\pi^{-1}(i), \ldots, \pi^{-1}(n)\}]$. Somit ist π perfekt und G^+_π nach Satz 2 chordal. □

Bemerkung 9. Um den Eliminationsgraphen „in-place“ auf G zu berechnen, kann das Eliminationsschema rückwärts durchlaufen werden: Sei v der aktuelle Knoten. Vervollständige die bereits zuvor gesehenden Nachbarn von v zu einer Clique [47]. Hier wurde nur aus definitorischen Gründen die andere Richtung gewählt.

Verschiedene Eliminationsschemata für das EliminationGame, welche die Cliquenzahl von G^+_π gering halten, liefern uns Approximationen der Baumweite. Die zugehörige Baumdekomposition lässt sich in linearer Zeit konstruieren (Korrolar 6). Um ein geeignetes Eliminationsschema zu raten, können wir verschiedene Heuristiken verwenden.

Beispiel 10. Folgende Heuristiken können in polynomieller Zeit berechnet werden [48].

- Wir halten die Baumweite gering, indem möglichst wenige Kanten zu G hinzugefügt werden. Denn die Wahrscheinlichkeit die Cliquenzahl zu erhöhen, ist umso geringer, je weniger neue Kanten hinzugefügt werden.

  Konstruiere das Eliminationsschema π wie folgt: Wähle einen Knoten v, der die minimale Anzahl an Füllkanten in G erzeugt. $v \leftarrow \arg\min_u fill_{G^+}(u)$ Vervollständige $N(v)$ zu einer Clique und entferne v. Füge v am Ende des Eliminationsschemas π an. Wiederhole, bis alle Knoten aus G eliminiert sind.

  min-fill

  Wir nennen diese Heuristik für Baumweite *min-fill* . Tatsächlich approximiert diese Heuristik auch eine Lösung zum Problem MinimumFillIn, welches minimale Chordalisierungen (Def. 1) enthält und von dem Yannakakis die **NP**-Vollständigkeit zeigte [49].

- Eine andere Möglichkeit, wenige Füllkanten zu erzeugen, ist die Nachbarschaft $N(v)$ des Knoten v, welcher eliminiert wird, klein zu halten:

Wir konstruieren π: Wähle einen Knoten v minimalen Knotengrades aus G: $v \leftarrow \arg\min_u deg_{G^+}(u)$; vervollständige $N(v)$ zu einer Clique und lösche v. Der nächste Knoten in π ist v. Wiederhole, bis alle Knoten entfernt sind.

Wir nennen diese Heuristik *min-degree* . Sie wurde ursprünglich von Markowitz für die Numerik auf dünn-besetzten Matrizen entwickelt [29]. min-degree

- Der bekannte Algorithmus MAXIMUMCARDINALITYSEARCH besucht alle Knoten eines Graphen G in einer Reihenfolge, so dass stets der Knoten mit der größten Anzahl zuvor besuchter Nachbarn als nächtes besucht wird. Eine Anwendung des Algorithmus ist, Graphen auf Chordalität zu prüfen. Für nicht notwendigerweise chordale Graphen impliziert der Algorithmus folgendes Eliminationsschema π:

  Besuche einen beliebigen Knoten v und markiere ihn. Der Knoten v ist der letzte Knoten in π. Wähle nun einen Knoten mit der größten Anzahl markierter Nachbarn und füge ihn *am Anfang* von π ein. Wiederhole, bis alle Knoten besucht wurden.

  Wir bezeichnen diese Heuristik mit *max-cardinality* . max-cardinality

Wir erhalten Baumzerlegungen für G. Wenngleich es heuristisch möglich ist, ein perfektes Eliminationsschema für einen chordalen Obergraphen H mit gleicher Baumweite zu G zu finden, erhalten wir keine Garantien für die Güte der Approximation. Das bedeutet: Schlimmstenfalls hat die so bestimmte Baumzerlegung zu H eine exponentiell schlechtere Weite als $\mathbf{tw}(G)$ [48]. Weitere vorgeschlagene Heuristiken ergeben sich aus Kombinationen der ersten Beiden [29]. Folge der selben Konstruktion und wähle den zu eliminierenden Knoten wie folgt:

- $v \leftarrow \arg\min_u deg_{G^+}(u) + fill_{G^+}(u)$ (*min-fill+degree*) min-fill+degree
- $v \leftarrow \arg\min_u fill_{G^+}(u) - deg_{G^+}(u)$ (*sparsest subgraph*) sparsest subgraph
- $v \leftarrow \arg\min_u deg_{G^+}(u) + \frac{1}{n^2} fill_{G^+}(u)$
- $v \leftarrow \arg\min_u fill_{G^+}(u) + \frac{1}{n} deg_{G^+}(u)$

3.1.7 k-Bäume

Direkt thematisch anschließend zum vorigen Abschnitt möchten wir die zweitbekannteste Charakterisierung der Baumweite über k-Bäume angeben.

Definition 23 (k-Baum, partieller k-Baum)**.** Ein k-Baum ist ein ungerichteter Graph G für den gilt,

- G ist eine $k+1$ Clique, oder
- G ist ein Graph der aus einem k-Baum G' entsteht, indem genau ein neuer Knoten hinzugefügt wird, der adjazent zu jedem Knoten einer k-Clique in G' ist.

Ein Teilgraph eines k-Baumes heißt partieller k-Baum.

Wir sehen direkt, dass k-Bäume spezielle chordale Graphen sind, denn die induktive Definition erzeugt nur simpliziale Knoten die aufeinanderfolgend ein perfektes Eliminationsschema ergeben.

Die im Induktionsanfang der Definition genannte $k+1$ Clique heiße X_0. In jedem Konstruktionsschritt eines k-Baums wird mit dem hinzugefügten Knoten eine neue $k+1$ Clique erzeugt. Wir nummerieren die Cliquen aufsteigend mit jedem Induktionsschritt $X_1, X_2, X_3 \ldots$. Wird im Schritt i ein neuer Knoten mit einer bereits existierenden k Clique verbunden, so ist diese Teilgraph der in einem vorigen Schritt $X_j, 0 \leq j < i$ konstruierten $k+1$ Clique. Wir merken uns die Kante (j, i) in einer Relation F.

$T_i = ([0, i]_{\mathbb{N}}, F)$ ist ein zusammenhängender, einfacher Graph mit $i+1$ Knoten und i Kanten und somit ein Baum.

Lemma 15. *$(\{X_k, 0 \leq k \leq i\}, T_i)$ ist eine Baumzerlegung des k-Baumes nach Konsturuktionsschritt i.*

Beweis. Jeder Knoten v und jede Kante e des k-Baums nach Schritt i ist in einem der über X_k induzierten Teilgraphen enthalten. Angenommen ein Knoten w fehle in einem X_m zu m auf dem Pfad von a nach b in T_i, wobei $w \in X_a \cup X_b$:

- Falls $m = 0$ so wurde die Definition von k-Bäumen dahingehend verletzt, dass der selbe Knoten mehrmals hinzugefügt wurde.
- Andernfalls sei m unter den Knoten auf dem a-b-Pfad minimal mit der Eigenschaft, dass $w \notin X_m$. $m \neq 0$ impliziert, dass ein Vorgängerknoten von m existiert, dessen Behälter w enthält. w wurde in Konstruktionsschritt m jedoch nicht als Teil der neu konstruierten Clique gewählt. Somit kann keine der Cliquen, die aus X_m konstruiert werden, w enthalten. Entweder X_a oder X_b enthält nicht w. Widerspruch.

□

Lemma 16. *Aus jeder Baumzerlegung zu einem Graphen G der Weite k kann ein k-Baum konstruiert werden, der G als Teilgraph enthält.*

Beweis. Sei σ das perfekte Eliminationsschema zur induzierten Chordalisierung H von G wie in Definition 21. Durchlaufe nun σ rückwärts. Verbinde die ersten $k+1$ besuchten Knoten zu einer Clique. Fülle dann für jeden weiteren Knoten v die Nachbarschaft $N_H(v)$ beliebig durch weitere zuvor besuchte Knoten auf sodass gilt $|N_H(v)| = k$.

Wir zeigen, dass diese Konstruktion korrekt ist: Bemerke, dass die Annahme $|N_H(v)| > k$ zum Widerspruch führt, denn dann enthielte H eine Clique der Größe $> k+1$. Aber die Baumzerlegung zu G und H hat Weite k. Also erfüllt die obige Konstruktion die induktive Definition von k-Bäumen. Offensichtlich enthält der so konstruierte Graph auch G, da wir alle Knoten und Kanten von G erhalten. G ist ein partieller k-Baum. □

Korollar 8. *Die Baumweite $\boldsymbol{tw}(G)$ ist die kleinste Zahl k, für die ein ungerichteter Graph G ein partieller k-Baum ist.*

Beweis. Folgt aus Lemma 15, Lemma 16, und Korrolar 6 □

Eine optimale Baumdekomposition zu einem k-Baum hat die folgende Struktur:

Korollar 9. *Sei G ein Graph mit Baumweite k. G hat eine Baumzerlegung $(\{X_i | i \in I\}, T = (I, F))$ mit Weite k und folgenden Eigenschaften:*

- *Für alle $i \in I$ gilt: $|X_i| = k + 1$.*
- *Für alle $(i, j) \in F$ gilt: $|X_i \cap X_j| = k$.*

Jeder ungerichtete Graph lässt sich für ein gewisses k in einen k-Baum einbetten. k-Bäume sind genau die maximalen Graphen mit Baumweite k [50]. Fügt man zu einem k-Baum eine Kante hinzu, so erhält man einen neuen Graphen mit höherer Baumweite. In andern Worten: k-Bäume sind bezüglich der Anzahl von Kanten die schlechtest möglichen Chordalisierungen, welche die Baumweite erhalten.

Korollar 10. *Ein Graph $G = (V, E)$ ist genau dann ein k-Baum, wenn $|V| > k$ und ein perfektes Eliminationsschema $\sigma = (v_1, \ldots, v_n)$ für G existiert, sodass für $i = 1, \ldots, n - k$ der Knoten v_i adjazent zu einer k-Clique in $G[\{v_i, \ldots, v_n\}]$ ist.*

Beweis. Folgt aus der Konstruktion im Beweis zu Lemma 16 □

Folgende Satz verstärkt die Aussage von Korrolar 9 um die entgegengesetzte Implikationsrichtung:

Satz 9 (Patil [51])**.** *Die folgenden Aussagen sind äquivalent:*

- *G ist ein k-Baum*
- *G ist chordaler Graph und alle maximalen Cliquen haben Größe $k + 1$.*
- *G ist chordaler Graph und alle minimalen Separatoren haben Größe k.*

Abschließend fassen wir die hier erläuterten Äquivalenzen zu chordalen Graphen zusammen:

Satz 10. *Die folgenden Aussagen sind äquivalent:*

- *Der Graph G ist chordal.*
- *Der Graph G enthält keinen Kreis der Länge mindestens 4 ohne Sehne (Definition 1).*
- *Sei C beliebiger induzierter Teilgraph von G: Falls C ein Kreis ist, dann ist C ein Dreiecksgraph (Definition 1).*
- *Der Graph G ist Schnittgraph einer Menge von Teilbäumen eines Baums (Satz 8).*

- *Jeder induzierte Teilgraph von G enthält einen simplizialen Knoten (Satz 1).*
- *Es gibt ein perfektes Eliminationsschema zu G (Satz 2).*
- *Es gibt ein* EliminationGame *ohne Füllkanten für G (Lemma 14).*
- *Jeder minimale a-b-Separator ist eine Clique (Satz 31).*

3.1.8 Kombinatorische Resultate auf Zufallsgraphen

Natürlich erwarten wir nicht von jedem Graphen, dass er tatsächlich Baum- oder Pfad-ähnlich wäre. Die folgende Betrachtung zeigt, auf welchen „Anteil aller Graphen“ dies tatsächlich zutrifft. Jeder Graph ist ein partieller k-Baum für ein gewisses k.

Wir können Wahrscheinlichkeitsverteilungen über Graphen betrachten. Die Zufallsvariable ist in diesem Fall ein sogenannter Zufallsgraph. Zwei Modelle für diese Betrachtung sind Gilbert-Graphen $G(n,p)$, für die jede mögliche Kante mit Wahrscheinlichkeit p existiert, und Erdős-Rényi-Graphen $G(n,m)$, welche genau m zufällig verteilte Kanten und n Knoten besitzen. Die Wahrscheinlichkeit der Gültigkeit von Grapheigenschaften, kann in diesen Modellen als Grenzwert bestimmt werden.

Nur wenige Graphen haben beschränkte Baumweite: Wir betrachten Graphen mit n Knoten und $m \geq \delta n$ Kanten. Für diese Graphen kann eine asymptotische untere Schranke der Baumweite gezeigt werden.

Satz 11 (Kloks [31])**.** *Für $\delta \geq 1,18$ hat fast jeder Zufallsgraph mit $m \geq \delta n$ Kanten eine Baumweite in $\Theta(n)$.*

Die Häufigkeit von Graphen in $G(n,m)$ mit geringerer Baumweite geht gegen 0 für $\lim_{n\to\infty}$.

In folgendem Satz etablieren wir, dass viele für uns interessante Graphfamilien tatsächlich nur einen winzigen Anteil aller möglichen Graphen stellen. Wir betrachten Graphenfamilien $\mathcal{G}$ die abgeschlossen bzgl. der Bildung von Minoren sind (z.B. Planare Graphen) dann gilt:

Satz 12 (Kloks [31])**.** *Sei $\mathcal{G}$ Graphenfamilie abgeschlossen bzgl. der Bildung von Minoren. Fast jeder Graph mit n Knoten und m Kanten, wobei $m \geq 1,18\,n$, ist nicht in $\mathcal{G}$ enthalten.*

3.1.9 Résumé zu Baumweite

Neben dem in Abschnitt 3.1.2 besprochenen Approximationen, existieren viele weitere Algorithmen mit unterschiedlichen beweisbaren Eigenschaften. Einen Überblick gibt Tabelle 3.1.

Es bleibt noch zu klären, ob es für die Komplexität parametrisierter Algorithmen einen Unterschied macht, ob eine Baumzerlegung oder lediglich die Baumweite des Eingabegraphen übergeben wird.

Bemerkung 10. Sei A ein **FPT**-Algorithmus mit einem Graphen G als Eingabe und einem Parameter k als obere Schranke der Baumweite. Falls $\mathbf{tw}(G) \leq k$, so können wir die benötigte Baumzerlegung in linearer **FPT**-Zeit berechnen.

TABELLE 3.1: Algorithmen für Baumweite [30]

Sei k die Baumweite des Graphen G. Jede Zeile nennt einen Algorithmus, welcher die Baumweite für G berechnet, mit Laufzeit in $f(k) \cdot g(n)$.
Für Näherungsalgorithmen ist die Weite der approximierten Lösung angegeben.

max. Weite	$f(k)$	$g(n)$	Veröffentlichung
exakt	$\mathcal{O}(1)$	$\mathcal{O}(n^{k+1})$	Arnborg, Corneil und Proskurowski [52]
$4k+3$	$\mathcal{O}(3^{3k})$	n^2	Abschnitt 3.1.2, Robertson und Seymour [28]
$8k+7$	$2^{\mathcal{O}(k \log k)}$	$n \log^2 n$	Lagergren [53]
$8k+\mathcal{O}(1)$	$2^{\mathcal{O}(k \log k)}$	$n \log n$	Reed [54]
exakt	$k^{\mathcal{O}(k^3)}$	n	Bodlaender [55]
$4,5k$	$\mathcal{O}(2^{3k}k^{3/2})$	n^2	Amir [33]
$(3+\frac{2}{3})k$	$\mathcal{O}(2^{3,6982k}k^3)$	n^2	Amir [33]
$\mathcal{O}(k \log k)$	$\mathcal{O}(k \log k)$	n^4	Amir [33]
$\mathcal{O}(k\sqrt{\log k})$	$\mathcal{O}(1)$	$n^{\mathcal{O}(1)}$	Feige, Hajiaghayi und Lee [56]
$3k+4$	$2^{\mathcal{O}(k)}$	$n \log n$	Bodlaender, Drange, Dregi u. a. [30]
$5k+4$	$2^{\mathcal{O}(k)}$	n	Bodlaender, Drange, Dregi u. a. [30]

Bei der Konstruktion eines theoretischen **FPT**-Algorithmus A mit Parameter Baumweite können wir also stets annehmen, dass eine optimale Baumzerlegung gefunden werden kann, ohne Sorge, die Klasse **FPT** zu verlassen. Falls A zu einem anderen Parameter **p** als der Baumweite **tw** entworfen muss, so hängt es von der Beziehung zwischen diesen Parametern ab, ob ein Zeuge für die Baumweite in der angestrebten Laufzeit produziert werden kann. Siehe hierzu Abschnitt 3.7.

In Implementierungen werden die Schritte „Berechnung einer Zerlegung“ und „Lösung eines Graphenproblems mithilfe der Zerlegung“ separat gekapselt, sowie die optimale Baumzerlegung in der Regel durch eine Approximation ersetzt. In letzterem Fall kann, abweichtend zu theoretischen Betrachtungen, die Optimalitätseigenschaft in der Implementierung nicht genutzt werden.

Wir fassen die in dieser Arbeit besprochenen Äquivalenzen zu Baumweite und Pfadweite zusammen.

Satz 13. *Sei G=(V,E) ein Graph und $k \geq 0$. Die folgenden Aussagen sind äquivalent.*

- *Die Baumweite von G ist höchstens k*
- *G ist ein partieller k-Baum.*
- *G ist Teilgraph eines chordalen Graphen mit maximaler Clique der Größe höchstens $k+1$*
- *Es existiert ein Eliminationsschema $\sigma = v_1 \dots v_n$ zu G, sodass gilt: Die Baumweite von G beträgt $\max |G[v_i, \dots, v_n] \cap N(v_i)|$*

Für weitere Äquivalenzen siehe [15, Theorem 1.].

Satz 14. *Sei $G = (V, E)$ und $k \geq 0$. Die folgenden Aussagen sind äquivalent:*

- *Die Pfadweite $\boldsymbol{pw}(G)$ ist höchstens k.*
- *G ist ein partieller k-Pfad.*
- *Die Intervallweite (engl. interval thickness) von G ist höchstens $k+1$.*
- *Die Knotenseparatorzahl von G ist höchstens k. siehe Abschnitt 4.2.*

Für weitere Äquivalenzen siehe [15, Theorem 2.].

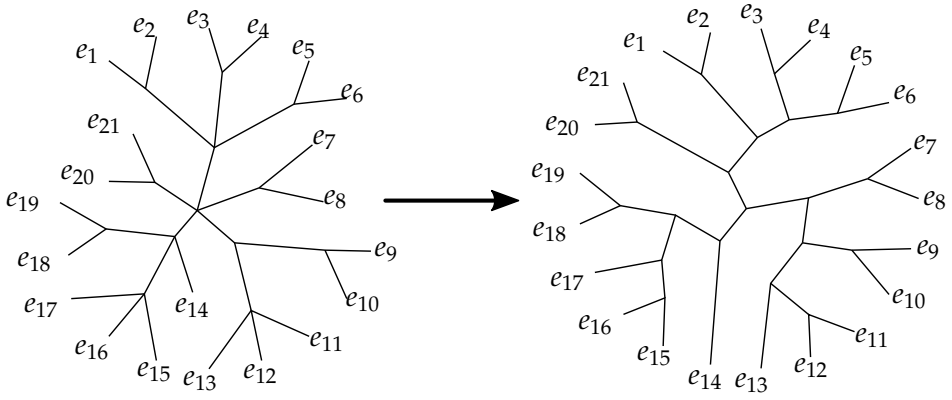
3.2 Zweigzerlegung und Zweigweite

Eine Zweigzerlegung ist eine hierarchische Partitionierung der Kanten eines Graphen $G = (V, E)$. Jeder Knoten habe Grad mindestens 1. Sei (E_L, E_R) eine Partition der Kantenmenge E. Wir benennen drei Knotenmengen:

- $Links(E_L, E_R) = \{ v \in V \mid (\exists x: vx \in E_L) \wedge (\nexists x: vx \in E_R) \}$
- $Mitte(E_L, E_R) = \{ v \in V \mid (\exists x: vx \in E_L) \wedge (\exists x: vx \in E_R) \}$
- $Rechts(E_L, E_R) = \{ v \in V \mid (\nexists x: vx \in E_L) \wedge (\exists x: vx \in E_R) \}$

Nach Konstruktion haben weder Knoten aus $Links(E_L, E_R)$ noch aus $Rechts(E_L, E_R)$ Nachbarn in der jeweils anderen Menge. Es folgt $Mitte(E_L, E_R)$ ist ein Knotenseparator für jede Wahl einer Kantenpartition (E_L, E_R). Wir Partitionieren nun rekursiv die Kantenmenge E bis im zugehörigen Rekursionsbaum $T = (I, F)$ nur noch einelementige Kantenmengen in den Blättern verbleiben. So lassen sich Kanten in E über eine Bijektion $\sigma: E \to \{ i \in I \mid d_T(i) = 1 \}$ den Blättern von T zuordnen. Abbildung 3.16 (links) zeigt einen solchen Rekursionsbaum mit den zugeordneten Kanten von G an den Blättern von T.

ABBILDUNG 3.16: Zwei Zerlegungsbäume zu der Kantenmenge E eines ungerichteten Graphen $G = (V, E)$



Definition 24.

- Eine *partielle Zweigzerlegung* (engl. partial branch decomposition) eines Graphen $G = (V, E)$ ist ein Paar $(T = (I, F), \sigma)$. T ist ein Baum. σ ist eine Bijektion zwischen E und den Blättern von T. partielle Zweigzerlegung
- $(T = (I, F), \sigma)$ heißt *Zweigzerlegung* , falls zusätzlich für alle Knoten $i \in I$ des Baums T gilt: $d_G(i) = 1 \vee d_G(i) = 3$. Ein solcher Baum heißt auch *ternär*. Zweigzerlegung

Eine triviale partielle Zweigzerlegung ist ein Sterngraph, dessen Blätter bijektiv den Kanten eines beliebigen Graphen zugeordnet werden. Jede partielle Zweigzerlegung lässt sich in eine gewöhnliche Zweigzerlegung umwandeln, indem alle Knoten j mit Grad ≥ 4 durch einen ternären Baum ersetzt werden, der $N(j)$ als Blattmenge besitzt. Abbildung 3.16 (rechts) zeigt eine so erzeugte Zweigzerlegung.

Jede Zweigzerlegung kodiert nun $m-2$ verschiedene Kantenpartitionen des Graphen $G=(V,E)$. Wenn wir die Kante $f \in F$ entfernen, zerfällt T in zwei gewurzelte Binärbäume T_L und T_R. Diese heißen *Zweige.* Wir erhalten die Kantenpartition (E_L, E_R) als die Urbildmengen unter σ der Blätter von T_A und T_B. Daraus ergeben sich wiederum $m-2$ verschiedene Knotenpartitionen wie oben.

Zweig

Wir versuchen solche Zweigzerlegungen zu finden, bei denen die Anzahl der Knoten, welche von Kanten in beiden Mengen E_L und E_R berührt werden, möglichst klein ist. Dazu halten wir folgende Definitionen fest.

Definition 25. Sei $G=(V,E)$ ein ungerichteter Graph mit Zweigzerlegung $(T=(I,F),\sigma)$.

Weite einer Kante

(i) Die *Weite* von $f \in F$ ist die Kardinalität der Menge $Mitte(E_L, E_R)$ bei Entfernung der Kante f.

Die Definition (i) ist äquivalent zu:

(i') Die *Weite* (auch Ordnung) von $f \in F$ ist die Anzahl solcher Knoten $v \in V$, zu denen adjazente Kanten $vw, vx \in E$ existieren, sodass f in dem Weg von Blatt $\sigma(vw)$ zu Blatt $\sigma(vx)$ in T enthalten ist.

Weite

(ii) Die *Weite* einer Zweigzerlegung ist das Maximum über die Weite aller Kanten $f \in F$.

Zweigweite

(iii) Die *Zweigweite* (engl. branch-width) $\mathbf{bw}(G)$ ist die minimale Weite über alle Zweigzerlegungen von G.

Zweigweite und Baumweite lassen sich durch den jeweils anderen Parameter nach oben und unten beschränken. Das bedeutet zu dem einen Parameter liegt der jeweils andere Parameter immer innerhalb eines konstanten Intervalls.

Satz 15 (Robertson und Seymour [57])**.** *Sei $G=(V,E)$ ein Graph mit $E \neq \emptyset$. Gegeben eine Baumzerlegung zu G der Weite k kann eine Zweigzerlegung zu G der Weite $k+1$ konstruiert werden. Gegeben eine Zweigzerlegung der Weite b kann eine Baumzerlegung der Weite höchstens $\frac{3}{2}b$ berechnet werden. Genauer gilt für die Baumweite und Zweigweite:*

$$\max(\boldsymbol{bw}(G),2) \leq \boldsymbol{tw}(G)+1 \leq \max\left(\left\lfloor \frac{3}{2}\cdot \boldsymbol{bw}(G)\right\rfloor, 2\right)$$

Wir nennen speziell die Zweigweite von planaren Graphen.

Satz 16 (Fomin und Thilikos, [23])**.** *Sei G ein planarer Graph mit n Knoten. Für die Zweigweite gilt $\boldsymbol{bw}(G) \leq \alpha\sqrt{n}$ mit $\alpha < 2{,}122$.*

3.2.1 BRANCHWIDTH und verwandte Entscheidungsprobleme

Das klassische Entscheidungsproblem, ob $G = (V, E)$ Zweigweite höchstens b hat, wenn sowohl G als auch b teil der Eingabe sind, ist **NP**-vollständig [58].

Der von Fomin, Mazoit und Todinca, 2009 vorgestellte Algorithmus konstruiert eine Zweigzerlegung mithilfe einer Variante von minimalen Chordalisierungen und potentiell maximaler Cliquen in exponentieller Zeit $(2\sqrt{3})^n \cdot n^{\mathcal{O}(1)}$ [59]. Beschränken wir unsere Eingabe auf die Familie der Graphen mit Zweigweite höchstens b, so gilt:

Satz 17 (Bodlaender und Thilikos, [60])**.** *Für jedes feste b existiert ein Linearzeit-Algorith-mus, der prüft ob ein Graph G Zweigweite $\leq b$ hat und eine Zweigzerlegung minimaler Weite konstruiert. b-*BRANCHWIDTH *ist in* **P**[6].

Obwohl **tw** und **bw** durch Satz 15 so nah beieinander liegen, scheinen sie sich algorithmisch sehr unterschiedlich zu verhalten. Die Beschränkung des Problems auf die Familie der planaren Graphen verändert die Komplexität des Problems. Dies ist insbesondere überraschend, da für PLANARTREEWIDTH kein solches Resultat bekannt ist.

PLANARBRANCHWIDTH
Eingabe: Ein ungerichteter planarer Graph $G = (V, E)$ und eine natürliche Zahl b.
Parameter:
Frage: Hat G Zweigweite höchstens b?

Satz 18 (Seymour und Thomas [58, (7.3) Algorithm.])**.** *Es existiert ein Algorithmus der in Zeit $\mathcal{O}(\,(n+m)^2\,)$ arbeitet und Graphen der Zweigweite mindestens b erkennt.* PLANAR-BRANCHWIDTH *ist in* **P**.

Darüber hinaus bilden Graphen mit Zweigweite höchstens b eine Graphenfamilie, die abgeschlossen unter der Bildung von Minoren ist [57]. Mit dem Satz von Robertson-Seymour [61] folgt, dass eine Charakterisierung durch verbotene Minoren existiert, durch die das parametrisierte Problem BRANCHWIDTH mit einer oberen Schranke der Zweigweite als Parameter in **FPT** liegt [7]. Zur Vollständigkeit definieren wir auch dieses parametrisiere Problem:

BRANCHWIDTH
Eingabe: Ein ungerichteter Graph $G = (V, E)$.
Parameter: Eine natürliche Zahl b.
Frage: Hat G Zweigweite höchstens b?

[6] Äquivalent auch BRANCHWIDTH $\in$ **XP**

[7] Außerdem sind die kanonisch parametrisierten Probleme PATHWIDTH und TREEWIDTH in **FPT**, über den selben Satz

3.2.2 Approximationsalgorithmen für Zweigweite

Es existiert ein Algorithmus von [62], der ähnlich wie Algorithmus 3 arbeitet und entweder eine approximative Zweigzerlegung eines Graphen G der Weite höchstens $3b$ produziert oder die korrekte Aussage produziert, dass gilt $\mathbf{bw}(G) \geq b$ [62].

Wir geben einen kurzen Überblick zu dem in den in [63] dargestellten Algorithmus zur Berechnung einer Zweigzerlegung kleiner Weite von Cook und Seymour. Diese Veröffentlichung greift ebenfalls auf Resultate von Robertson und Seymour [57] zurück um die Korrektheit des Algoirthmus zu zeigen.

Der Algorithmus FINDBRANCHDECOMPOSITIONBYSPLITTING startet auf einem Sterngraphen und „spaltet“ iterativ innere Knoten in zwei neue Knoten die durch eine Kante verbunden werden. Wir erhalten partielle Zweigzerlegungen. Wenn schließlich alle inneren Knoten den Grad 3 aufweisen, ist die Iteration abgeschlossen. Unter der Spaltung von Knoten verstehen wir genau die Gegenoperation zur Kontraktion von Kanten. Wir verteilen die Nachbarschaft $N(v)$ eines Knoten v auf zwei neue Mengen X und Y, welche als die Nachbarschaft neuer Knoten x und y gesetzt werden. xy wird eine neue Kante der partiellen Zweigzerlegung. Die Spaltung der inneren Knoten erfolgt anhand verschiedener Zerlegungsregeln. Dabei werden stets „sichere Spaltungen“ bevorzugt, welche die Weite der Zweigzerlegung gegenüber der Zweigweite des Eingabegraphen nicht erhöhen. Falls dies nicht mehr möglich ist wird der nächste zu spaltende Knoten anhand einer Eigenvektorheuristik ausgewählt.

3.2.3 Algorithmen mit Parameter Zweigweite

Zweigzerlegungen eigenen sich ebenso gut wie Baumzerlegungen als Grundlage zur Lösung von Optimierungsproblemen durch dynamische Programmierung. Für viele Graphoptimerungsprobleme existieren Approximationsalgorithmen die entweder auf Zweig- oder Baumzerlegungen arbeiten. Für die Zweigweite von planaren Graphen sind exakte Algorithmen mit polynomieller Laufzeit bekannt, weshalb für den Entwurf von Algorithmen auf planaren Graphen Zweigzerlegungen gegenüber Baumzerlegungen bevorzugt werden. Cook und Seymour demonstrieren in [63] ein Approximationsverfahren für TSP. Der Parameter Zweigweite kommt auch in SAT-Algorithmen zum Einsatz.[59].

Fomin und Thilikos führt in Argumentation für Zweigzerlegungen darüber hinaus an, dass besserer Schranken für Approximationen möglich seien und sich keine hohen Konstanten ergäben, die in der $\mathcal{O}$-Notation versteckt wären. In [65] wird hingegen gezeigt, dass BRANCHWIDTH eingeschränkt auf einige Graphenfamilien **NP**-schwer bleibt, während das Baumweiteproblem für die selben Graphenfamilien in polynomieller Zeit entschieden werden kann.

DOMINATINGSET und WEIGHTEDINDEPENDENTSET sind **FPT** für den Parameter Zweigweite [44].

3.2.4 Charakterisierung von Graphen geringer Zweigweite

Graphen mit geringer Zweigweite können folgendermaßen charakterisiert werden:

Satz 19 (Robertson und Seymour [57])**.**

- *Ein Graph G hat Zweigweite 0 genau dann, wenn jede Zusammenhangskomponente von G höchstens eine Kante enthält. Diese Graphfamilie lässt sich über die verbontenen Minoren P_2 und C_3 beschreiben*
- *Ein Graph G hat Zweigweite höchstens 1 genau dann, wenn jede Zusammenhangskomponente höchstens einen Knoten v mit $deg(v) \geq 2$ hat. Dies sind genau die Graphen in denen alle ZHKn Sterngraphen sind. Die verbotenen Minoren dieser Familie sind P_3 und C_3.*
- *Ein Graph G hat Zweigweite höchstens 2 genau dann, wenn G nicht K_4 als Minor enthält. Betrachte eine Kantenpartion $(E_1, E_2, \ldots)$, sodass jeder Teilgraph über den Kanten E_i größtmöglich und 2-zusammenhängend gewählt ist. Dann gilt: G hat Zweigweite höchstens 2, falls alle 2-zusammenhängenden Teilgraphen serien-parallele Graphen sind.*
- *Ein Graph G hat Zweigweite höchstens 3 genau dann, wenn er nicht den vollständigen Graphen K_5, den Wagner-Graphen M_8, den Würfelgraphen Q_3, oder den Graphen zum regulären Oktaeder als Minoren enthält.*

3.3 Baumtiefenzerlegung und Baumtiefe

Bei der Baumtiefe handelt es sich um einen Graphenparameter der in der Literatur unter verschiedenen Namen auftaucht. Ursprünglich wurde er als Invariante auf ungerichteten Graphen unter dem Namen „cycle rank“ eingeführt. Die Baumtiefe kann ganz analog definiert werden; jedoch auf ungerichteten Graphen:

Definition 26. Sei $G = (V, E)$ ein ungerichteter Graph mit ZHKn $(G_i)_i$. Die Baumtiefe $\mathbf{td}(G)$ ist rekursiv definiert via:

$$\mathbf{td}(G) = \begin{cases} 1, & \text{falls } |G| = 1. \\ 1 + \min_{v \in V} td(G - v), & \text{falls } G \text{ zusammmenhängend und } |G| > 1. \\ \max_i td(G_i), & \text{sonst.} \end{cases}$$

Folgende äquivalente Definition charakterisiert die Baumtiefe ohne Rekursion [66], [67]:

Satz 20 (z.B. [67, Lemma 6.1.])**.** *Die Baumtiefe $\mathbf{td}(G)$ eines Graphen G ist die minimale Höhe eines Waldes F (mit Wurzelknoten für jeden Baum) sodass $G \subseteq clos(F)$.*

Die Höhe eines Waldes ist die maximale Länge eines Pfades zwischen einem Wurzelknoten und einem Blattknoten plus 1. $clos(F)$ bezeichnet die transitive Hülle eines Graphen H, in der die Kantenrelation um alle indirekten Vorgänger- Nachfolger Beziehungen erweitert wird, sodass $clos(F)$ der Vergleichbarkeitsgraph zu dieser partiellen Ordnung $\prec_F$ ist. Wir benennen nun die einzelen Sturkturen, welche in Satz 20 implizit vorkommen.

Definition 27.

Baumtiefenzerlegung
- Eine *Baumtiefenzerlegung* eines Graphen $G = (V, E)$ ist ein gewurzelter Wald F über der selben Knotenmenge V, sodass für jede Kante $uv \in E$ gilt: Entweder ist u (möglicherweise indirekter) Vorgänger von v in F, also $u \prec_F v$ oder andersherum $v \prec_F u$[8].

Tiefe
- Die *Tiefe* einer Baumtiefenzerlegung ist die maximale Länge eines Knotenpfades von einer Wurzel zu einem Blatt von T.

Baumtiefe
- Die *Baumtiefe* von G ist die minimale Tiefe über alle Baumtiefenzerlegungen von G.

Die Intuition hinter letzterer Definition lässt sich in folgender Weise erläutern: Wir ordnen jedem Knoten von G einen Knoten in einem neu konstruierten Wald F zu. Dabei muss zu jeder Kante $\{v, w\}$ in G ein Pfad zwischen dem zugeordneten Knoten w in F und einem (indirekten) Vorgängerknoten v existieren, oder andersherum. Aus der Konstruktion folgt unmittelbar: Jede Zusammenhangskomponente von G entspricht einem Baum im Wald F.

Einige Beispiele verdeutlichen die obigen Definitionen:

Beispiel 11.

- Wir suchen die Baumtiefe des vollständigen Graphen auf n Knoten. Der zugehörige Wald ist ein Baum. Konstruieren wir diesen Induktiv über die Knoten, so müssen wir in jedem Schritt den neuen Knoten als Vorgänger oder Nachfolger aller anderen Knoten setzen um die Definition zu erfüllen. Es folgt, der zugehörige Baum ist ein Pfad und es gilt $\mathbf{td}(K_n) = n$.

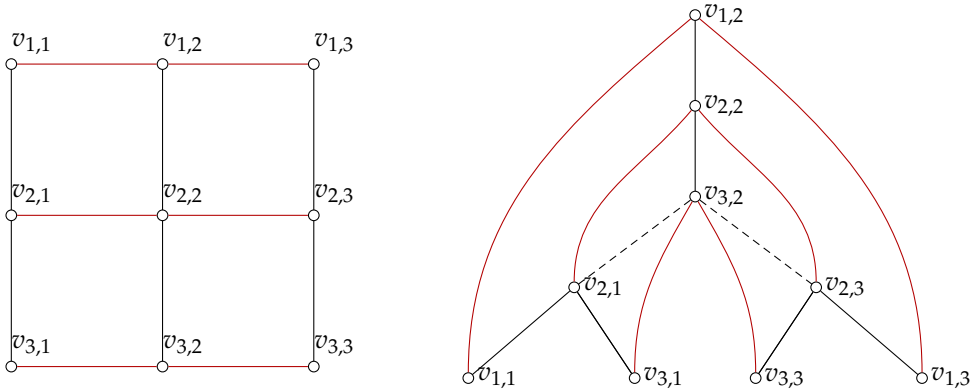
- Wir nennen den Graphen $K_{1,n}$ der Sterngraph S_n mit $n + 1$ Knoten. Gesucht ist die Baumtiefe dieses Graphen. Es gilt $\mathbf{td}(S_n) = \min_{v \in V} \mathbf{td}(S_n - v)$. Beachte dass die Baumtiefe durch das Entfernen eines Knotens um genau 0 oder 1 fällt. Wähle den einzigen inneren Knoten von S_n als v, dann zerfällt $S_n - v$ in n einzelne Knoten. Es gilt $\mathbf{td}(S_n - v) = \max_i \mathbf{td}(R_i)$; und für alle ZHKn R_i folgt $\mathbf{td}(R_i) = 1$. Also war unsere Wahl von v optimal. Insgesamt folgt: Alle Sterngraphen haben Baumtiefe 2.

- Ein Gittergraph $G_{n,m}$ ist ein ungerichteter Graph, dessen Zeichnung ein rechteckiges $n \times m$ Gitter bildet. Dem Gittergraphen $G_{3,3}$ können wir den Wald in Abbildung 3.17 zuordnen. Der Graph $G_{3,3}$ hat Baumtiefe 4.

[8] Diese Eigenschaft charakterisiert $G \subseteq clos(F)$

ABBILDUNG 3.17: Baumtiefe des Graphen $G_{3,3}$

Graph $G_{3,3}$ (links). Eine optimale Baumtiefenzerlegung F (rechts, schwarze Kanten). In $clos(F)$ durch indirekte Vorgänger-Nachfolger-Beziehungen (rot) abgedeckte Kanten von $G_{3,3}$. Zusätzliche Kanten in F, welche in $G_{3,3}$ nicht vorkommen (gestrichelt).



Wir untersuchen nun die Baumtiefe der Pfadgraphen P_n. Eine zugehörige optimale Baumtiefenzerlegung ist in Abbildung 3.18 dargestellt.

Lemma 17. *Für den Pfad P_n gilt: $\boldsymbol{td}(P_n) = \lceil \log_2(n+1) \rceil$.*

Beweis. Wir zeigen die Aussage mit vollständiger Induktion über n:

IA. $n = 1$: Nach Definition gilt $\mathbf{td}(P_1) = 1 = \lceil \log_2(2) \rceil$

IS. $\leq n \to n+1$:

$$\begin{aligned}
\mathbf{td}(P_{n+1}) &\overset{\text{Def. 26}}{=} \min_{1 \leq k \leq n-1} \left[1 + \max(\mathbf{td}(P_k), \mathbf{td}(P_{(n+1)-k}))\right] \\
&\overset{\heartsuit}{=} \begin{cases} 1 + \max(\mathbf{td}(P_{\frac{n+1}{2}+\frac{1}{2}}), \mathbf{td}(P_{\frac{n+1}{2}-\frac{1}{2}})) & \text{, falls } n+1 \text{ ungerade} \\ 1 + \max(\mathbf{td}(P_{\frac{n+1}{2}}), \mathbf{td}(P_{\frac{n+1}{2}}) & \text{, falls } n+1 \text{ gerade} \end{cases} \\
&= \begin{cases} 1 + \mathbf{td}(P_{\frac{n+2}{2}}) \overset{\text{IB.}}{=} 1 + \log_2\left(\left\lceil \frac{n+2}{2} \right\rceil\right) = \log_2\left(\left\lceil 2\frac{n+2}{2} \right\rceil\right) & \text{, falls } n \text{ gerade} \\ 1 + \mathbf{td}(P_{\frac{n+1}{2}}) \overset{\text{IB.}}{=} 1 + \log_2\left(\left\lceil \frac{n+1}{2} \right\rceil\right) = \log_2\left(\left\lceil 2\frac{n+1}{2} \right\rceil\right) & \text{, falls } n \text{ ungerade} \end{cases} \\
&= \lceil \log_2(n+2) \rceil
\end{aligned}$$

Die Gleichheit $\heartsuit$ gilt, da die Logarithmusfunktion $\lceil \log_2(n+1) \rceil$ monoton steigt und für alle $P_k < n+1$ die Induktionsbehauptung gilt.

□

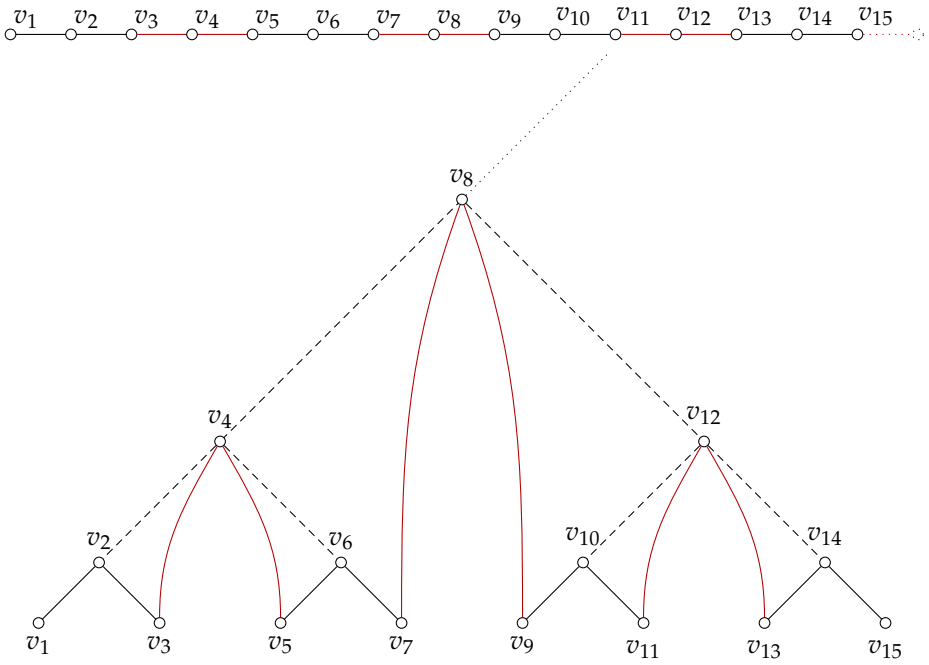
Vergleichbar mit der Intuition, dass die Baumweite die Ähnlichkeit eines zusammenhängenden Graphen zu einem Baum ausdrückt, können wir Baumtiefe als die Ähnlichkeit eines zusammenhängenden Graphen zu einem Sterngraph interpretieren.

Lemma 18. *Falls H ein Minor des Graphen G ist, so gilt $\boldsymbol{td}(H) \leq \boldsymbol{td}(G)$.*

Beweis. Sei F eine Baumtiefenzerlegung von $G = (V, E)$ optimaler Weite $\mathbf{td}(G)$. Es gilt $G \subseteq \text{clos}(F)$. Falls wir eine Kante e oder einen Knoten u aus G entfernen, so sind

ABBILDUNG 3.18: Baumtiefe des Graphen G_{15}

Graph P_{15} (oben). Eine optimale Baumtiefenzerlegung F (unten, schwarze Kanten). In $clos(F)$ durch indirekte Vorgänger-Nachfolger-Beziehungen (rot) abgedeckte Kanten von P_{15}. Zusätzliche Kanten in F, welche in G_{15} nicht vorkommen (gestrichelt). Die gepunkteten Kanten deuten an, wie die Baumtiefenzerlegung für längere Pfade fortgesetzt werden kann.



$G - e$ bzw. $G - u$ Teilgraphen von G und Minoren nach Definition. Wir betrachten also die Kontraktion einer Kante vw aus G, sodass der entstehende Knoten den Namen v behält. Dieser Graph heiße H. Sei o.B.d.A. v (indirekter) Vorgänger von w in F. H enthält genau die neuen Kanten $E_{\text{aug.}} = \{ vx \mid x \in N_G(w) \}$ Es gilt $H = (V, E \cup E_{\text{aug.}}) - w$. Bemerke, dass diese neuen Kanten bereits in der transitiven Hülle $clos(F)$ enthalten sind. Zu zeigen bleibt, dass $clos(F)$ eingeschränkt auf die Knoten von H transitiv bleibt. Dies folgt jedoch insbesondere als Korollar aus Satz 20. Konstruiere alternativ die Baumtiefenzerlegung F' aus F in welcher die Kinder von w genau mit dem Elternknoten von w verbunden sind. Dies ist eine Baumtiefenzerlegung für H und die Länge eines maximalen Wurzel-Blatt-Pfades kann sich höchstens verkürzen. Auf beiden Wegen folgt die Aussage durch die Iteration des obigen Arguments entspechend der Minorendefinition. □

Korollar 11. *Falls P_n ein Minor des Graphen G ist, so gilt $\lceil \log_2(n+1) \rceil \leq \boldsymbol{td}(G)$.*

Wir können einige Graphen in der die Familie $\mathcal{F}$ der Graphen mit Baumtiefe höchstens d identifizieren.

Lemma 19. *Falls P_{d+1} nicht als Minor in einem Graphen G enthalten ist, so hat G Baumtiefe höchstens d.*

Beweis. Für jede Zusammenhangskomponente von G führen wir einen Tiefendurchlauf durch und erzeugen so einen Wald F mit gewurzelten Teilbäumen R_i als Zusammenhangskomponenten. Da für alle i der Pfad P_{d+1} nicht als Minor in R_i enthalten

ist, hat der im Tiefendurchlauf erzeugte Baum R_i höchstens die Höhe d. F ist eine Baumtiefenzerlegung der Tiefe höchstens d. Optimale Baumtiefenzerlegungen haben daher Tiefe $\leq d$. Also gilt $\mathbf{td}(G) < d + 1$. □

Definition 28. Seien $\overrightarrow{G}_i$ die starken Zusammenhangskomponenten eines *gerichteten* Graphen $\overrightarrow{G}$ *zyklische Rang* $\mathbf{r}(\overrightarrow{G})$ ist rekursiv wie folgt definiert: zyklischer Rang

$$\mathbf{r}(\overrightarrow{G}) = \begin{cases} 0, & \text{falls } \overrightarrow{G} \text{ azyklisch, i.e. keinen gerichteten Kreis enthält} \\ 1 + \min_{v \in V} \mathbf{r}(\overrightarrow{G} - v), & \text{falls } \overrightarrow{G} \text{ genau eine starke ZHK bildet und } |\overrightarrow{G}| > 1. \\ \max_i \mathbf{r}(\overrightarrow{G}_i), & \text{sonst.} \end{cases}$$

Der zyklische Rang ist eine Verallgemeinerung der Baumtiefe auf gerichtete Graphen. Auf jedem ungerichteten Graphen G sind starke ZHKn genau die gewöhnlichen ZHKn G_i. Wir können G als als einen gerichteten Graphen $\overleftrightarrow{G}$ mit symmetrischer Kantenrelation auffassen. Dieser ist genau dann azyklisch wenn jede ZHK von G nur einen Knoten enthält. In diesem Fall gilt $\mathbf{r}(\overleftrightarrow{G}) = 0$; während für die Baumtiefe von G gilt $\mathbf{td}(G) = \max_i \mathbf{td}(G_i) = \max_i 1 = 1$. Für Graphen der Form $\overleftrightarrow{G}$, sind die Definition von **r** und **s** darüber hinaus identisch. Es folgt:

Bemerkung 11. Für alle ungerichteten Graphen G gilt $\mathbf{td}(G) - 1 = \mathbf{r}(\overleftrightarrow{G})$.

Der Parameter Baumtiefe **td** wird eingesetzt um Färbbarkeitsprobleme auf dünnbesetzten Graphen zu parametrisieren [67].

3.4 Degeneriertheit

Wir betrachten einen weiteren Parameter nach Lick und White [68] für dünn-besetzte Graphen. Verschiedene Autoren ziehen unterschiedliche Maße heran, um zu definieren, wann eine Matrix dünn-besetzt ist [50]. Das Problem liegt darin definitoisch scharfe Grenzen zwischen den Begriffen „dünn“ und „dicht“, sodass alle Graphen abgedeckt sind und gleichzeitig Familien bekannter Graphen in der „richtigen“ Graphklasse liegen. Im Rahmen dieser Arbeit ignorieren wir diese Zweiteilung[9] und verlangen, dass für eine *Familie dünn-besetzter Graphen* $\mathcal{F}$ gelte: $G = (V, E) \in \mathcal{F}$, impliziert $|E| \in \mathcal{O}(|V|)$. Familie dünn-besetzter Graphen

Definition 29. Die *Degeneriertheit* eines Graphen G ist die kleinste natürliche Zahl k, sodass jeder nicht-leere Teilgraph $H \sqsubseteq G$ mindestens einen Knoten mit Grad höchstens k enthält. Wir nennen G dann k-degeneriert. Degeneriertheit

Der selbe Parameter hat in der Literatur weitere Namen wie engl. „degeneracy“, „linkage“, „coloring number + 1“, oder „d-core number“.

[9] in dichte und dünne Adjazenzmatrizen oder Dreiteilung in Graphen welche „bounded-size“, „nowhere-dense“ und „somewhere-dense“ sind [50]

Beispiel 12.

- Planare Graphen sind 5-degeneriert.
- Bäume sind 1-degeneriert, denn jeder nicht-leere induzierte Teilgraph ist ein Baum oder Wald entweder nur isolierte Knoten mit Grad 0 oder es existieren Blätter mit Grad 1.

Lemma 20. *Sei G ein k-degenerierter Graph. Es gilt $\omega(G) \leq k+1$.*

Beweis. Angenommen es existiere eine Clique C der Größe $k+2$ in G. Die Clique C ist zu sich selbst induzierter Teilgraph und alle Knoten in C haben Grad $k+1$. Widerspruch. □

Die k-Degeneriertheit eines Graphen lässt sich über ein Eliminationsschema charakterisieren:

Lemma 21. *Ein Graph $G = (V, E)$ ist k-degeneriert, genau dann wenn es ein Eliminationsschema σ der Knoten gibt, in der jeder Knoten v vor höchstens k seiner Nachbarn $N(v)$ liegt.*

Beweis. Sei G k-degeneriert. Wir konstruieren σ indem wir wiederholt Knoten v wählen, die einen Grad $\leq d$ haben und diese aus G eliminieren. Da alle induzierten Teilgraphen nach Definition Degeneriertheit höchstens k haben, existiert ein solches v in jedem Schritt. Offensichtlich sind zum Zeitpunkt der Elimination von v nur noch höchstens k Nachbarn in G enthalten. σ listet die Knoten in der Reihenfolge ihrer Elimination von G auf. Falls andersherum σ ein Eliminationsschema mit obiger Eigenschaft ist, dann gilt für jeden induzerten Teilgraphen $G[W]$ zu einer Knotenteilmenge W, Das σ eingeschränkt auf die Knoten von W den gesuchten Knoten mit Grad $\leq d$ als Ersten im verbleibenden Eliminationsschema enthält. □

Bemerkung 12 (Nešetřil und Mendez [67, Proposition 3.2.]). Wir wählen die Formulierung des obigen Lemma aufgrund der vertrauten Terminologie. Analog können wir beweisen, dass ein Graph G k-degeneriert, ist genau dann wenn $G = (V, E)$ azyklisch zu einem gerichteten Graphen $\overrightarrow{G} = (V, \overrightarrow{E})$ orientiert werden kann, sodass für alle Knoten v gilt $deg^-(v) \leq k$

Das obige Resultat bleibt wahr, wenn wir in jedem Eliminationsschritt den Knoten v minimalen Grades $\delta(G)$ auswählen [69]. Mit dieser Strategie können wir die Degeneriertheit eines Graphen sehr einfach berechnen.

Algorithmus 4 (GREEDYDEGENERACY)**.**

Eingabe: ungerichteter Graph $G = (V, E)$

Ausgabe: Eliminationsschema σ, Degeneriertheit k

1: $k \leftarrow \infty$
2: **while** $V \neq \emptyset$ **do**
3: $\quad k \leftarrow \min\{d, \delta(G)\}$

4: Wähle v beliebig aus $\{ v \mid deg(v) = \delta(G) \}$
5: $G \leftarrow G - v$
6: Füge v am Ende von σ an.
7: **return** σ, d

Lemma 22 (Batagelj und Zaversnik [69])**.** *Der Algorithmus* GreedyDegeneracy *ist korrekt. Es gilt* Degeneracy $\in$ **P**.

Durch geschickte Wahl der Datenstrukturen kann dieser Algorithmus in linearer Laufzeit über der Anzahl der Knoten implementiert werden [70].

Wie wir für dünn-besetzte Graphen erwarten, ist die Anzahl möglicher Cliquen in k-degenerierten Graphen klein.

Lemma 23. *Sei G ein k-degenerierter Graph. Dann hat G höchstens $\binom{k}{t-1}n$ Cliquen der Größe t. Die Anzahl der Cliquen insgesamt beträgt höchstens $2^k n$ und ist somit linear in n.*

Beweis. Sei ein Eliminationsschema σ für G gegeben und C eine beliebig gewählte t-Clique in G. Die Knoten von C in der Reihenfolge von σ sei $v_1, v_2, \dots v_t$. Jeder Knoten in σ kann potentiell eine t-Clique starten, wir haben also n Möglichkeiten v_1 zu wählen. Aus den $|N(v_1)| \leq k$ Positionen welche als Nachbarn in σ auf v_1 folgen, müssen wir weitere $t-1$ Positionen für die verbleibenden Knoten von C berücksichtigen. Hierfür zählen wir $\binom{k}{t-1}$ Möglichkeiten. Insgesamt ergibt sich die Anzahl möglicher t-Cliquen zu höchstens dem Produkt $\binom{k}{t-1}n$. Als obere Schranke für der Anzahl aller Cliquen erhalten wir

$$\sum_{t=2}^{k+1} = \binom{k}{t-1} n = 2^k n$$

□

Satz 21. *Die Degeneriertheit eines Graphen G beträgt höchstens $\boldsymbol{tw}(G)$ bzw. $\boldsymbol{pw}(G)$*

Beweis. Falls ein Graph G die Baumweite bzw. Pfadweite höchstens k hat, dann ist G ein partieller k-Baum. Der zugehörige k-Baum sei H. Da H chordal ist, existiert ein perfektes Eliminationsschema σ für H. Jeder Knoten v liegt in σ vor höchstens k seiner Nachbarn, sowohl für H als auch G. Daher ist G höchstens k-degeneriert. Es folgt die Aussage. □

Beispiel 13. Als Beispiel für einen **FPT**-Algorithmus für den Parameter Degeneriertheit betrachten wir eine Modifikation des Bron–Kerbosch-Algorithmus, welcher alle maximalen Cliquen der Grapheninstanz $G = (V, E)$ bei Aufruf BronKerboschPivot$(G, (V, \varnothing, \varnothing))$ produziert.

Algorithmus 5 (BronKerboschPivot). **Eingabe:** ungerichteter Graph $G = (V, E)$, Partition P, R, X einer Teilmenge von V
Ausgabe: alle maximalen Cliquen in $G[P \cup R \cup X]$
1: **if** $P \cup X = \varnothing$ **then**

2: Gebe die maximale Clique R zurück.
3: Wähle einen Knoten $u \in P \cup X$ als Pivot.
4: **for** $v \in P \backslash N(u)$ **do**
5: BronKerboschPivot$(P \cap N(v), R \cup \{v\}, X \cap N(v))$
6: $P \leftarrow P \backslash \{v\}$
7: $X \leftarrow X \cup \{v\}$

Für die Korrektheit dieses Algorithmus siehe [71]. Schritt 3 reduziert die Anzahl der rekursiven Aufrufe. Die Laufzeit dieses Algorithmus beträgt schlechtestenfalls $\mathcal{O}(3^{\frac{n}{3}})$ und ist in klassischer Komplexitätsanalyse optimal [72], unter der Annahme **P** $\neq$ **NP**. In jedem rekursiven Aufruf gibt der Algorithmus genau alle Cliquen in $P \cup X$ aus, welche in $G[P \cup R \cup X]$ maximal sind. Die Menge R induziert zu jedem Zeitpunkt eine Clique, welche aber nicht notwendigerweise maximal ist. Die Menge $P \cup X$ enthält genau die Knoten, welche Nachbarn aller Knoten in R sind. Dabei enthält P solche Knoten die möglicherweise (engl. **p**otentially) die R zu einer größeren Clique erweitern und X solche Knoten für die dies garantiert nicht der Fall ist (engl. **e**xclude). In der Scheife des Algorithmus wird für jeden Knoten v aus P überprüft, ob v zu R oder zu X hinzugefügt wird.

Eppstein, Löffler und Strash ersetzen den ersten Aufruf des Algorithmus durch folgende Prozedur [70]:

Algorithmus 6 (BronKerboschDegeneracy). **Eingabe:** ungerichteter Graph $G = (V, E)$, Eliminationsschema σ, in der jeder Knoten v vor höchstens k seiner Nachbarn $N(v)$ liegt.

1: **for** v_i in der Reihenfolge $\sigma = (v_1, v_2, v_3, \ldots)$ **do**
2: $P \leftarrow N(v_i) \cap \{v_{i+1}, \ldots, v_n\}$
3: $X \leftarrow N(v_i) \cap \{v_1, \ldots, v_{i-1}\}$
4: BronKerboschPivot$(P, \{v_i\}, X)$

Wir erinnern aus dem Grundlagenabschnitt 2.1.2, dass zu einem Eliminationsschema σ jede maximale Clique welche v_i enthält auch in $G[\{v_1, \ldots, v_i\}]$ enthalten sein muss. Der Algorithmus BronKerboschDegeneracy nutzt genau diese Information um die Knotenmenge V anhand von einem geeigneten Eliminationsschema σ in die Mengen P, R und X aufzuteilen. Um σ zu berechnen verwenden wir den Algorithmus GreedyDegeneracy. Die Laufzeit des zugehörigen parametrisierten Algorithmus ist in $\mathcal{O}(dn^{\frac{d}{3}})$ und hat Platzbedarf $\mathcal{O}(n)$ [70]. Die Autoren zeigen die Korrektheit und überprüfen das Resultat mit großem Erfolg experimentell auf dünn-besetzten Graphen [70]. Der Algorithmus hat einen geringeren Platzbedarf als vergleichbare Algorithmen zur Enumeration von maximalen Cliquen.

3.5 Cliquenweite

Wir haben gesehen, dass für einige **NP**-Schwere Probleme **FPT**-Algorithmen mit den Parametern Baum- und Pfadweite konstruiert werden können. Wir möchten nun

einen alternativen Typ von Zerlegungen betrachten. Wir ordnen diesen Zerlegungen einen neuen Graphenparameter zu, den wir Cliquenweite $\mathbf{cw}(G)$ nennen. Außerdem werden wir Abschätzungen zwischen Cliquenweite und anderen Graphenparametern kennenlernen. Die Cliquenweite ist eine Alternative zu Baumweite für dicht-besetzte Graphen, wie vollständige und vollständig bipartite Graphen.

Die Cliquenweite eines Graphen G wird über einen algebraischen Ausdruck definiert, der Schrittweise angibt, wie ein Graph G konstruiert wird. Zur Konstruktion des Graphen dürfen nur bestimmte Operationen verwendet werden. Konkret versuchen wir den algebraischen Ausdruck so zu konstruieren, dass möglichst wenige „Knotenlabel" benötigt werden. Das Konzept Cliquenweite mit seiner heutigen Bedeutung wurde von Courcelle, Engelfriet und Rozenberg, im Rahmen von kontextfreien Graphgrammatiken zur Manipulation von Hypergraphen eingeführt [73, Definition 5.5]. Die Autoren beobachteten später, dass falls eine Abfolge von Produktionen für einen Graphen G bekannt ist, sich klassisch **NP**-schwere Probleme auf G effizienter lösen lassen. Sie untersuchten diese Erkenntnis in [74] und führten die heutige Terminologie ein. Wir betrachten hier einige Resultate aus der Survay-Veröffentlichung [75]. Eine sehr aktuelle Übersicht zu Cliquenweite liefert [76], 2019.

markierter Graph

Definition 30. Ein *markierter Graph* (engl. labeled graph) ist ein Tripel $(V, E, \gamma) \in \mathcal{G}^\bullet$, wobei (V, E) ein Graph und γ eine Funktion $V \to \{1, \dots k\}$ ist. Wir sagen ein Knoten v hat Markierung (engl. label) i. Wir notieren markierte Graphen mit einem hochgestellten $^\bullet$ und den zugehörigen unmarkierten Graphen ohne $^\bullet$.

Definition 31. Sei $G^\bullet = (V, E, \gamma)$ ein markierter Graph, der rekursiv aus der Anwendung folgender Operationen hervorgeht:

- $\bullet_i$ Konstruiere einen neuen markierten Graphen $G^\bullet = (v, \varnothing, \{v \mapsto i\})$ welcher aus genau einem bisher nicht verwendeten Knoten v besteht, der die Markierung i erhält.
- $\oplus$ Konstruiere aus zwei oder mehr markierten Graphen $(G_i^\bullet)$, mit $G_i^\bullet = (V_i, E_i, \gamma_i) \forall i$, einen neuen Graphen G durch die Bildung der *disjunkten markierten Vereinigung*

  disjunkte markierte Vereinigung

$$\bigoplus_i G_i^\bullet = (\biguplus_i V_i, \biguplus_i E_i, \bigcup_i \gamma_i).$$

- $\eta_{i,j}$ Verbinde jeden Knoten mit Markierung i zu jedem Knoten mit Markierung j wobei $i \neq j$.
- $\rho_{i,j}$ Benenne die Markierung i zu j um.

Die Operation $\eta_{i,j}$ erzeugt einen vollständig bipartiten Teilgraphen in $G^\bullet$ zwischen den Knoten mit Markierung i und solchen mit Markierung j. Wir benötigen stets mindestens zwei unterschiedliche Markierungen, um eine Kante zu erzeugen.

Wenn wir $k = n = |V|$ wählen ist offensichtlich, dass wir jeden einfachen Graphen durch obige Operationen konstruieren können. Wende hierzu zunächst $\bullet_i$-Operationen an, um alle Knoten zu erzeugen und anschließend $\eta_{i,j}$-Operationen, um

jede einzelne Kante der vorgegebenen Kantenrelation zu rekonstruieren. Verwerfe anschließend alle Markierungen. Es stellt sich direkt die Frage wie viele unterschiedliche Markierungen mindestens benötigt werden, um einen vorgegeben G zu rekonstruieren.

Wir möchten nun markierte Graphen als algebraische Struktur $(\mathcal{G}^\bullet; \bullet_i, \oplus, \eta_{i,j}, \rho_{i,j})$, mit den oben gegebenen Operationen als Interpretation[10] zu den gleichnamigen inneren Verknüpfungssymbolen, auffassen. Zu jedem einfachen Graphen G existiert ein algebraischer Ausdruck, der angibt wie sich G mit diesen Operationen rekonstruieren lässt.

Definition 32. Ein **cw**, k-Ausdruck ist ein Ausdruck über $(\mathcal{G}^\bullet; \bullet_i, \oplus, \eta_{i,j}, \rho_{i,j})$, sodass für alle i und j gilt: $i \leq k \wedge j \leq k$.

Definition 33. Die Cliquenweite $\mathbf{cw}(G)$ eines markierten Graphen G ist die kleinste natürliche Zahl k sodass G durch einen **cw**, k-Ausdruck konstruiert wird.

Aus obiger Erläuterung entnehmen wir:

Bemerkung 13. Für jeden Graphen $G = (V, E)$ gilt $\mathbf{cw}(G) \leq |V|$. Für jeden Graphen G' mit mindestens einer Kante gilt $2 \leq \mathbf{cw}(G')$

Wie im Abschnitt über Baumweite entwickeln wir hier **cw**, k-Ausdrücke und die Cliquenweite zu einigen bekannten Graphen.

Beispiel 14. (i) Wir erhalten alle vollständig bipartiten Graphen $K_{a,b}$ bis auf Markierung durch den **cw**, 2-Ausdruck:

$$K^\bullet_{a,b} \equiv \eta_{1,2}(\underbrace{\bullet_2 \oplus \ldots \oplus \bullet_2}_{b \text{ mal}} \oplus \underbrace{\bullet_1 \oplus \ldots \oplus \bullet_1}_{a \text{ mal}})$$

(ii) Folgender rekursiver **cw**, 2-Ausdruck konstruiert (bis auf Markierung) den vollständigen Graphen K_n:

$$K^\bullet_n \equiv \begin{cases} \bullet_1 & \text{, falls } n = 0, \\ \rho_{2,1}(\eta_{2,1}(\bullet_2 \oplus K_{n-1})) & \text{, falls } n > 1. \end{cases}$$

(iii) Wir erhalten den Pfad P_n (bis auf Markierung) aus einem rekursiven **cw**, 3-Ausdruck:

$$P^\bullet_n \equiv \begin{cases} \bullet_2 & \text{, falls } n = 0, \\ \rho_{3,2}(\rho_{2,1}(\eta_{2,3}(\bullet_3 \oplus P_{n-1}))) & \text{, falls } n > 1. \end{cases}$$

(iv) Für alle Kreise C_n mit $n \geq 3$ konstruieren wir zunächst einen Pfad der Länge $n-1$ mit einem eindeutig markierten Startknoten und schließen den Kreis mit der letzten η-Operation. Wir erhalten $C_n \equiv A_n(n)$ für folgenden rekursiven **cw**, 4-Ausdruck A_n:

[10] Da die Argumente von $\oplus$ kommutativ und assoziativ sind können wir diese Operation durch eine zweistellige Operation $\oplus$ repräsentieren.

$$A_n(k) = \begin{cases} \eta_{1,3}(\bullet_1 \oplus \bullet_3) & \text{, falls } 2 = k < n, \\ \rho_{4,3}(\rho_{3,2}(\eta_{3,4}(\bullet_4 \oplus A_n(k-1)))) & \text{, falls } 2 < k < n, \\ \eta_{1,4}(\eta_{3,4}(\bullet_4 \oplus A_n(k-1))) & \text{, falls } n = k. \end{cases}$$

Insbesondere für C_5 und C_6 können wir Markierungen sparen, da folgende $\mathbf{cw}$, 3-Ausdrücke ebenfalls C_5 und C_6 produzieren:

$$C_5^\bullet \equiv \eta_{2,3}(\bullet_3 \oplus \rho_{3,1}(\eta_{1,3}(\bullet_3 \oplus \bullet_2 \oplus \eta_{1,2}(\bullet_1 \oplus \bullet_2))))$$

$$C_6^\bullet \equiv \eta_{2,3}(\bullet_3 \oplus \rho_{3,1}(\eta_{1,3}(\bullet_3 \oplus \eta_{1,2}(\bullet_1 \oplus \bullet_2) \oplus \eta_{1,2}(\bullet_1 \oplus \bullet_2))))$$

Wir zeigen die Optimalität der ersten drei Konstruktionen.

Lemma 24. *(i) Für vollständig bipartite Graphen gilt:* $\boldsymbol{cw}(K_{a,b}) = 2$

(ii) Für vollständige Graphen mit $n \geq 2$ *gilt:* $\boldsymbol{cw}(K_n) = 2$.

(iii) Für Pfade mit $n \geq 4$ *gilt:* $\boldsymbol{cw}(P_n) = 3$.

Beweis. (i) Gilt nach Bemerkung 13.

(ii) Ebenso.

(iii) Sei $n > 3$. $P_n^\bullet$ hat mindestens 4 Knoten. Wir zeigen, dass kein $\mathbf{cw}$, 2-Ausdruck existiert, der $P_n^\bullet$ konstruiert. Angenommen es gäbe einen solchen Ausdruck A. Da wir mit der Operation $\bullet_i$ höchstens einen neuen Knoten v mit Markierung i erzeugen können, existiert ein $\mathbf{cw}$, 2-Teilausdruck B der einen Teilgraphen $Q^\bullet \sqsubseteq P_n^\bullet$ mit $n-1$ Knoten produziert und sich mit höchstens $2 = \max d_{P_n}(2)$ neuen Kanten zu $P_n^\bullet$ verbinden lässt. $Q^\bullet$ hat mindestens 3 Knoten und höchstens die zwei Markierungen i und j mit $i \neq j$. Schließlich müssen wir die Operation $\eta_{i,j}$ verwenden um v mit $Q^\bullet$ zu verbinden. Es gilt $A = \ldots \circ \eta_{i,j} \circ \ldots \circ B$. Wir unterscheiden folgende Fälle:

- Der Teilgraph $Q^\bullet$ hat keine Knoten mit Markierung j, dann ist $P_n^\bullet$ nicht zusammenhängend. Widerspruch.
- Der Teilgraph $Q^\bullet$ genau einen Knoten y mit Markierung j; also mindestens zwei Knoten x und z mit Markierung i. Falls ein x,z-Pfad via y existiert, so hat y in $P_n^\bullet$ den Grad 3. Widerspruch. Andernfalls erhalten wir eine zusätzliche neue Kante xy oder yz. Die Operation $\eta_{i,j}$ produziert mindestens zwei neue Kanten. Dann enthält $P_n^\bullet$ einen Kreis. Widerspruch.
- Der Teilgraph $Q^\bullet$ hat genau zwei Knoten y,z mit Markierung j also mindestens einen Knoten x mit Markierung i. Der Widerspruch folgt analog.

Insgesamt folgt $\mathbf{cw}(P_n) = 3$ für $n > 3$.

□

Folgende Resultate geben wir hier ohne Beweis:

Bemerkung 14 (siehe z.B. [77]).

(iv) Für Kreise gilt: $\mathbf{cw}(C_n) = \begin{cases} 2 & \text{, falls } n \in \{3,4\} \\ 3 & \text{, falls } n \in \{5,6\} \\ 4 & \text{, sonst.} \end{cases}$

(v) Für alle Bäume T gilt $\mathbf{cw}(T) \leq 3$.

(vi) Ein Graph H ist genau dann ein Co-Graph, wenn gilt $\mathbf{cw}(H) \leq 2$

Wir sehen, dass die Cliquenweite für einige sehr dichte Graphen wie vollständige und vollständig bipartite Graphen sehr klein ist. Während die Baumweite auf dichten Graphen proportional mit der Größe der Eingabeinstanz steigt, lässt sich die Cliquenweite auch auf manchen Familien dicht-besetzter Graphen beschränkten. Wir können den Zusammenhang zu Graphfamilien beschränkter Baumweite genauer fassen:

Lemma 25 (Courcelle, Makowsky und Rotics [74][Theorem 5.5])**.** *Sei G ein einfacher Graph. Es gilt $\boldsymbol{cw}(G) \leq 2^{2\boldsymbol{tw}(G)+2} + 1$*

Lemma 26. *Für allgemeine einfache Graphen G existiert keine Funktion $f: \mathbb{N} \to \mathbb{N}$, sodass $\boldsymbol{tw}(G) \leq f(\boldsymbol{cw}(G))$.*

Beweis. Betrachte die Familie der vollständigen Graphen. Es gilt $\mathbf{cw}(K_k) = 2$, $\forall k$. Daher ist $f(\mathbf{cw}(K_k)) \leq n$ für ein festes $n_0 \in \mathbb{N}$, jedoch ist die Baumweite vollständiger Graphen unbeschränkt. □

Bemerkung 15. Für allgemeine Graphen können wir die Baumweite nicht nach oben durch einen Ausdruck über der Cliquenweite beschränken. Jedoch wird dies möglich, wenn wir uns auf bestimmte Graphfamilien beschränken. Siehe auch [78].

Die Algebraische Struktur $(\mathcal{G}^\bullet; \bullet_i, \oplus, \eta_{i,j}, \rho_{i,j})$ erlaubt uns die Wirkung einzelner Operationen auf die Cliquenweite zu untersuchen. Gurski bewies hierzu zahlreiche Sätze [79]. Beispielsweise erlaubt folgendes Lemma die Cliquenweite auf Zusammenhangskomponenten oder die Partitionsklassen eines vollständige bipartiten Graphen zurückzuführen.

Lemma 27 (Gurski [79])**.** *Seien zwei ungerichtete Graphen G_1 und G_2 gegeben. Die Operation $\otimes$ erzeuge den vollständig bipartiten Graphen zwischen den Konten von G_1 und denen von G_2. Es gilt: $\boldsymbol{cw}(G1 \oplus G2) = max(\boldsymbol{cw}(G1), \boldsymbol{cw}(G2))$ $\boldsymbol{cw}(G1 \otimes G2) = max(\boldsymbol{cw}(G1), \boldsymbol{cw}(G2), 2)$*

Die Cliquenweite über Manipulationen von Graphen, wie dem Hinzufügen oder Entfernen von Knoten und Kanten, Ersetzungen von Knoten durch Graphen, geweissen Produkt- und Summenkonstruktionen, Unterteilungen von Kanten, usw. zu untersuchen, ist wichtig um gute $\mathbf{cw}, k$-Ausdrücke zu finden; Denn die Cliquenweite scheint sich der effizienten algorithmischen Handhabung zu entziehen:

3.5.1 Das Problem CLIQUEWIDTH

Das zu Cliquenweite gehörige Entscheidungsproblem lautet:

CLIQUEWIDTH
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k
Frage: Hat G Cliquenweite höchstens k?

CLIQUEWIDTH ist **NP**-vollständig [80].

Es ist im allgemeinen **NP**-schwer Graphen der Cliquenweite k zu erkennen. Die Einschränkung auf Graphen mit Cliquenweite $k \leq 3$ erlaubt einen deterministischen Algorithmus mit polynomieller Laufzeit [81]. Von dem Entscheidungsproblem für größere feste Werte von k ist weder die Mitgliedschaft in **P**noch **NP**-schwere bekannt [82, Fact 21]. Die Frage ist nun seit mehr als 20 Jahren bekannt und ungelöst.

Im Jahr 2006 veröffentlichten Oum und Seymour einen Algorithmus, der für Graphen der Cliquenweite höchstens k einen $\mathbf{cw}, 2^{3k+2} - 1$-Ausdruck bestimmt. Der Algorithmus arbeitet in Zeit $\mathcal{O}(\, n^9 \log(n)\,)$ [83]. Durch diesen Algorithmus ist es möglich Algorithmen für Graphen beschränkter Cliquenweite zu konstruieren, die jedoch bereits auf kleinen Graphen hohe Laufzeiten aufweisen. Dabrowski, Johnson und Paulusma stellen heute fest: „Despite the usefulness of boundedness of clique-width, our understanding of clique-width itself is still very limited." Die Komplexität von CLIQUEWIDTH sei weiterhin selbst auf stark eingeschränkten Graphenfamilien wie Intervallgraphen offen [76].

3.5.2 Algorithmen mit Parameter Cliquenweite

Ein mächtiges Werkzeug, um die durch Cliquenweite parametrisierte Komplexität von Entschiedungsproblemen zu zeigen ist:

Satz 22. *(Courcelle, Makowsky und Rotics [74, Theorem 4],[82, Theorem 31]) Entscheidungsprobleme und Zählprobleme die durch eine Formel in monadischer Prädikatenlogik erster Stufe ausgedrückt werden können, mit Quantoren über den Knoten, aber nicht über Kanten, können auf jeder Klasse $\mathcal{C}$ durch ein festes k beschränkter Cliquenweite in polynomieller Zeit gelöst werden, falls zusätzlich zu dem Graphen G ein $\boldsymbol{cw}, k$-Ausdruck für G gegeben ist.*

Courcelle, Makowsky und Rotics weisen darauf hin, dass dieses Theorem im allgemeinen Fall keinen implementierbaren Algorithmus liefert [82]. Jedoch seien auf eingeschränkten Graphfamilien, welche über das Vorkommen des Pfades P_4 charakterisiert sind, praktisch ausführbare Algorithmen möglich.

Das Problem DOMINATINGSET ist in linearer Zeit auf Graphen beschränkter Cliquenweite lösbar [44]. Weiter Beispiele für Entscheidungsprobleme die lineare **FPT**-Algorithmen auf Cliquenweite erlauben sind 3-COLOURABILITY [84] und WEIGHTEDINDEPENDENTSET [44].

3.6 Modulare Zerlegung und modulare Weite

Die „Cliquen-Zerlegung“ in Abschnitt 3.5 durch **cw**, k-Ausdrücke ist eine Spezialisierung des allgemeineren Begriffs der modularen Zerlegung. Modulare Zerlegungen können außerdem neben k-zusammenhängenden Komponenten als weitere Möglichkeit gesehen werden Zusammenhangskomponenten zu generalisieren.

Eine modulare Zerlegung definiert eine hierarchische Struktur über Knotenteilmengen eines Graphen $G = (V, E)$. Die induzierten Teilgraphen über diesen Knotenteilmengen heißen *Module*[11].

Definition 34. Sei $G = (V, E)$ ein ungerichteter Graph.

Modul
- Der Teilgraph $X = (W, F) \sqsubseteq G$ ist heißt *Modul* , falls für jedes $v \in G - X$ gilt: entweder $v, w \in F$ oder $v, w \notin F$ für jedes $w \in W \backslash \{v\}$. Vorige Aussage ist äquivalent dazu, dass für alle $v \in X$ gilt: Die Nachbarschaft $N(v) \backslash W$ ist für alle v identisch.

maximal
- Ein Modul M heißt *maximal* bezüglch eines Teilgraphen $S \sqsubseteq G$, falls kein Obergraph von M, der Teilgraph von S ist, ein Modul bildet.

stark
- Ein Modul M heißt *stark* falls es sich mit keinem anderen Modul N überschneidet, d.h. die Knotenmengen von M und N sind disjunkt oder M enthält N vollständig oder wird von N enthalten.

G und $\bar{G}$[12] haben die selben Module, wie direkt aus der Symmetrie in der Definition erkennbar ist.

Anschaulich können wir die Knoten eines Moduls nicht unterscheiden, wenn wir lediglich eine „Sicht von Außen“ auf das Modul einnehmen, welches in dieser Anschauung eine Black-Box ist; Das bedeutet, wenn wir alle Knotennamen vergessen, können wir anhand aller Knoten außerhalb des Moduls nicht unterscheiden, ob im Modul unterschiedliche, oder überhaupt mehr als ein Knoten enthalten sind.

Aus dieser Anschauung geht die Intuition hervor, dass viele Eigenschaften von X bestimmt werden können ohne den gesamten Graphen G zu betrachten. X kann in einer näher zu bestimmenden Weise als unabhängig vom Rest des Graphen gesehen werden.

Jeder Graph $G = (V, E)$ hat die trivialen Module $(G[\{v\}], G[\varnothing])$ für jedes $v \in V$ und G selbst. An diesem Beispiel wird ebenso deutlich dass ein Modul andere Module enthalten kann. Weitere Beispiele für Module von G sind die ZHKn von G und die ZHKn von $\bar{G}$. Falls zwei Module X und Y disjunkt voneinander sind, so gibt es für ihre Beziehung zueinander genau zwei Möglichkeiten: Entweder alle Knoten aus X sind vollständig mit allen Knoten aus Y verbunden oder es gibt keine Kante zwischen irgendeinem Paar x,y mit $x \in X$ und $y \in Y$. Es wäre also möglich diese Beziehung

[11] manche Autoren definieren Module als Knotenteilmenge. Mit der hier verwendeten Definition als Teilgraph, sind direkt Begriffe wie „adjazent“, „Nachbarschaft“, etc. sauber vererbt

[12] $\bar{G}$ bezeichnet den Komplementgraphen in dem die Rollen von non-Kanten und Kanten getauscht sind.

abstrakter *mit nur einer Kante oder non-Kante* in einem Graphen abzubilden, dessen „Beutel“ die Module X und Y sind.

Eben diese Zerlegung wird vorgenommen, indem auf $G = (V, E)$ eine *modulare Partition* P bestimmt wird, welche aus disjunkten Knotenmengen geeigneter (starker) Module besteht und V überdeckt. Treu der algebraischen Notation kann nun ein *Quotient* G/P definiert werden welcher genau die Module als Äquivalenzklasse bezüglich der Partition bildet. Ebenfalls aus der Algebra überträgt sich die Konvention die Module nunmehr in dem Kontext des Quotienten G/P als *Faktoren* zu bezeichnen. Für die einzelnen Faktoren können rekursiv weitere Quotienten gebildet werden, bis schließlich nur noch triviale Module vorkommen. Die algebraische Darstellung von Modulen mithilfe der Quotientalgebra, liefert mächtige Werkzeuge zur Beweisführung in zugehörigen Sätzen und Lemmata. Siehe hierzu [85], [86].

modulare Partition

Quotient

Faktoren

Die Verbindung zur bisher eingeführten Terminologie kann direkter durch folgende konstruktive Definitionen hergestellt werden, die zu einer äquivalenten Definition der modularen Zerlegung eines Graphen führt. Wir gehen dazu den „algebraischen Weg“ rückwärts und beschreiben[13] wie wir Graphen durch die, zu den algebraischen Quotienten, „inversen“ Operationen konstruieren.

Definition 35. Seien $G = (V, E)$ und $G_i = (V_i, E_i)$ Graphen, der rekursiv aus der Anwendung folgender Operationen hervorgehen:

- Konstruiere einen neuen Graphen $G = (v, \varnothing)$, welcher aus genau einem bisher nicht verwendeten Knoten besteht. Dies ist ein Blatt im modularen Zerlegungsbaum.

$\biguplus$ Konstruiere aus zwei oder mehr Graphen (G_i), mit $G_i = (V_i, E_i) \forall i$, einen neuen Graphen G durch die Bildung der *disjunkten Vereinigung* :

disjunkte Vereinigung

$$\bigcupdot_i G_i = (\bigcupdot_i V_i, \bigcupdot_i E_i)$$

Im modularen Zerlegungsbaum korrespondiert diese Operation zu einem „Parallel-Knoten“ (ohne Kinder).

$\times$ Konstruiere aus zwei oder mehr Graphen (G_i), mit $G_i = (V_i, E_i) \forall i$, einen neuen Graphen durch die Bildung eines *vollständigen disjunkten Verbundes* :

vollständiger disjunkter Verbund

$$\bigtimes_i G_i = (\bigcupdot_i V_i, \bigcupdot_i E_i \cup \{ vw \mid \forall i, j : v \in V_i \wedge w \in V_j \rightarrow i \neq j \})$$

Dieser Knoten (ohne Kinder) heißt „Serien-Knoten“ im modularen Zerlegungsbaum von G.

$H \rightarrow G_i$ Wir nennen diese Operation eine Substitution. Sei $H = (W, F)$ ein beliebiger, ungerichteter Graph mit Knotenmenge $w_1, \ldots w_l$ und $G_i = (V_i, E_i), 1 \leq i \leq l$ Graphen, welche aus der Konstruktion dieser rekursiven Definition 35 hervorgehen.

[13] analog zur Konstruktion für Cliquenweite

Der neu konstruierte Graph G hat Knotenmenge $\biguplus_{1\le i\le l} V_i$ und Kantenmenge

$$\biguplus_i E_i \cup \{\, vw \mid \forall w_i w_j \in F : v \in V_i \wedge w \in V_j \to i \neq j \wedge 1 \le i \le j \le l \,\}$$

Wir erhalten G aus H, indem alle Knoten von H durch Graphen (G_i), die genau dann paarweise vollständig verbunden sind, wenn die zum Paar (G_i, G_j) gehörige Kante $w_i w_j$ in H vorkommt. Im Zerlegungsbaum repräsentieren wir diesen Konstruktionsschritt durch einen Knoten für die Substitutionsoperation und Kinderknoten für die Teilzerlegungsbäume der G_i.

Bemerke, dass alle Knoten des Zerlegungsbaums Ausdrücke über $\bullet$, $\uplus$ und $\times$ enthalten, während die Substitutionsoperation durch Verzweigung des Baumes repräsentiert sind.

Definition 36. Der Parameter Modular-width $\mathbf{mw}(G)$ ist definiert als der Maximalgrad des Baums zur einer optimalen modularen Zerlegung von G. Eine optimale modulare Zerlegung hat minimalen Maximalgrad. Aquivalent ist dies die minimale Anzahl l über alle modularen Zerlegungen der Faktoren (G_i) zur Substitutionsoperation.

Wenn wir den Zerlegungsbaum zuschneiden[14], so bilden die Graphen in den Blättern eine Zerlegung von G, sodass alle Knoten von G in genau einem dieser Graphen vorkommt. Dies sind genau die modularen Partition P der algebraischen Terminologie. Die Substitutionsoperation ist die „inverse“ Operation zu algebraischen Quotientenbildung G/P. Im optimalen Fall ist $P = (M_1, \ldots, M_l)$ so gewählt, dass die M_i maximale starke Module sind. Wir nennen eine solche Partition *maximal modular* .

maximale modulare Partition

Durch die Operationen $\bullet$, $\uplus$ und $\times$ kann genau die Familie der Co-Graphen konstruiert werden. Auf Co-Graphen können viele **NP**-schwere Probleme in linearer Zeit gelöst werden. Eine Eigenschaft die wir gerne ausnutzen. Die Substitutionsoperation ist offensichtlich sehr mächtig, denn zusammen mit der Operation $\bullet$ kann jeder ungerichtete Graphen (also auch jeder Co-Graphen) in einem Zerlegungsbaum der Tiefe 2 erzeugt werden. Dies entspricht in der algebraischen Sicht der Wahl, die einelementigen Teilmengen von V[15] als modulare Partition für die Quotientenbildung zu nutzen. Jedoch ist die Tiefe der Zerlegung nicht maßgeblich sondern der Maximalgrad, welcher in diesem Beispiel n wäre. Dies ist im Allgemeinen keine nützliche modulare Zerlegung von G. Eine optimale Zerlegung wird daher die Operationen $\bullet$, $\uplus$ und $\times$ so häufig wie möglich verwenden, statt die Substitutionsoperation zu nutzen.

Überraschender Weise verwenden wir die Substitutionsoperation unter dieser Vorgabe genau dann, wenn die $G_i = M_i$ maximale starke Module von G sind. I.e. *die* maximale modulare Partitionen ist eindeutig in G (siehe z.B. [87]). Dies führt uns auf

[14] wir kappen ganze Zweige oder bilden beliebige Vereinigungen der Knotenmengen welche in den Operationen $\uplus$ oder $\times$ repräsentiert sind

[15] genauer die zugehörigen Graphen, welche nur einen Knoten enthalten.

die eindeutige modulare Zerlegung, welche erstmalig durch Gallai, 1967 beschrieben wurde. Die oben genannte Definition, die wir leicht auf unsere Terminologie angepasst haben ist äquivalent [43].

Darüber hinaus kann die modulare Zerlegung eines Graphen in linearer Zeit berechnet werden [86], also ebenso der Parameter $\mathbf{mw}(G)$.

In der letzten Dekade wurden verschiedene effiziente Algorithmen zu Entscheidungsproblemen mit Parameter **mw** entworfen. Diese Arbeiten mit dynamischer Programmierung oder engl. „integer linear programming“ [43], [87]. Wir schließen unsere Exploration der Graphenparameter im nächsten Abschnitt dadurch ab, zu sehen warum **mw** ein interessanter Parameter für Graphenprobleme ist.

3.7 Über die Beziehungen zwischen verschiedenen Graphenparametern

Um festzustellen welche Graphenparameter einander ähneln, können wir diese entsprechend ihrer Eigenschaft untersuchen, je auf bekannten Graphklassen beschränkt oder unbeschränkt zu sein.

Gegeben zwei Graphenparameter **p** und **q** sagen wir **p** *dominiert* **q**, gdw. eine Funktion f existiert, sodass $\forall G\colon \mathbf{p}(G) \leq f(\mathbf{q}(G))$. Beachte das f abgesehen von $\mathbf{q}(G)$ keine weiteren Parameter, insb. nicht $n = |V|$ oder $m = |E|$, als Argumente erhält. dominiert

Beispiel 15. Wir betrachten die Graphenfamilie der Bäume: Baumweite eines Baumes durch 2 beschränkt, jedoch kann die Pfadweite eines Baumes beliebig groß sein. Beispielsweise gilt für die Familie der vollständigen Binärbäume $\mathcal{B}$, dass $\mathbf{pw}(B_d) \to_{d\to\infty} \infty, B_d \in \mathcal{B}$ der Höhe d. Somit gilt **pw** dominiert nicht **tw**, denn

$$\forall f\colon \exists \text{ Baum } B\colon \underbrace{f(2)}_{\text{konstant}} = f(\mathbf{tw}(B)) < \mathbf{pw}(B)$$

Falls **q** durch **p** dominiert wird, aber nicht andersherum, so sagen wir in diesem Kontext. **p** ist der *allgemeinere* Parameter und **q** der speziellere Parameter. Falls für eine gewisse Graphklasse $\mathcal{C}$ der Parameter **q** beschränkt ist (siehe Definition 13), so ist für $\mathcal{C}$ auch der Parameter **p** via der Funktion f beschränkt. Wir drücken aus Abschnitt 3.1 bekannte Resultate und unsere Beobachtung in Beispiel 15 in dieser neuen Terminologie aus: allgemeinerer Parameter

Korollar 12. *Der Parameter Baumweite* ***tw*** *ist allgemeiner als der Parameter Pfadweite* ***pw****. Jede Graphenfamilie $\mathcal{C}$ mit beschränkter Pfadweite hat auch beschränkte Baumweite. Andererseits können wir jedes Problem, das mit dem Parameter Baumweite in* **FPT** *liegt, auch auf Graphen beschränkter Pfadweite in* **FPT** *lösen.*

Auf analoge Weise erhalten wir folgende Resultate:

Lemma 28. *Der Parameter Degeneriertheit ist allgemeiner als der Parameter Baumweite* ***tw****. Jede Graphenfamilie $\mathcal{C}$ mit beschränkter Baumweite hat auch beschränkte Degeneriertheit. Andererseits können wir jedes Problem, das mit dem Parameter Degeneriertheit in* **FPT** *liegt, auch auf Graphen beschränkter Baumweite in* **FPT** *lösen.*

Beweis. Planare Graphen haben beschränkte Degeneriertheit, jedoch existieren Familien von planaren Graphen deren Baumweite nicht beschränkt ist. Beispielsweise sind die quadratischen $k \times k$-Gitter 5-degeneriert (Beispiel 12) aber haben Baumweite k (Lemma 3). Andererseits können wir mit Satz 21 die Degeneriertheit durch die Baumweite beschränken. Also dominiert die Degeneriertheit eines Graphen die Baumweite aber nicht andersherum. □

Korollar 13. *Der Parameter Cliquenweite* ***cw*** *ist allgemeiner als der Parameter Baumweite* ***tw****. Jede Graphenfamilie $\mathcal{C}$ mit beschränkter Baumweite hat auch beschränkte Cliquenweite. Andererseits können wir jedes Problem das auf dem Parameter Cliquenweite polynomielle Laufzeit hat auch auf Graphen beschränkter Baumweite in polynoieller Zeit lösen.*

Beweis. Die Familie der vollständigen Graphen hat beschränkte Cliquenweite jedoch nicht beschränkte Baumweite (Lemma 26). Jedoch können wir die Cliquenweite eines Graphen durch eine obere Schranke über der Baumweite beschränken (Lemma 25) □

äquivalenter Parameter

Falls für zwei Parameter **p** und **q** sowohl gilt **q** dominiert **p** als auch **p** dominiert **q**, dann sagen wir **p** und **q** sind *äquivalente Parameter* .

Satz 23 (z.B. [76])**.** *Baumweite* ***tw*** *und Zweigweite* ***bw*** *sind äquivalent.*

Satz 24 (Oum, 2017 [88]; siehe auch [76])**.** *Cliquenweite* ***cw****, modulare Weite* ***mw****, und Rangweite* ***rw*** *sind äquivalent.*

Obwohl viele Fragen zur Komplexität von Cliquenweite offen sind und wir bisher nur geringen algorithmischen Nutzen aus diesem Parameter schöpfen können, ist ist die modulare Weite, welche mit zugehöriger Zerlegung in **P**berechnet werden kann, äquivalent.

4 Algorithmische Konzepte für Graphenparameter

Die Verwendung der oben eingeführten Zerlegungen ermöglicht oft konstruktive Beweise für polynomielle Laufzeitschranken parametrisierter Algorithmen [89]. Jedoch sind die konstanten Faktoren der Schranken meist so hoch, dass der Algorithmus in der Praxis nicht genutzt werden kann. Wir können davon ausgehen, dass ein Algorithmus, dessen Laufzeit bei großzügiger Prozessorleistung die Lebensdauer unseres Sonnensystems überschreitet, nie ausgeführt wird. Wir möchten hier Ansätze vorstellen, welche die Berechnung von Graphenparametern praktisch möglich machen.

Ein theoretisch optimaler **FPT**-Algorithmus für Baumweite wurde bereits 1996 von Hans L. Bodlaender vorgestellt [27], ist jedoch aufgrund der hohen Konstanten in der $\mathcal{O}$-Notation für eine praktische Implementierung unbrauchbar.

Algorithmen aus konstruktiven Beweisen dienen der Einordnung von Problemen in Komplexitätsklassen. Mit den Beweisideen werden oft neue algorithmische Ansätze entwickelt, die zunächst für Implementierungen ungeeignet scheint, jedoch eventuell Jahre später ihren Weg in eine ausführbare Software findet. Auch hängt die praktische Nutzbarkeit eines neuen Ansatzes auch vom Fortschritt in einem unabhängigen Fachgebiet ab: Mit den heute verfügbaren SAT-Solvern können mit einer geeigneten Kodierung des jeweiligen Problems Instanzen beachtlicher Größe praktisch gelöst werden. In den aktuellen Veröffentlichungen [90], [91], werden Kodierungen für SAT vorgestellt. Kask & Lam verwendeten als Teilnehmer der PACE-Challenge in 2016 einen SAT- basierten parallelisierten Algorithmus zur Approximationen der Baumweite und gewannen knapp den 1. Platz vor einem Separator-basierten Algorithmus [92]. In einer Veröffentlichung von 2019 berichten die Autoren von SAT-Kodierungen für Baumtiefe Auch dynamische Programmierung wird als Ansatz zur Berechnung von Baumzerlegungen verwendet. Die Arbeitsgruppe von Bodlaender stellt die meisten Veröffentlichungen zu dynamischer Programmierung für Baumweite. Der DP-Algorithmus von Hiromu Ohtsuka & Hisao Tamaki [93] errang den 2. Platz für exakte Baumweite [94] in PACE 2017. Für eine umfassende Einführung in die dynamische Programmierung siehe [95].

Insgesammt kann sich herausstellen, dass Instanzen in der Praxis einfacher sind als im allgemeinen Fall angenommen werden kann und dadurch die tatsächlichen Laufzeiten geringer sind als erwartet [89]. Welche Eigenschaften die praktischen Instanzen „einfacher“ machen, ist wiederum Gegenstand der Forschung mit dem

Ziel, die beobachtete kleine Laufzeit auch theoretisch zu garantieren.

Für die in diesem Kapitel erläuterten algorithmischen Konzepte verwenden wir hauptsächlich die Terminologie zum Parameter Baumweite. Viele Ansätze lassen sich auf andere Graphenparameter, z.B. Zweigweite, übertragen. In einigen Veröffentlichungen wird darauf hingewiesen das die vorgestellten algorithmischen Ideen auch auf weitere Parameter übertragbar sind. Insbesondere Seymour und Thomas und die Arbeitsgruppen um Hans Bodlaender nehmen diese Beobachtungen mit großer Sorgfalt vor. Veröffentlichungen zu praktischen Implementierungen stellen solche Verbindungen leider oft nicht her. Dies ist wohl dadurch zu erklären, dass die Theorie der parametrisierten Komplexität noch nicht als Standard dient oder schlicht noch nicht bekannt war. Beispielsweise erklären Eppstein, Löffler und Strash, dass zentrale Beobachtungen zu ihrer Forschung in [70] bereits zuvor gemacht, aber nicht Parameter Degeneriertheit in Verbindung gebracht wurden.

4.1 Vorverarbeitung

Wir erwarten, dass die Laufzeit unserer Algorithmen für fast alle Eingaben mit deren Größe ansteigt. Um Laufzeit zu sparen, möchten wir, sofern möglich, unsere Eingabeinstanz auf eine oder mehrere kleinere Instanzen des selben Problems abbilden. Dabei muss gewährleistet werden, dass sich das Resultat der Berechnung nicht ändert, oder wir schnell aus der Lösung zu *verkleinerten Instanzen* eine Lösung zur *ursprünglichen Instanz* ableiten können. Neben dem pragmatischen Nutzen, erhalten wir auch Einsichten, die uns dem theoretischen „Kern“ des Problems näher bringen. Die Bestimmung der „schweren“ Instanzen des Entscheidungsproblems ist von zentralem wissenschaftlichen Interesse; denn die $\mathcal{C}$-schwere des Problems, für eine Komplexitätsklasse C, wird an der Existenz jener „schweren“ Instanzen festgemacht. Kleine „schwere“ Instanzen sind eine Teilmenge jener, die nach der Vorverarbeitung übrig bleiben.

verkleinerte
ursprüngliche

Zunächst können wir die Berechnung der Baumweite direkt abbrechen, falls der Eingabegraph zu viele Kanten enthält:

Lemma 29 (z.B. [55, Lemma 2.3])**.** *Sei $G = (V, E)$ ein Graph. Falls die Baumweite von G höchstens k ist, dann gilt: $|E| \leq k|V| - \frac{1}{2}k(k+1)$.*

Um Baum und Pfadweite zu bestimmen ist es ausreichend alle Zusammenhangskomponenten einzeln zu betrachten.

Satz 25. *Für jeden Graphen $G = (V, E)$ gilt:*

- *Die Baumweite von G ist das Maximum der Baumweite seiner ZHKn.*
- *Die Pfadweite von G ist das Maximum der Pfadweite seiner ZHKn.*
- *Die Baumweite von G ist das Maximum der Baumweite seiner 2-zusammenhängenden Komponenten.*

Beweis.

- Da alle ZHKn C_h, mit $1 \leq h \leq z$, Teilgraphen von G sind haben wir $\mathbf{tw}(G) \geq \mathbf{tw}(C_h)\ \forall\ h$. Seien optimale Baumzerlegung $(\mathcal{X}_h, (I_h, F_h))$ der C_h der Weite w_h und geeignet benannter Knoten gegeben. Wir erhalten eine Baumzerlegung $(\biguplus_h \mathcal{X}_h, (I, F))$, indem wir $z-1$ neue Kanten $f_{h,h+1}$ hinzufügen, die je beliebige Knoten aus I_h und I_{h+1} verbinden. F ergibt sich zu $\bigcup_h F_h \cup \{ f_{h,h+1} \mid 1 \leq h \leq z-1 \}$ und wir setzen $I = \biguplus_h I_h$. Dann ist (I, F) ein Baum. Da alle $(\mathcal{X}_h, (I_h, F_h))$ optimale Baumzerlegungen sind ist $(\biguplus_h \mathcal{X}_h, (I, F))$ eine Baumzerlegung zu G der Weite $\max_h w_h$ und optimal.
- Analog, indem wir alle Pfade Ende an Ende verbinden durch $z-1$ Kanten verbinden.
- Wir können o.B.d.A annehmen dass G 2-zusammenhängende Komponenten besitzt. Wir benutzen die Baumstruktur aus dem Beweis zu Satz 3 um eine neue Baumzerlegung $(\mathcal{X}, T)$ für G zu konstruieren. Repräsentiere jede Kante zwischen zwei benachbarten Artikulationspunkten durch einen Beutel aus zwei Knoten und jede 2-zusammenhängende Komponente Z_i durch eine optimale Baumzerlegung für Z_i. Es ist leicht zu überprüfen, dass alle Knoten und Kanten überdeckt sind und für alle v die T_v zusammenhängende Teilbäume von T bilden. Wir erhalten eine Baumdekomposition für G. Da gilt $\forall\ i : \mathbf{tw}(Z_i) \leq tw(G) = Weite(\mathcal{X}, T) = \max_j \mathbf{tw}(Z_j)$ folgt die Aussage.

□

Ein analoges Resultat gilt für die Zweigweite [63].

Bemerkung 16. Da Bäume beliebig große Pfadweite haben können, ist die Pfadweite eines Graphen im Allgemeinen nicht gleich dem Maximum der Pfadweite seiner 2-zusammenhängenden Komponenten.

Wir betrachten eine Sammlung an Reduktionsregeln auf Graphen mit dem Ziel die Instanzgröße zu verringern. Wir müssen in der Lage sein, den Einfluss der Reduktionsregel auf den jeweiligen Graphenparameter exakt anzugeben. Um die Verkleinerung der Probleminstanz durchzuführen, benutzen wir auch hier Knotenseparatoren.

Definition 37. Sei S ein Separator des Graphen $G = (V, E)$ und seien (W_i) die ZHK des Graphen $G - S$. Wir verbinden im induzierten Graphen $G[W_i \cup S]$ die Teilmenge S zu einer Clique. Den so konstruierten Graphen notieren wir mit $G[W_i \cup S] + \text{clique}(S)$. Der Separator S heißt *sicher für Baumweite* , falls $\mathbf{tw}(G) = \max_i \mathbf{tw}(G[W_i \cup S] + \text{clique}(S))$. sicher für Baumweite

Die Intuition zu dieser Definition ist, solche Separatoren zu verwenden, deren Ausnutzung die Weite der Baumzerlegung nicht erhöht. Grundsätzlich verkleinert sich die Baumweite durch die Reduktionsregeln nicht:

Lemma 30 (Bodlaender und Koster [38, Lemma 5])**.** *Für jeden Graphen $G = (V, E)$ und jeden Separator $S \subseteq V$ gilt: $\boldsymbol{tw}(G) \leq \max_i \boldsymbol{tw}(G[W_i \cup S] + clique(S))$, mit (W_i) den ZHK des Graphen $G - S$.*

Eine Vorverarbeitung mit *sicheren* Separatoren ist exakt. Bei jeder Anwendung einer Reduktionsregel wird der betrachtete Graph an einem sicheren Separator zerlegt und durch zwei oder mehr kleinere Graphen repräsentiert. Die Lösung des Baumweiteproblems auf den kleineren Graphen ist äquivalent zur Lösung des Baumweiteproblems auf dem ursprünglichen Graphen. Die Prozedur wird wiederholt, bis in den kleineren Graphen keine sicheren Separatoren mehr gefunden werden. Manche der verbleibenden Instanzen sind trivial. Die Vorverarbeitung ist abgeschlossen. Für alle verbleibenden Graphen der Sammlung können zum Beispiel Branch-and-Bound- oder Approximationsalgorithmen zum Einsatz kommen.

Lemma 31 (Bodlaender und Koster [38, Theorem 17])**.** *Sei S eine minimal separierende Menge in G der Größe k, so dass $G - S$ mindestens k ZHKn besitzt, dann ist S sicher für Baumweite.*

Cook und Seymour definieren einen zu „sicherer Separator für Baumweite“ sehr ähnlichen Begriff „sichere Spaltung eines Knotens für Zweigweite“, um analoge Resultate auf partiellen Zweigzerlegungen zu zeigen.

4.1.1 Cliquen und beinahe-Cliquen

Zur Bestimmung der Baumweite kann ein Graph ohne weiteres an bekannten Cliquen zerlegt werden.

Lemma 32. *Sei S ein Separator in G der eine Clique in G induziert. S ist sicher für Baumweite.*

Beweis. Die Baumweite $\mathbf{tw}(G)$ ist mindestens die Baumweite eines beliebigen Teilgraphen von G. Mit Benennungen wie in Definition 37 und $S = clique(S)$ folgt $\mathbf{tw}(G) \geq \mathbf{tw}(G[W_i \cup S] + \text{clique}(S))$. Außerdem gilt $\mathbf{tw}(G) \leq \mathbf{tw}(G[W_i \cup S] + \text{clique}(S))$ nach Lemma 30. □

Wir möchten nun dieses Lemma auf Minoren mit erhaltenen Knotennamen (Definition 19) von G erweitern die Cliquen sind.

Lemma 33 (Bodlaender und Koster [38, Lemma 11, Corollary 12])**.** *Sei S ein Separator in $G = (V, E)$. Falls für jede ZHK W von $G - S$ gilt, $clique(S)$ ist Minor mit erhaltenen Knotennamen von $G - W$. Dann ist S sicher für Baumweite.*

Definition 38. Die Knotenteilmenge S heißt beinahe Clique, falls ein Knoten v in S existiert, sodass $S \backslash \{v\}$ eine Clique ist. v heißt der nicht-Clique Knoten von S.

Satz 26. *Sei S eine minimal separierende beinahe Clique in G. Dann ist S sicher für Baumweite.*

Beweis. Sei S minimale separierende beinahe Clique in G mit nicht-Clique Knoten v. Seien (W_i) die ZHK des Graphen $G - S$. Wir betrachten nun die Graphen $G[W_i \cup S]$:

Da S minimal separierend ist hat jeder Knoten aus S mindestens einen Nachbarn in W_i, so auch v. W_i ist zusammenhängend, also existieren Pfade von v zu Knoten

$w \in S \backslash \{v\}$ die abgesehen von v und w nur durch W_i verlaufen. Wir kontrahieren alle Knoten aus W_i in den Knoten v. Der So entstehende Graph S' ist Minor von $G[W_i \cup S]$. Jeder w-v-Pfad wird zu einer Kante $\{v, w\}$ in S' kontrahiert. S' ist eine Clique über der Knotenmenge S die Minor von $G[W_i \cup S]$ ist. Die Aussage folgt mit Lemma 33. □

4.1.2 Kleine Separatoren

Für ein festes kleines k können in polynomieller Laufzeit $n^{\mathcal{O}(1)}\binom{n}{k} \subseteq \mathcal{O}(n^{k+\mathcal{O}(1)})$ kleine Separatoren gesucht werden. Im folgenden werden minimal separierende Mengen mit aufsteigender Größe von 1 bis 4 beschrieben.

Korollar 14. *Jede minimal separierende Menge der Größe* 1 *ist sicher für Baumweite.*

Beweis 1. Die Menge aller 2-zusammenhängenden Komponenten von G besteht aus induzierten Teilgraphen die sich gegenseitig genau in den Artikulationspunkten (siehe Beispiel 2) von G überlappen. Zu jeder separierenden Mengen $\{s\}$ der Größe 1, ist s Artikulationspunkt. Die Zerlegung wie in Definition 37 ist sicher nach Satz 25. □

Beweis 2. Die Aussage folgt direkt aus Lemma 32. □

Korollar 15. *Jede minimal separierende Menge der Größe* 2 *ist sicher für Baumweite.*

Beweis. Ein minimal separierende Menge der Größe 2 ist entweder Clique oder beinahe Clique. Die Aussage folgt mit Lemma 32 oder Lemma 26. □

Satz 27 (Bodlaender und Koster [38, Corollary 12, Lemma 15, Theorem 16])**.** *Sei S eine separierende Menge der minimalen Größe* 3*. Falls $G - S$ mindestens zwei ZHK besitzt und alle ZHKn mindestens zwei Knoten enthalten, dann ist S sicher für Baumweite*[1]*.*

Der Beweis gliedert sich durch mehrere Teilbehauptungen: Da S Separator minimaler Größe ist, existieren in G keine Separatoren der Größe 1 oder 2. Durch diese Eigenschaft von G lässt sich zeigen, dass in jeder ZHK W von $G - S$ ein Kreis C existiert. Der Graph $G[W \cup S]$ kann so kontrahiert werden, dass sich C zu einem Kreis über den Knoten von S zusammenzieht. Ein Kreis der Größe 3 ist eine Clique und so folgt das Resultat mit Lemma 33.

Zuletzt haben wir folgenden Satz für minimal separierende Mengen S der Größe $|S| = 4$, von der zusätzlich verlangt wird, dass $G[S]$ einen Stern als Teilgraphen enthält.

Satz 28 (Bodlaender und Koster [38, Theorem 18])**.** *Sei $S = \{v, w, x, y\}$ ein Separator der Größe* 4 *in $G = (V, E)$, mit $\{\{v, w\}, \{v, x\}, \{v, y\}\} \subseteq E$. Sei v in keinem Separator der Größe* 2 *oder* 3 *enthalten. Falls jede Zusammenhangskomponente von $G - S$ mindestens zwei Knoten enthält, dann ist S sicher für Baumweite.*

[1] Beachte, dass die Prämisse des Satzes gegenüber den vorigen Korollaren verschärft ist. Während eine separierende Menge der minimalen Größe 3 auch minimal separierend ist, so gilt die Umkehrung im Allgemeinen nicht.

Die wesentliche Idee des Beweises ist, den Graphen $G - v$ zu betrachten und durch die Menge $\{w, x, y\}$ zu zerlegen. Für die ZHKn (X_i) ergeben sich analog zu Satz 27 Minoren der $G[X_i \cup \{w, x, y\}]$, die Cliquen sind. Wenn der Knoten v wieder hinzugefügt wird, ergibt sich ein Minor mit erhaltenen Knotennamen. Alle dafür notwendigen Kanten sind in der Prämisse des Satzes gefordert. Die Aussage folgt mit Lemma 33.

4.1.3 Algorithmus zur Vorverarbeitung für Baumweite

Der Algorithmus besteht genau aus der Hintereinanderausführung obiger Vereinfachungsschritte [38]. Als interner Zustand des Algorithmus muss eine Menge von Graphen $\mathcal{M}^{(j)}$ gehalten werden. Den aktuellen Vorverarbeitungsschritt bezeichnen wir mit j, beginnend bei 0. G ist der einzige in $\mathcal{M}^{(0)}$ enthaltene Graph. In jedem Schritt suchen wir sichere Separatoren für die einzelnen Graphen in $\mathcal{M}$. Separatoren der Größe null existieren genau dann wenn der Graph nicht zusammenhängend ist. Wir ersetzen also in $\mathcal{M}$ zunächst den Eingangsgraphen durch seine Zusammenhangskomponenten. Ein Tiefendurchlauf realisiert dies in Zeit $\mathcal{O}(n + m)$. Entsprechend Korollar 14 suchen wir anschließend alle Artikulationspunkte in den Graphen $G_i^{(1)} \in \mathcal{M}$. Wir erhalten diese indem wir alle 2-zusammenhängenden Komponenten von allen Graphen in $\mathcal{M}$ bestimmen [97, Lemma 5, Theorem 8]. Dies ist in linearer Zeit bezüglich der einzelnen Graphen in $\mathcal{M}$ möglich [97, Theorem 7]. Falls in einem $G_i^{(1)}$ eine Artikulationspunkte gefunden werden ersetzen wir diesen Graphen durch die 2-zusammenhängenden Komponenten. Wir fahren mit der Suche nach separierenden Mengen der minimalen Größe 3 fort und ersetzen Graphen in $\mathcal{M}$ entsprechend. Sobald wir keinen sicheren Separator mehr finden ist die Bearbeitung abgeschlossen. Wir lösen oder approximieren die verbleibenden Instanzen nun durch einen anderen Algorithmus zur Baumzerlegung. Schließlich verwenden wir die Ersetzungsstruktur über den Graphen von $\mathcal{M}$ um eine Baumzerlegung der Teillösungen zusammenzusetzen.

Eine Zusammenstellung weiterer Regeln zur Vereinfachung von Graphinstanzen zum Baumweiteproblem findet sich in [98].

4.2 Separatoren

Wir greifen die Definition aus Abschnitt 2.1.4 wieder auf und zeigen zuerst einige nützliche Lemmata. Minimale Separatoren lassen sich in folgender Weise charakterisieren:

Lemma 34. *Sei S ein a, b-Separator eines Graphen $G = (V, E)$. S ist minimaler Separator genau dann, wenn zwei verschiedene ZHK von $G - S$ existieren, sodass jeder Knoten aus S je einen Nachbar in beiden Komponenten hat.*

Beweis. Sei S minimaler a, b-Knotenseparator. S trenne die ZHK $G[A]$ und $G[B]$ mit $a \in A$ und $b \in B$. Zu jedem Knoten $s \in S$ existiert ein a, b-Pfad p via s, der durch keinen

weiteren Knoten t von S verläuft. Andernfalls können wir entweder s oder t und eventuelle Zwischenknoten aus S entfernen um einen kleineren a,b-Separator zu konstruieren. Im Widerspruch zur Minimalität von S. Also sind der Vorgänger und Nachfolger von s in p je Nachbarn aus $G[A]$ und $G[B]$.

Habe nun jeder Knoten in S je einen Nachbarn in den ZHK $G[A]$ und $G[B]$, mit $a \in A$ und $b \in B$. Angenommen S wäre nicht minimal, dann existiert ein Knoten $x \in S$, sodass $S \backslash x$ ein kleinerer Separator ist. Dann existiert jedoch ein a,b-Pfad via x im Widerspruch dazu, dass S a,b-Separator ist. □

Eine ähnliche Charakterisierung existiert für minimal separierende Mengen:

Lemma 35. *Sei S ein a,b-Separator eines zusammenhängenden Graphen $G = (V,E)$. Die Menge S ist minimal separierend genau dann, wenn jeder Knoten $s \in S$ einen Nachbarn in jeder Zusammenhangskomponente von G hat.*

Beweis. Sei S eine minimal separierende Menge. Der Graph $G - S$ habe die ZHK $G[A], G[B], G[C], \ldots$. Da S nach Definition keinen kleineren Separator enthält, ist S minimaler Separator für alle Knotenpaare des Graphen $G - S$ deren Komponenten aus unterschiedlichen ZHK gewählt werden. Mit vorigem Lemma folgt die Aussage.

Andersherum habe nun jeder Knoten in S Nachbarn in allen ZHK von $G - S$. Wenn wir für den Widerspruchsbeweis annehmen, dass ein $s \in S$ in mindestens einer ZHK keinen Nachbarn besitzt, so folgt analog zum obigen Beweis, dass wir einen minimalen Separator $S - s$ konstruieren können. Im Widerspruch dazu, dass S minimal separierende Menge ist. □

Satz 29 (Robertson und Seymour [20])**.** *Sei $G = (V,E)$ ein Graph der Baumweite höchstens k und $W \subseteq V$. Dann existiert ein Separator S der Größe höchstens $k+1$, sodass jede ZHK von $G - S$ höchstens $\frac{1}{2}|W|$ Knoten aus W enthält.*

Korollar 16. *In einem Graphen der Weite höchstens k existieren $\frac{1}{2}, k+1, W$-Separatoren.*

Wir betrachten die von Amir in [33] dargestellten Algorithmen für $\frac{2}{3}, k, W$-Separatoren, welche die Algorithmen von Robertson und Seymour [20] leicht verbessern.

Algorithmus 7 (FIND$\frac{2}{3}$BALANCEDPARTITION)**.**
Eingabe: Graph $G = (V,E)$, Knotenteilmenge $W \subseteq V$, natürliche Zahl k
Ausgabe: $\frac{2}{3}, k, W$-Separator S, separierte Knotenteilmengen A und B.

1: Rate eine Menge $W_1 \subset W$ mit $\left\lceil \frac{|W|}{2} \right\rceil$ Knoten und eine Menge $W_2 \subset W \backslash W_1$ mit $\left\lceil \frac{|W|}{3} \right\rceil$ Knoten.
2: Kopiere $G' \leftarrow G$. Vervollständige W_1 und W_2 zu Cliquen in G' und füge zwei neue Knoten w_1 und w_2 zu G' hinzu. In G' verbinde w_1 zu allen Knoten aus W_1 und w_2 zu allen Knoten aus W_2.
3: Finde einen minimalen (w_1, w_2)-Separator S.
4: **if** $|X| \leq k$ **then**
5: Bilde Knotenteilmengen A und B aus den ZHKn von $G - S$

6: **return** S, A, B
7: **else**
8: **Ablehnen**

Lemma 36 (Amir [33, Lemma 4.2])**.** *Seien $G = (V, E)$ ein einfacher Graph, $k \in \mathbb{N}$, und $W \subseteq V$ der Größe $3k + 2$. Der Algorithmus* FIND$\frac{2}{3}$BALANCEDPARTITION *findet einen $\frac{2}{3}, k, W$, falls dieser existiert und lehnt sonst ab. Der Algorithmus Arbeitet in Zeit $\mathcal{O}(\frac{2^{4,38k}}{k} t(|V|, |E| + k^2, k))$, falls ein Algorithmus gegeben ist, der einen minimalen (a, b)-Separator in Zeit $t(n, m, k)$ berechnet.*

Die (a, b)-Separatoren berechnen wir über maximale Flüsse. FIND$\frac{2}{3}$BALANCEDPARTITION ist ein Algorithmus wie verlangt in Voraussetzung 1. Falls FIND$\frac{2}{3}$BALANCEDPARTITION ablehnt, so ist die Baumweite von G mindestens $k + 1$.

Korollar 17 (Robertson und Seymour [28])**.** *Es existiert ein Algorithmus, der auf Eingabe $G = (V, E)$, $W \subseteq V$ und $k \in \mathbb{N}$ entweder die korrekte Ausgabe $\boldsymbol{tw}(G) > k$ produziert, oder einen $\frac{2}{3}, k + 1, W$-Separator berechnet. Der Algorithmus läuft in Zeit $\mathcal{O}(3^{|W|} \cdot k^{\mathcal{O}(1)} \cdot (n + m))$.*

4.2.1 Approximation kleiner balancierter Separatoren

Wir betrachten den von Brandt und Wattenhofer 2019 vorgestellten Las Vegas Algorithmus zur Approximation kleiner balancierter Knotenseparatoren gegeben [12]. Die Autoren zeigen für ein beliebiges $\frac{2}{3} \leq \alpha < 1$ und $0 < \varepsilon < 1 - \alpha$: Falls der Graph G einen minimalen α-Separator der Größe K enthält, so findet der Algorithmus einen $(\alpha + \varepsilon)$-Separator der Größe $\mathcal{O}(\varepsilon^{-1} K^2 \log^{1+o(1)} n)$. Der Algorithmus läuft in Zeit $\mathcal{O}(\varepsilon^{-1} K^3 m \log^{2+o(1)} n)$. Brandt und Wattenhofer argumentieren, dass balancierte Knotenseparatoren zur Zerlegung von Graphen zum Zwecke der Parallelisierung besser geeignet seien, als balancierte Kantenseparatoren. Insbesondere lassen sich Schranken für die Güte der Approximation nicht ohne Weiteres übertragen, wenn ein Knotenseparator aus einem approximierten Kantenseparator konstruiert wird. Für die Effizienz der Gesamtberechnung ist entscheidend, dass die Laufzeit zur Berechnung des Knotenschnitts sehr klein ist, da dieser Schritt nicht parallelisiert werden kann. Eine polynomielle Laufzeit sei bereits nicht mehr akzeptabel [12]. Der vorgestellte Algorithmus erzielt für Graphen mit minimalen Separatoren der Größe $\mathcal{O}(\text{ polylog } n)$ eine fast lineare Laufzeit, explizit in $\mathcal{O}(\varepsilon^{-1} m \text{ polylog } n)$. Da K im Allgemeinen nicht bekannt ist, wird dieser Parameter bei jeder fehlschlagenden Suche verdoppelt.

Feige, Hajiaghayi und Lee zeigen in ihrem Paper von 2005 eine polynomielle Laufzeit für die Suche eines $\frac{3}{4}$-Separator mit Größe $K \log^{\frac{1}{2}} K$, falls ein $\frac{2}{3}$-Separator der Größe K existiert [8]. In einer Veröffentlichung von 2008 präsentieren Feige, Hajiaghayi und Lee einen randomisierten Algorithmus, der einen α-Separator der Größe k findet, es sei denn, der Graph enthält einen kleineren $(\alpha + \varepsilon)$-Separator mit Größe strikt kleiner k. Dieser Algorithmus läuft in der Zeit $n^{\mathcal{O}(1)} 2^{\mathcal{O}(k)}$. Für $k \in \mathcal{O}(\log n)$ ergibt sich also eine polynomielle Laufzeit.

Zudem impliziert das Resultat von [8] folgende interessante Beobachtung. k sei die Größe eines minimalen α-Separators in G. Die einzige Möglichkeit eine **NP**-schwere Instanz für das MINIMUMα-SEPARATORPROBLEM zu konstruieren, ist einen solchen minimalen α-Separator unter anderen minimalen Separatoren des selben Graphen$G$ zu „verbergen“, die zwar kleiner als k, aber weniger balanciert $\alpha + \epsilon$ sind.

4.2.2 Balancierte Knotenseparatorzahl

Ähnlich zur Cliquenzahl, welche die Kardinalität einer größten Clique in einem Graphen G angibt, definieren wir die balancierte Knotenseparatorzahl.

Definition 39. Die balancierte Knotenseparatorzahl $\mathbf{s}(G)$ eines Graphen G ist definiert als die kleinste Zahl k sodass jeder induzierte Teilgraph von G einen k-Separator enthält.

Satz 30 (Gruber [7, Lemma 4, Theorem 6, Corrolary 8])**.** *Für jeden Graph G mit balancierter Knotenseparatorzahl $\boldsymbol{s}(G)$ und n Knoten gelten $\boldsymbol{s}(G) \leq \boldsymbol{tw}(G)$ und $\boldsymbol{r}(G) \leq \boldsymbol{s}(G) \cdot (1 + \log \frac{n}{\boldsymbol{s}(G)})$. Diese Schranken sind bestmöglich.*

Korollar 18. *Sei G ein ungerichteter Graph mit n Knoten und $n \geq 2$: Es gilt:*

$$\boldsymbol{s}(G) \leq \boldsymbol{tw}(G) \leq \boldsymbol{pw}(G) \leq \boldsymbol{r}(\overleftrightarrow{G}) \leq \boldsymbol{s}(G) \cdot (1 + \log \frac{n}{\boldsymbol{s}(G)})$$

4.3 Potentiell maximale Cliquen

Unser Interesse an Cliquen in chordalen Graphen ist durch folgenden Satz von Dirac begründet:

Satz 31 (Dirac [6], [31])**.** *Ein Graph H ist trianguliert genau, dann wenn jeder minimale Separator eine Clique in H ist.*

Die folgenden Lemma geben Charakterisierungen minimaler Chordalisierung:

Lemma 37 (Rose, Tarjan und Lueker [99])**.** *Sei $H = (W, F)$ eine Chordalisierung des Graphen $G = (V, E)$. H ist eine minimale Chordalisierung von G, gdw. für alle Kanten $e \in F \backslash E$ der Graph $H - e$ nicht chordal ist.*

potentiell maximale Clique

Eine *potentiell maximale Clique* eines Graphen G ist eine Knotenmenge, die eine maximale Clique in einer minimalen Triangulation von G induziert. [46]. Minimale Separatoren gruppieren sich im Graphen G dort, wo eine potentiell maximale Clique nachgewiesen werden können. [46]. Wir können also potentielle maximale Cliquen in exakten Algorithmen für Baumweite verwenden, um die Suche nach geeigneten minimalen Separatoren für den Zerlegungsschritt einzuschränken. Für Graphenfamilien, deren potentielle Cliquen in polynomieller Zeit aufgezählt werden können, kann die Baumweite in polynomieller Zeit berechnet werden. [46]

4.4 „Branch and Bound“ Algorithmen

Branch-and-Bound ist ein Meta-Verfahren zur Lösung schwieriger Optimierungs- Probleme. Insbesondere solcher, deren zugehöriges Entscheidungsproblem **NP**-schwer ist. In diesem Kapitel wird dargestellt, wie Branch-and-Bound zur Bestimmung der exakten Baumweite eines Graphen genutzt wird.

Ein Branch-and-Bound Algorithmus verläuft in der Regel in zwei Phasen. In der ersten Phase wird ein beliebiger, nicht notwendigerweise optimaler Lösungskandidat des Problems gesucht. Dieser kann je nach Problem sehr schnell konstruiert, durch Tiefensuche im Suchraum gefunden werden oder als Ergebnis eines vorherigen Approximationsverfahrens vorliegen. Ein Maß über den Lösungskandidaten dient zur Bestimmung einer oberen Schranke für die Güte aller später gefundenen Lösungen. Diese obere Schranke wird mit dem Maß des zuerst konstruierten Lösungskandidaten oder $+\infty$ global initialisiert.

In der zweiten Phase wird der Suchraum erneut betrachtet. Der „Branch“-Schritt verkleinert den Suchraum, indem dieser in zwei oder mehr bestenfalls disjunkte Suchräume geteilt wird. Für jeden gefundenen Teilsuchraum wird nacheinander der „Bound“-Schritt ausgeführt: Für alle Instanzen des Suchraums wird eine untere Schranke des Gütemaßes bestimmt. Falls die berechnete untere Schranke die gespeicherte obere Schranke übersteigt, kann der ganze Teilsuchraum verworfen werden, da bereits ein besserer Lösungskandidat bekannt ist, als jede Instanz in diesem Teilsuchraum liefern würde. Andernfalls wird der Teilsuchraum weiter geteilt. Wird im Verlauf des Verfahrens ein besserer Lösungskandidat gefunden, so wird die globale obere Schranke aktualisiert.

Das Verfahren kann als Baum visualisiert werden, der ausgehend vom Wurzelknoten, welcher den gesamten Suchraum repräsentiert, im Branch-Schritt durch Kindknoten erweitert wird. Der Bound-Schritt bewirkt, dass nicht oder teil-explorierte Knoten weggeschnitten und nicht weiter betrachtet werden.

Die Laufzeit dieses Verfahrens hängt stark von der Heuristik zur Bestimmung einer unteren Schranke für das betrachtete Maßes ab. Wenn im Worst-Case der Suchraum nicht durch gute untere Schranken beschnitten werden kann, degeneriert das Verfahren zur vollständigen Enumeration des Suchraums, was in der Regel eine exponentielle Laufzeit impliziert.

Gogate und Dechter stellen in ihrem Paper von 2012 einen Branch-and-Bound Algorithmus zur Berechnung der Baumweite vor [48]. Der Suchraum wird durch perfekte Eliminationsschemata modelliert, als Maß dient die Baumweite der über die Eliminationsschemata konstruierten chordalen Vervollständigungen. Zur Bestimmung einer unteren Schranke auf der Baumweite verwenden die Autoren ihre eigene Heuristik „minor-min-width“.

5 Diskussion und Ausblick

Wir haben verschiedene Graphenparameter kennen gelernt, die sich zum Teil sehr unterschiedlich verhalten. Es scheint gängige Praxis eine numerische Gracheninvariante erst dann als Graphenparameter zu bezeichnen, wenn diese als Parameter in einem Algorithmus im Sinne der parametrisierten Komplexität verwendet wird. Die Anzahl bekannter Graphenparameter ist überschaubar; Vatshelle identifizierte 2012 in seiner Doktorarbeit zweiunddreißig verschiedene Graphenparameter [44], während wir wesentlich mehr Graphinvarianten zählen.

Die Degeneriertheit, maximaler Grad, Radius, Diameter eines Graphen sowie viele weitere numerische Graphinvarianten können in polynomieller Zeit bestimmt werden. Sie werden in Graphenalgorithmen verwendet. Eine parametrisierte Komplexitätsanalyse wird nicht immer durchgeführt, weshalb gegenüber „worst-case“ optimalen Algorithmen bezüglich der klassischen Komplexität manchmal Verbesserungen im Sinne der parametrisierten Komplexität möglich sind, die zu kleineren praktischen Laufzeiten führen (siehe [70]). Für Graphenalgorithmen wird oft die Laufzeit bezüglich $n = |V|$ und $m = |E|$, statt über die Länge der Kodierung N, angegeben. Es scheint nützlich diese Konvention auch bezüglich weiterer Graphenparameter zu verwenden, auch wenn dies nicht für die Mitgliedschaft in klassischen Komplexitätsklassen insb. **P** relevant ist.

Ein Graphen können in polynomieller Zeit auf Planarität getestet werden [1]. Aufgrund der praktischen Relevanz existieren sehr viele Graphenalgorithmen für die Familie der planaren Graphen die wesentlich bessere Laufzeiten als Algorithmen für allgemeine Graphen aufweisen. Eine mögliche wenn auch nicht erschöpfende Erläuterung dieses Phänomens ist die Feststellung, dass planare Graphen 5-degeneriert und damit dünn-besetzt sind. Das bedeutet in Graphenalgorithmen müssen wir wesentlich weniger Kanten betrachten, als dies für allgemeine Graphen der Fall ist.

Die Einführung der Parameter Baumweite und Zweigweite durch Robertson und Seymour führte uns zu den Graphenfamilien beschränkter Baum- und Zweigweite die ähnlich wie planare Graphen gute algorithmische Eigenschaften aufweisen und als Verallgemeinerung der Bäume, Serien-Parallelen-Graphen und Sterngraphen aufgefasst werden können. Zum Teil sind diese Verallgemeinerungen bekannte graphentheoretische Objekte wie k-Bäume. Durch ihre Mitgliedschaft in **FPT** sind die zugehörigen Entscheidungsprobleme mit vertretbarem Aufwand algorithmisch lösbar, oder zumindest praktisch approximierbar. Genau wie planare Graphen können

[1] beispielsweise mit Satz von Kuratowski und Satz von Wagner-Robertson-Seymour durch Minorentest auf $K_{3,3}$ und K_5 oder den Planaritätstest von Schnyder

Graphen beschränkter Baum- und Zweigweite können ebenfalls durch das Verbot bestimmter endlicher Mengen von Minoren charakterisiert werden. Insbesondere die Zweigweite erweist sich als mächtiges algorithmisches Werkzeug für Algorithmen auf Graphen welche auch planar sind, da PLANARBRANCHWIDTH $\in$ **P**. Zum Beispiel für das bekannte Problem TSP [63]. Aber auch Baumweite kann für fast-planare Graphinstanzen, die sich an der Topologie der Erde orientieren, wie Energie-, Schienen- und Straßennetze, ein geeigneter Parameter sein. Andererseits sind die planaren Graphen weder in der Familie beschränkter Baum- noch Zweigweite enthalten, und wir müssen hohe Parameterwerte für manche Graphen erwarten. Die Zerlegungen zu beiden Algorithmen erlauben dynamische Programmierung als generischen Ansatz zur Konzeption von Graphenalgorithmen. Für manche Entscheidungsprobleme kann ein Baumalgorithmus, welcher das Problem eingeschränkt auf Bäume löst, relativ direkt verallgemeinert werden. Wir haben dies am Beispiel WEIGHTEDINDEPENDENTSET gesehen. Für den zugehörigen Baumalgorithmus siehe [5, Kapitel 14]. Andere graphentheoretische Begriffe (wie z.B. Cliquen) haben schlicht auf Bäumen keine nicht-triviale Bedeutung, weshalb für zugehörige Graphenprobleme völlig neue algorithmische Ideen gesucht werden. Die Parameter Baumweite und Zweigweite dominieren den Parameter Degeneriertheit. Wir haben festgestellt, das daher für Graphen beschränkter Baum- und Zweigweite c auch alle Algorithmen verwendet werden können, die als Parameter beschränkte Degeneriertheit verlangen. Verwende c als Schranke.

Kloks stellte fest, dass in jeder Graphenfamilie $\mathcal{G}$, die abgeschlossen bezüglich der Bildung von Minoren ist, fast jeder Graph mit n Knoten und $\geq 1,18\,n$ Kanten nicht in $\mathcal{G}$ enthalten ist (Satz 12). Dieses Argument trifft sowohl auf planare Graphen als auch auf Graphen beschränkter Baum- und Zweigweite zu. Die Parameter Baum- und Zweigweite nehmen also nur auf Graphen aus Familien dünn-besetzter Graphen durchgehend geringe Werte an. Zwar mag dies einerseits ein Indiz gegen die weitreichende Anwendbarkeit dieser Parameter sein, andererseits haben wir in vielen Anwendungsfeldern die Eigenschaft, dass nur Instanzen einer oft nicht näher bestimmten Familie von Graphen betrachtet werden. Es ist daher angemessen zu überprüfen, ob die für das betrachtete Graphenproblem „schweren“ Graphinstanzen in der Praxis überhaupt vorkommen, oder ob die „worst-case“ Laufzeit zu pessimistisch angesetzt ist. Zwei Wege diese Frage zu beantworten sind die Familie der Eingabegraphen anhand der Anwendung genauer zu charakterisieren oder algorithmische Experimente durchzuführen, um schwere Graphinstanzen kategorisch oder wenigstens empirisch auszuschließen. Ersterer Ansatz erfordert eine gute Kenntnis von Graphenparametern und parametrisierter Analyse unter Informatikern, die praktisch mit Graphenalgorithmen arbeiten. Für letzteren Ansatz sind große Datensätze mit im besten Fall bekannten Werten für gängige Graphenparameter nützlich, um die tatsächlichen Laufzeiten bekannter Algorithmen zu vergleichen. Die Initiative „Parameterized Algorithms and Computational Experiments“ *PACE* entwickelt solche Datensätze für Entscheidungsprobleme, die mit großen Interesse

im Forschungsfeld der parametrisierten Komplexität untersucht werden. Außerdem werden regelmäßig Implementierungs-Herausforderungen gestellt; mit dem Ziel schnelle Algorithmen auf diesen Datensätzen zu finden. Die praktischen Algorithmen finden wiederum Einzug in sehr spezialisierte Anwendungen und fördern auch die theoretische Forschung. Zur Zeit existieren noch keine großen Softwarepakete mit parametrisierten Graphenalgorithmen, die fachfremden Entwicklern leichten Zugriff auf die entwickelten Methoden geben. Oft auftretende hohe Konstanten in der Abschätzung von Laufzeiten mittels $\mathcal{O}$-Notation verlangen ebenfalls nach mehr experimentellen Studien zu Implementierung von Algorithmen aus dem Forschungsgebiet der theoretischen Informatik.

Algorithmen zu Graphenparametern profitieren von einer ausführlichen Vor- und Nachbearbeitung, um Teile der Eingabeinstanz zu finden die sich (auch praktisch) als „Kern“ des Problems herausstellen. Auch wenn sich die „worst-case“ Komplexität durch die Vorverarbeitung nicht verbessern sollte. Manchmal sind bereits die Reduktionsregeln der Vorverarbeitung ausreichend um eine optimale Zerlegungen bezüglich des Parameters zu finden [38].

Für dichte Graphen ist Baumweite oft kein geeigneter Parameter, da dicht-besetzte Graphen verhältnismäßig große Cliquen als Minoren enthalten, welche die Baumweite asymptotisch schnell steigen lassen. Der Parameter Cliquenweite wurde mit Blick auf dieses Problem vorgeschlagen und weist auch kleine Parameterwerte für Familien dicht-besetzter Graphen auf. Leider können optimale Zerlegungen zum Parameter **cw** von Graphen der Cliquenweite ≥ 4 bisher nur mit hohem Zeitaufwand konstruiert werden. Das Approximationsverfahren von Oum und Seymour, liefert ebenfalls nur Laufzeit- und Gütegarantien, die für den Parameter Baumweite als nicht ausreichend für die Praxis gelten würden. Entsprechend können wir Algorithmen mit einem **cw**-Ausdruck als Parameter ungeachtet der Komplexitätsklasse des entschiedenen Problems nicht praktisch ausführen, da aktuell zu einem Graphen G der Cliquenweite höchstens k kein **cw**,k-Ausdruck effizient berechnet werden kann. Neugierige Leser können [76] zum aktuellen Stand der Cliquenweite-Forschung heranziehen.

Da die Forschung aber stark an Graphenparametern interessiert ist, die für dicht-besetzte Graphen geringe Werte annehmen, wurden viele zu Cliquenweite verwandte oder äquivalente Parameter vorgeschlagen, die im Umfang dieser Arbeit nicht abgedeckt werden konnten. Einer dieser Parameter ist die modulare Weite. Dieser wurde exemplarisch erläutert. Modulare Zerlegung sind schon lange bekannt. Die modulare Weite wurde aber erst relativ kürzlich als Parameter herangezogen. Die modulare Weite eines Graphen kann in linearer Zeit bestimmt werden. Wir erhalten außerdem für jeden Graphen eine eindeutige Zerlegung mit einer Struktur, die sehr gut algebraisch untersucht ist.

Es können Beziehungen zwischen verschiedenen Graphenparametern hergestellt, die angeben wie „allgemein“ der Parameter verglichen mit anderen Parametern ist. Eine sehr lesenswerte Vertiefung dieses Ansatzes gibt Vatshellein seiner Doktorarbeit [44, Chapter 5]. Algorithmen parametrisiert durch einen spezielleren Parameter, wie

etwa der Baumweite gegenüber Degeneriertheit bzw. Cliquenweite, können genutzt werden um dass selbe Problem über einer Familie begrenzter Degeneriertheit bzw. Cliquenweite zu lösen. Mit anderen Worten für Graphen aus Familien beschränkter Degeneriertheit bzw. Cliquenweite ist das Entscheidungsproblem bereits durch einen Algorithmus mit Parameter Baumweite theoretisch gelöst. Praktisch nützlich ist diese Strategie im Allgemeinen nicht, da sich die Größe des Parameterwertes in beliebiger Größenordnung ändern kann. Jedoch gewinnen wir die Intuition, dass durch den Parameter Baumweite nicht weniger Entscheidungsprobleme parametrisiert werden können als mindestens durch den spezielleren Parameter. Beispielsweise wurde festgestellt, dass sich MAXCUT GRAPHCOLORING, HAMILTONIANCYCLE und EDGEDOMINATINGSET parametrisiert durch Baumweite in **FPT** lösen lassen, nicht jedoch für den Parameter Cliquenweite in **FPT** gelöst werden können, außer es gilt **FPT** = **W**[1] [100], [101].

Keine Arbeit, welche die Baumweite erwähnt ist vollständig ohne Bezug auf den Satz von Courcelle, welcher besagt, dass jede Grapheigenschaft, die in einer Variante der monadischen Prädikatenlogik erster Stufe, genannt MSO_1, ausgedrückt werden kann, auf Graphen beschränkter Baumweite in linearer Zeit entschieden wird. Varianten dieses Satzes, und damit die **FPT**-Zugehörigkeit vieler Graphenprobleme, wurden für einige gängige Graphenparameter gezeigt wie **cw**, **tw**, **bw**, **pw** und **td** [44]. Hingegen existieren für andere Parameter **p**, wie beispielsweise die Degeneriertheit, Entscheidungsprobleme mit Parameter **p** die sich nicht in MSO_1 definieren lassen [44]. Eine Ausweitung der Grapheigenschaften auf solche die in MSO_2 definierbar sind, führt dazu dass ein analoger Satz für **cw** gilt, die spezielleren Parameter **tw**, **bw**, **pw** und **td** jedoch weiterhin ausreichend sind um die Eigenschaft zu entscheiden [44].

Die Beobachtung dass Baumweite für manche Entscheidungsproblem als „zu speziell“ und Cliquenweite „zu allgemein“ wäre, eröffnet die Suche nach Parametern „zwischen“ oder „abseits von“ Baum und Zweigweite. Eine solcher Vorstoß gelang Sæther und Telle 2016 mit einem neuen Parameter **sm**, der wiederum auf die Arbeit von Vatshelle zu einem Parameter zurückgeht, der über maximale Matchings definiert ist. Es ist wahrscheinlich, dass weitere ähnliche Parameter vorgeschlagen werden, für welche die Frage nach exakten Algorithmen, Approximierbarkeit und Güte der Approximierungen beantwortet werden müssen, um praktische Implementierungen zu ermöglichen.

Einige Graphenparameter lassen sich auf Hypergraphen definieren oder wurden bereits ursprünglich auf Hypergraphen definiert. Eine Forschungsgruppe an der Technischen Universität Wien untersuchte Verallgemeinerungen von Graphenparametern auf Hypergraphen (siehe z.B. [103]) welche natürliche Anwendungen in verschiedenen **NP**-vollständigen Problemen haben, die in der praktischen Informatik von Interesse sind, wie Constraint-Satisfaction-Probleme und Query-Probleme der Datenbanktheorie [104], [105]. In ihrer Veröffentlichung von 2018 verlassen Jeong, Kim und Oum die Graphentheorie vollständig und Verallgemeinern die Bestimmung

verschiedener Graphenparameter und Zerlegungen zu einem Schnittproblem über gewissen Teilräumen eines endlich-dimensionalen Vektorraums über wiederum verschiedenen endlichen Körpern $\mathbb{F}$. Sie zeigen, dass in ihrem Modell und einigen Spezialisierungen die Probleme BRANCHWIDTH, RANKWIDTH, und CARVINGWIDTH in der asymptotisch selben Laufzeit gelöst werden können wie der Laufzeit kompetitiver Algorithmen. Diese Fortschritte werfen die Frage auf, ob alle „Graphen"-Parameter gleichermaßen gut in der Graphentheorie angesiedelt sind, oder eine allgemeinere Modellierung auf einer anderen Algebra passender wäre.

Ein Ansatz Graphenzerlegungen sehr allgemein zu untersuchen ist, sogenannte H-Dekompositionen zu betrachten. Das zugehörige Entscheidungsproblem fragt, ob ein Graph als eine Kanten-disjunkte Vereinigung von Graphen dargestellt werden kann, welche zu H isomorph sind. Das Forschungsfeld hat eine lange Tradition in der Graphentheorie und es stellen sich viele offene Fragen zu Komplexität dieses und ähnlicher Probleme Cohen und Tarsi [107]. Die parametrisierte Analyse dieser Probleme könnte Gegenstand neuer Forschung sein.

Für planare Graphen können Zerlegungen entwickelt werden, die sich stärker an der Topologie einer Einbettung orientieren. Als Beispiel dienen die Veröffentlichung von Dorn, Penninkx, Bodlaender u. a. [108] und Anhang A.1. Die Kombinationsmöglichkeiten von Graphenfamilien eines beschränkten Graphenparameters, für die zusätzliche Grapheigenschaften gelten, wie Planarität, Perfektion, usw., sind sehr zahlreich. Informiert durch Fragestellungen der praktischen Informatik, werden in diesen Forschungsfeldern Algorithmen, Optimierungen und Zerlegungen entwickelt, die möglicherweise andere Begriffe verwenden und daher erst mit der Zeit zusammengetragen werden. Angenommen die Komplexität des jeweiligen Problems könnte durch den Satz von Courcelle beantwortet werden; so sind die entwickleten algorithmischen Methoden dennoch unverzichtbar, um Implementierungen mit geringer praktischer Laufzeit zu finden.

Die theoretischen Resultate legt in vielen Forschungsfeldern der parametrisierten Komplexität Maßstäbe für die Laufzeit, die praktische Implementierungen nur selten erreichen. Erklärungen hierfür können einerseits in möglichen Schwächen der $\mathcal{O}$-Notation liegen, andererseits ist es führ das theoretische Forschungsprogramm nicht notwendig, sich auf ausführbare Algorithmen zu beschränken. Dies bewirkt dass viele der betrachteten Algorithmen potentiell nie vollständig entwickelt werden. Aus praktischen Interessen und der Beobachtung das Implementierungen neue Forschungsansätze und Korrekturen vorheriger Veröffentlichungen beitragen [109], werden zunehmend experimentelle Studien konkreter Implementierungen durchgeführt und gegeneinander verglichen [33], [34], [38], [70], [110]–[112]. Implementierungen können ebenfalls genutzt werden um Konstanten im Algorithmus empirisch zu optimieren [63]. Für das Forschungsfeld der parametrisierten Komplexität wurden zu einigen der hier dargestellten Graphenparameter Wettbewerbe im Format PACE ausgetragen [92], [94], die nun regelmäßig fortgesetzt werden.

A Verwandte Graphzerlegungen

Im Umfang einer Masterarbeit können nicht alle aufkommenden Themen auch nur ansatzweise vollständig dargestellt werden. Dieser Anhang beschreibt kurz zwei weitere Zerlegungsmethoden für Graphen abseits des „roten Fadens“ dieser Arbeit, die dennoch erwähnt werden mögen.

A.1 2- und 3-zusammenhängende Komponenten und SPQR-Bäume

Hopcroft & Tarjan's Algorithmus für 2-zusammenhängende Komponenten [113] partitioniert einen Graph in zusammenhängende Komponenten, zwei-zusammenhängende Komponenten und einfache Pfade in Platz und Zeit $\mathcal{O}(\max|V|,|E|)$. Details können in [113] nachgelesen werden. Ein ähnlicher Algorithmus verfeinert das Resultat indem in den 2-zusammenhängenden auch 3-zusammenhängende Komponenten identifiziert werden. Der Algorithmus zur Bestimmung drei-zusammenhängender Komponenten eines Graphen geht ebenfalls auf Hopcroft und Tarjan zurück und wird ausführlich in [114] beschrieben. Über den so gefundenen Teilgraphen können wir SPQR-Bäume definieren, die beschrieben wie der ursprüngliche Graph aus den 3-zusammenhängenden Komponenten zusammengebaut werden kann. Baumzerlegungen können als Verallgemeinerung der SPQR-Zerlegungen aufgefasst werden.

SPQR-Bäume haben viele Anwendungen im Zusammenhang mit planaren und fast planaren Graphen. Die Knoten eines SPQR-Baums sind die $\leq$ 3-zusammenhängenden Komponenten eines ungerichteten, 2-zusammenhängenden Multigraphen G. Die $\leq$ 3-zusammenhängenden Komponenten überlappen sich in sogenannten separierenden Paaren von G. Separierende Paare bestehen je aus zwei Knoten v und w. In unserer bisherigen Terminologie ist $\{v, w\}$ ein Knotenseparator der Größe 2 und außerdem minimal separierende Menge, da 2-zusammenhängende Graphen nach Definition keine Artikulationspunkte enthalten. Die Kante $\{v, w\}$, welche nicht notwendigerweise zu G gehört und deshalb als „virtuell“ bezweichnet wird, separiert zwei benachbarte 3-zusammenhängende Komponenten im eindeutigen SPQR-Baum von G. Wir erinnern daran, dass ein Separator der Größe 2 den Graphen in maximal zwei ZHK trennt, also hier genau zwei Komponenten trennt.

Gutwenger und Mutzel entdeckten 2001 einige Fehler in Hopcroft und Tarjan's Algorithmus, korrigierten diese und zeigten wie durch kleine Modifikationen die in theoretischen Arbeiten oft vorausgesetzte SPQR-Zerlegung tatsächlich berechnet werden kann. Für genauere Informationen und Korrektheitsbeweise siehe [109], [114].

A.2 Cliqueüberdeckung CLIQUECOVER

Eines von Karp's 21 **NP**-vollständigen Problemen ist ebenfalls mit einer, besonderen Partition der Knoten eines ungerichteten Graphen $G = (V, E)$ befasst. Eine Cliqueüberdeckung ist eine Partition der Knotenmenge V durch Knotenteilmengen $C_i \subseteq V$, die Cliquen in G sind. Das zugehörige Entscheidungsproblem fragt, ob eine Cliquenüberdeckung mit k Cliquen hat. Das minimale $k*$ für welche eine Cliquenüberdeckung möglich ist heißt Cliquenüberdeckungszahl.

Eine Cliquenüberdeckung kann aus einer optimalen Färbung des Komplementgraphen von G konstruiert werden. Die Farbklassen im Komplementgraphen $\bar{G}$ bilden je eine Clique in der zugehörigen Cliqueüberdeckung von G.

Im Unterschied zu den hier betrachteten Zerlegungen sind in einer Knotenüberdeckung zu einem nicht vollständigen Graphen G nicht alle Kanten von G durch Clique überdeckt. Das heißt es gibt Kanten in G deren Endpunkte in unterschiedlichen Knoten einer Überdeckung liegen.

Espelage, Gurski und Wanke bewiesen 2001, dass eine optimale Knotenüberdeckung von G für Graphen mit beschränkter Cliquen-Weite in polynomieller Zeit gefunden werden [17] kann. CLIQUECOVER ist damit **FPT** für den Parameter Cliquenweite. Neben Cliqueüberdeckungen zeigen die Autoren ebenfalls die Mitgliedschaft in **FPT** der Entscheidungsprobleme zu Überdeckungen durch Dreiecke[1], vollständig bipartiter Graphen, Hamiltonscher Pfade, sowie einiger weiterer Überdeckungen, die ähnlichem Sinne wie oben eine Überdeckung des Graphen G durch Einbettung gleichartiger Graphen vornehmen [17].

[1] C_3 bzw. K_3 heißt Dreiecksgraph

B Ausgabe, Export, Layout

B.1 TikZ

Die zur Zeit der Bearbeitung aktuelle TikZ Version ist 3.1.5b.

In diesem Abschnitt wird kurz die erstellung von Graphendarstellungen mit TikZ erklärt.

B.1.1 Stile

Mit dem Befehl `\tikzstyle{every node}` kann das Aussehen aller Knoten in nachfolgenden {tikzpicture}-Umgebungen verändert werden.

Beispiel 16. `\tikzstyle{every node}=[circle, draw, fill=black!50, inner sep=0pt, minimum width=4pt]`

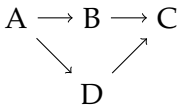
Die obige Stildefinition besagt, dass jeder als ausgefüllter grauer Kreis mit größe mitdestens 4pt dargestellt wird.

`\tikzstyle{every graph}` setzt Stildefinitionen die vor Auswertung der Optionen und der Ausführung jedes Befehls `\graph` angewendet werden.

B.1.2 Die TikZ Bibliothek „graphs“

Um die exportierten Graphendarstellungen in einer `.tex`-Datei zu benutzen muss in der Präambel des Dokuments `\usepackage{tikz}` angegeben werden. Die Bibliothek `graphs` wird verwendet um einfache Knoten-Kanten-Diagramme zu erzeugen. In einer LateX Umgebung kann sie mit dem Befehl `\usetikzlibrary{graphs}` geladen werden.

Beispiel 17.




```
\tikz \graph { A -> {B, D} -> C };
```

`graph` setzt rekursiv den Gesamtgraph aus kleineren Graphen zusammen. Die Graphspezifikation wird in low-level Befehle wie `node` oder `edge` übersetzt. Knoten sind normale TikZ nodes und Kanten normale TikZ lines. Das heißt Stildefinitionen sind auch für Darstellungen gültig, die mit dem Befehl `graph` erzeugt wurden.

Die Graphspezifikation erfolgt in einer Sprache ähnlich zu `dot` von GraphViz und ist typischerweise sehr kurz im Vergleich dazu, Knoten und Kanten einzeln zu definieren.

Die Bibliothek `graphs.standard` enthält Makros für einige sehr häufig benötigte Graphen.

Knoten können konzeptuell „gefärbt“ werden um an späterer Stelle die gefärbte Knotenmenge zu referenzieren. Verwende die Option `[nodes=red]` um allen Knoten der folgenden Gruppe rot zu färben. Färbungen werden nicht gezeichnet. Sie diehnen nur der Benennung der Knotenmenge.

Kanten können mit der Option `[edge label=e]` mit dem Label „e“ versehen werden. `edge label'` erzeugt ein Label auf der gegenüberliegenden Seite.

Optionen können sowohl auf Gruppen als auch auf einzelnen Knoten gesetzt werden.

Um Kanten zu spezifizieren wird eine der folgenden Zeichenfolgen in Infix-Notation verwendet: `->`, `-`, `<-`, `<->`, `-!-`. Letztere kann genutzt werden um eine exisiterende Kante zu entfernen. Kantenspezifikationen erlauben Optionen. So kann beispielsweise ein Verbindungsalgorithmus für die Knotengruppen links und rechts des Infix-Operators gewählt werden. `[complete bipartite]` erzeugt etwa einen bipatiten Teilgraphen. Die verschiedenen Infix-Operatoren führen zu folgender Darstellung

a ⟶ b
c — d
e ⟵ f
g ⟷ h
i j

Um Knoten zu spezifizieren werden entweder Knotengruppen angegen, neue Knoten durch eine Zeichenkette definiert oder existierende Knoten durch Klammerung `( )` referenziert. Neue Knoten starten mit Namen; weiterer Text kann nach einem Schrägstrich `/` folgen. Die Option `[fresh nodes]` bewirkt, dass alle Knoten in der Gruppenspezifikation neu erzeugt werden auch wenn existierende Knoten den selben Namen führen.

Die Option `[math nodes, nodes={circle, draw}]` erzeugt umrandete Knoten deren Namen als Knoten-Label in der Umgebung `{math}` gesetzt werden. Die Option `as` überschreibt jeglichen Text, der durch von PGF/TikZ für einen Knoten ermittelt wurde. E.g. `a [as=$x$]` wird gezeichnet als x

Literatur

[1] M. C. Golumbic, *Algorithmic graph theory and perfect graphs, Acad*. New York: Elsevier, 1980, S. 306. DOI: https://doi.org/10.1016/C2013-0-10739-8.

[2] N. Christofides, *Graph Theory: An Algorithmic Approach*, 4. 1976, Bd. 27, S. 1027. DOI: 10.2307/3009122.

[3] R. G. Downey und M. R. Fellows, *Parameterized Complexity*, Ser. Monographs in Computer Science. New York, NY: Springer New York, 1999, S. 398, ISBN: 978-1-4612-6798-0. DOI: 10.1007/978-1-4612-0515-9.

[4] R. Diestel, *Graphentheorie*. Springer-Verlag Heidelberg 1996, 2000, 2000, ISBN: 3540676562.

[5] S. Krumke und H. Noltemeier, *Graphentheoretische Konzepte und Algorithmen*. 2012, ISBN: 3834822647.

[6] G. A. Dirac, „On rigid circuit graphs“, *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, Jg. 25, Nr. 1, S. 71–76, 1961, ISSN: 18658784. DOI: 10.1007/BF02992776.

[7] H. Gruber, „On balanced separators, treewidth, and cycle rank“, *Journal of Combinatorics*, Jg. 3, Nr. 4, S. 669–681, 2012, ISSN: 21563527. DOI: 10.4310/joc.2012.v3.n4.a5. arXiv: 1012.1344.

[8] U. Feige, M. Hajiaghayi und J. R. Lee, „Improved Approximation Algorithms for Minimum Weight Vertex Separators“, *SIAM Journal on Computing*, Jg. 38, Nr. 2, S. 629–657, 2008. DOI: 10.1137/05064299X.

[9] T. N. Bui und C. Jones, „Finding good approximate vertex and edge partitions is NP-hard“, *Information Processing Letters*, Jg. 42, Nr. 3, S. 153–159, 1992, ISSN: 00200190. DOI: 10.1016/0020-0190(92)90140-Q.

[10] D. Marx, „Parameterized Graph Separation Problems“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2004, S. 71–82, ISBN: 9783540230717. DOI: 10.1007/978-3-540-28639-4_7.

[11] U. Feige und M. Mahdian, „Finding small balanced separators“, Techn. Ber., 2006.

[12] S. Brandt und R. Wattenhofer, „Approximating Small Balanced Vertex Separators in Almost Linear Time“, *Algorithmica*, Jg. 81, Nr. 10, S. 4070–4097, 2019, ISSN: 14320541. DOI: 10.1007/s00453-018-0490-x.

[13] R. G. Downey und M. R. Fellows, *Fundamentals of Parameterized Complexity*, Ser. Texts in Computer Science. London: Springer London, 2013, ISBN: 978-1-4471-5558-4. DOI: 10.1007/978-1-4471-5559-1.

[14] R. Impagliazzo und R. Paturi, „Complexity of k-SAT“, in *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference) (Cat.No.99CB36317)*, IEEE Comput. Soc, 1999, S. 1–4, ISBN: 0-7695-0075-7. DOI: 10.1109/CCC.1999.766282. Adresse: http://ieeexplore.ieee.org/document/766282/.

[15] H. L. Bodlaender, „A partial k-arboretum of graphs with bounded treewidth“, *Theoretical Computer Science*, Jg. 209, Nr. 1-2, S. 1–45, 1998, ISSN: 03043975. DOI: 10.1016/S0304-3975(97)00228-4.

[16] R. M. Karp, „Reducibility among Combinatorial Problems“, in *Complexity of Computer Computations*, Springer US, 1972, S. 85–103. DOI: 10.1007/978-1-4684-2001-2_9. Adresse: https://link.springer.com/chapter/10.1007/978-1-4684-2001-2_9.

[17] W. Espelage, F. Gurski und E. Wanke, „How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time“, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Jg. 2204, S. 117–128, 2001, ISSN: 16113349. DOI: 10.1007/3-540-45477-2_12.

[18] M. R. Garey, D. S. Johnson und L. Stockmeyer, „Some simplified NP-complete problems“, in *Proceedings of the sixth annual ACM symposium on Theory of computing - STOC '74*, Bd. 5, New York, New York, USA: ACM Press, 1974, S. 47–63, ISBN: 9788578110796. DOI: 10.1145/800119.803884. arXiv: arXiv:1011.1669v3.

[19] P. L. Renz, „Intersection representations of graphs by arcs“, *Pacific Journal of Mathematics*, Jg. 34, Nr. 2, S. 501–510, 1970.

[20] N. Robertson und P. D. Seymour, „Graph minors. II. Algorithmic aspects of tree-width“, *Journal of Algorithms*, Jg. 7, Nr. 3, S. 309–322, Sep. 1986, ISSN: 01966774. DOI: 10.1016/0196-6774(86)90023-4.

[21] N. Robertson, P. Seymour und R. Thomas, *Quickly excluding a planar graph*, 1994. DOI: 10.1006/jctb.1994.1073.

[22] S. Arnborg und A. Proskurowski, „Linear time algorithms for NP-hard problems restricted to partial k-trees“, *Discrete Applied Mathematics*, Jg. 23, Nr. 1, S. 11–24, 1989, ISSN: 0166218X. DOI: 10.1016/0166-218X(89)90031-0.

[23] F. V. Fomin und D. M. Thilikos, „New upper bounds on the decomposability of planar graphs“, *Journal of Graph Theory*, Jg. 51, Nr. 1, S. 53–81, Jan. 2006, ISSN: 10970118. DOI: 10.1002/jgt.20121.

[24] J. van Leeuwen, „Graph Algorithms“, in *Algorithms and Complexity*, Cambridge, MA, USA: Elsevier, 1990, S. 525–631, ISBN: 0444880712. DOI: 10.1016/B978-0-444-88071-0.50015-1.

[25] H. L. Bodlaender, „A Tourist Guide through Treewidth“, Techn. Ber. 1-2, 1994.

[26] I. Sau, „Efficient algorithms on graphs of bounded treewidth“, CNRS, LIRMM, Université de Montpellier, Grenoble, France, Techn. Ber., 2018.

[27] H. L. Bodlaender, „A linear-time algorithm for finding tree-decompositions of small treewidth“, *SIAM Journal on Computing*, Jg. 25, Nr. 6, S. 1305–1317, 1996, ISSN: 00975397. DOI: 10.1137/S0097539793251219.

[28] N. Robertson und P. Seymour, „Graph Minors .XIII. The Disjoint Paths Problem“, *Journal of Combinatorial Theory, Series B*, Jg. 63, Nr. 1, S. 65–110, Jan. 1995, ISSN: 00958956. DOI: 10.1006/jctb.1995.1006.

[29] H. L. Bodlaender und A. M. C. A. Koster, „Treewidth computations I. Upper bounds“, *Information and Computation*, Jg. 208, Nr. 3, S. 259–275, 2010, ISSN: 08905401. DOI: 10.1016/j.ic.2009.03.008.

[30] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov und M. Pilipczuk, „An $O(c^k n)$ 5-Approximation Algorithm for Treewidth“, in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, IEEE, Okt. 2013, S. 499–508, ISBN: 978-0-7695-5135-7. DOI: 10.1109/FOCS.2013.60.

[31] T. Kloks, *Treewidth: computations and approximations*. Springer Science & Business Media, 1994, Bd. 842, S. 209, ISBN: 3540583564. DOI: 10.1007/bfb0045375.

[32] A. Becker und D. Geiger, „A sufficiently fast algorithm for finding close to optimal clique trees“, *Artificial Intelligence*, Jg. 125, Nr. 1-2, S. 3–17, Jan. 2001, ISSN: 00043702. DOI: 10.1016/S0004-3702(00)00075-8.

[33] E. Amir, „Approximation Algorithms for Treewidth“, *Algorithmica*, Jg. 56, Nr. 4, S. 448–479, Apr. 2010, ISSN: 0178-4617. DOI: 10.1007/s00453-008-9180-4.

[34] ——, „Efficient Approximation for Triangulation of Minimum Treewidth“, S. 7–15, Jan. 2013. arXiv: 1301.2253.

[35] A. M. C. A. Koster, *Frequency assignment: Models and algorithms*. Arie Koster, 1999.

[36] A. M. C. A. Koster, S. P. M. van Hoesel und A. W. J. Kolen, „Solving partial constraint satisfaction problems with tree decomposition“, *Networks*, Jg. 40, Nr. 3, S. 170–180, Okt. 2002, ISSN: 0028-3045. DOI: 10.1002/net.10046.

[37] F. F. Gavril, „The Intersection Graphs of Subtrees in Trees Are Exactly the Chordal Graphs“, *Journal of Combinatorial Theory. Series B*, Jg. 16, Nr. 1, 1974, ISSN: 10960902. DOI: 10.1016/0095-8956(74)90094-X.

[38] H. L. Bodlaender und A. M. C. A. Koster, „Safe separators for treewidth“, *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithms and Combinatorics*, S. 70–78, 2004.

[39] H. L. Bodlaender, „Dynamic programming on graphs with bounded treewidth“, in, 1988, S. 105–118. DOI: 10.1007/3-540-19488-6_110.

[40] S. Arnborg, A. Proskurowski und D. G. Corneil, „Forbidden minors characterization of partial 3-trees“, *Discrete Mathematics*, Jg. 80, Nr. 1, S. 1–19, Feb. 1990, ISSN: 0012365X. DOI: 10.1016/0012-365X(90)90292-P.

[41] S. Ramachandramurthi, „The Structure and Number of Obstructions to Treewidth“, *SIAM Journal on Discrete Mathematics*, Jg. 10, Nr. 1, S. 146–157, Feb. 1997, ISSN: 0895-4801. DOI: 10.1137/S0895480195280010.

[42] K.-i. Kawarabayashi, Y. Kobayashi und B. Reed, „The disjoint paths problem in quadratic time“, *Journal of Combinatorial Theory, Series B*, Jg. 102, Nr. 2, S. 424–435, März 2012, ISSN: 00958956. DOI: 10.1016/j.jctb.2011.07.004.

[43] J. Gajarský, M. Lampis und S. Ordyniak, „Parameterized algorithms for modular-width“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Bd. 8246 LNCS, 2013, S. 163–176, ISBN: 9783319038971. DOI: 10.1007/978-3-319-03898-8_15.

[44] M. Vatshelle, „New Width Parameters of Graphs“, Diss., University of Bergen, 2012.

[45] H. L. Bodlaender und K. Jansen, „On the complexity of the maximum cut problem“, in, 1934, Bd. 74, 1994, S. 769–780. DOI: 10.1007/3-540-57785-8_189.

[46] V. Bouchitté und I. Todinca, „Treewidth and minimum fill-in: Grouping the minimal separators“, *SIAM Journal on Computing*, Jg. 31, Nr. 1, S. 212–232, Juli 2001, ISSN: 00975397. DOI: 10.1137/S0097539799359683.

[47] H. L. Bodlaender, „Treewidth: Structure and Algorithms“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Bd. 4474 LNCS, Springer, Berlin, Heidelberg, 2007, S. 11–25, ISBN: 9783540729181. DOI: 10.1007/978-3-540-72951-8_3.

[48] V. Gogate und R. Dechter, „A Complete Anytime Algorithm for Treewidth“, *arXiv preprint arXiv:1207.4109*, Nr. 1, S. 201–208, 2012. arXiv: 1207.4109.

[49] M. Yannakakis, „Computing the Minimum Fill-In is NP-Complete“, *SIAM Journal on Algebraic Discrete Methods*, Jg. 2, Nr. 1, S. 77–79, März 1981, ISSN: 0196-5212. DOI: 10.1137/0602010.

[50] J. Nešetřil und P. O. de Mendez, „Structural Properties of Sparse Graphs“, Techn. Ber. C, 2008, S. 247–251. DOI: 10.1016/j.endm.2008.06.050.

[51] H. P. Patil, „On the structure of k-trees“, *Journal of Combinatorics, Information and System Sciences*, Jg. 11, Nr. 2-4, S. 57–64, 1986.

[52] S. Arnborg, D. G. Corneil und A. Proskurowski, „Complexity of Finding Embeddings in a k -Tree“, *SIAM Journal on Algebraic Discrete Methods*, Jg. 8, Nr. 2, S. 277–284, Apr. 1987, ISSN: 0196-5212. DOI: 10.1137/0608024.

[53] J. Lagergren, „Efficient Parallel Algorithms for Graphs of Bounded Tree-Width“, *Journal of Algorithms*, Jg. 20, Nr. 1, S. 20–44, 1996, ISSN: 01966774. DOI: 10.1006/jagm.1996.0002.

[54] B. A. Reed, „Finding approximate separators and computing tree width quickly“, in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing - STOC '92*, New York, New York, USA: ACM Press, 1992, S. 221–228, ISBN: 0897915119. DOI: 10.1145/129712.129734.

[55] H. L. Bodlaender, „A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth“, *SIAM Journal on Computing*, Jg. 25, Nr. 6, S. 1305–1317, Dez. 1996, ISSN: 0097-5397. DOI: 10.1137/S0097539793251219.

[56] U. Feige, M. Hajiaghayi und J. R. Lee, „Improved approximation algorithms for minimum-weight vertex separators“, in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing - STOC '05*, New York, New York, USA: ACM Press, 2005, S. 563, ISBN: 1581139608. DOI: 10.1145/1060590.1060674.

[57] N. Robertson und P. D. Seymour, „Graph minors. X. Obstructions to tree-decomposition“, *Journal of Combinatorial Theory, Series B*, Jg. 52, Nr. 2, S. 153–190, 1991, ISSN: 10960902. DOI: 10.1016/0095-8956(91)90061-N.

[58] P. D. Seymour und R. Thomas, „Call routing and the ratcatcher“, *Combinatorica*, Jg. 14, Nr. 2, S. 217–241, 1994. Adresse: http://users.uoa.gr/$%5Csim$sedthilk/papers/price.pdf.

[59] F. V. Fomin, F. Mazoit und I. Todinca, „Computing branchwidth via efficient triangulations and blocks“, *Discrete Applied Mathematics*, Jg. 157, Nr. 12, S. 2726–2736, Juni 2009, ISSN: 0166218X. DOI: 10.1016/j.dam.2008.08.009.

[60] H. L. Bodlaender und D. M. Thilikos, „Constructive linear time algorithms for branchwidth“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Bd. 1256, 1997, S. 627–637, ISBN: 3540631658. DOI: 10.1007/3-540-63165-8_217.

[61] N. Robertson und P. D. Seymour, „Graph Minors. XX. Wagner's conjecture“, *Journal of Combinatorial Theory. Series B*, Jg. 92, Nr. 2 SPEC. ISS. S. 325–357, 2004, ISSN: 00958956. DOI: 10.1016/j.jctb.2004.08.001.

[62] N. Robertson und P. D. Seymour, „Graph Minors .XIII. The Disjoint Paths Problem“, *Journal of Combinatorial Theory, Series B*, Jg. 63, Nr. 1, S. 65–110, Jan. 1995, ISSN: 00958956. DOI: 10.1006/jctb.1995.1006. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S0095895685710064.

[63] W. Cook und P. D. Seymour, „Tour Merging via Branch-Decomposition“, *INFORMS Journal on Computing*, Jg. 15, Nr. 3, S. 233–248, 2003. DOI: 10.1287/ijoc.15.3.233.16078.

[64] F. V. Fomin und D. M. Thilikos, „Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up“, *SIAM Journal on Computing*, Jg. 36, Nr. 2, S. 281–309, Jan. 2006, ISSN: 0097-5397. DOI: 10.1137/S0097539702419649.

[65] T. Kloks, J. Kratochvíl und H. Müller, „Computing the branchwidth of interval graphs“, *Discrete Applied Mathematics*, Jg. 145, Nr. 2, S. 266–275, Jan. 2005, ISSN: 0166218X. DOI: 10.1016/j.dam.2004.01.015.

[66] J. Nešetřil und P. Ossona de Mendez, „Tree-depth, subgraph coloring and homomorphism bounds“, *European Journal of Combinatorics*, 2006, ISSN: 01956698. DOI: 10.1016/j.ejc.2005.01.010.

[67] J. Nešetřil und P. O. de Mendez, *Sparsity: Graphs, Structures, and Algorithms*. 2012, Bd. 28, ISBN: 978-3-642-27874-7. DOI: 10.1007/978-3-642-27875-4.

[68] D. R. Lick und A. T. White, „k-Degenerate Graphs“, *Canadian Journal of Mathematics*, Jg. 22, Nr. 5, S. 1082–1096, Okt. 1970, ISSN: 0008-414X. DOI: 10.4153/CJM-1970-125-1.

[69] V. Batagelj und M. Zaversnik, „An O(m) Algorithm for Cores Decomposition of Networks“, Okt. 2003. arXiv: 0310049 [cs].

[70] D. Eppstein, M. Löffler und D. Strash, „Listing all maximal cliques in large sparse real-world graphs in near-optimal time“, in *ACM Journal of Experimental Algorithmics*, Bd. 18, Association for Computing Machinery, Dez. 2013. DOI: 10.1145/2543629.

[71] C. Bron und J. Kerbosch, „Algorithm 457: finding all cliques of an undirected graph“, *Communications of the ACM*, Jg. 16, Nr. 9, S. 575–577, Sep. 1973, ISSN: 0001-0782. DOI: 10.1145/362342.362367.

[72] J. W. Moon und L. Moser, „On cliques in graphs“, *Israel Journal of Mathematics*, Jg. 3, Nr. 1, S. 23–28, März 1965, ISSN: 0021-2172. DOI: 10.1007/BF02760024.

[73] B. Courcelle, J. Engelfriet und G. Rozenberg, „Handle-rewriting hypergraph grammars“, *Journal of Computer and System Sciences*, Jg. 46, Nr. 2, S. 218–270, Apr. 1993, ISSN: 00220000. DOI: 10.1016/0022-0000(93)90004-G.

[74] B. Courcelle, J. A. Makowsky und U. Rotics, „Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width“, *Theory of Computing Systems*, Jg. 33, Nr. 2, S. 125–150, März 2000, ISSN: 1432-4350. DOI: 10.1007/s002249910009. Adresse: http://link.springer.com/10.1007/s002249910009.

[75] M. Kamiński, V. V. Lozin und M. Milanič, „Recent developments on graphs of bounded clique-width“, *Discrete Applied Mathematics*, Jg. 157, Nr. 12, S. 2747–2761, Juni 2009, ISSN: 0166218X. DOI: 10.1016/j.dam.2008.08.022.

[76] K. K. Dabrowski, M. Johnson und D. Paulusma, „Clique-width for hereditary graph classes“, in *Surveys in Combinatorics 2019*, Cambridge University Press, Juni 2019, S. 1–56. DOI: 10.1017/9781108649094.002. arXiv: 1901.00335.

[77] B. Courcelle und S. Olariu, „Upper bounds to the clique width of graphs“, *Discrete Applied Mathematics*, 2000, ISSN: 0166218X. DOI: 10.1016/S0166-218X(99)00184-5.

[78] F. Gurski und E. Wanke, „The Tree-Width of Clique-Width Bounded Graphs without $K_{n,n}$“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Bd. 1716, 2000, S. 196–205, ISBN: 3540666664. DOI: 10.1007/3-540-40064-8_19.

[79] F. Gurski, „The Behavior of Clique-Width under Graph Operations and Graph Transformations“, *Theory of Computing Systems*, Jg. 60, Nr. 2, S. 346–376, 2017, ISSN: 14330490. DOI: 10.1007/s00224-016-9685-1. Adresse: http://dx.doi.org/10.1007/s00224-016-9685-1.

[80] M. R. Fellows, F. A. Rosamond, U. Rotics und S. Szeider, „Clique-Width is NP-Complete“, *SIAM Journal on Discrete Mathematics*, Jg. 23, Nr. 2, S. 909–939, Jan. 2009, ISSN: 0895-4801. DOI: 10.1137/070687256. Adresse: http://epubs.siam.org/doi/10.1137/070687256.

[81] D. G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed und U. Rotics, „Polynomial Time Recognition of Clique-Width *leq* 3 Graphs“, in *SIAM Journal on Discrete Mathematics*, 2, Bd. 23, Jan. 2000, S. 126–134, ISBN: 3540666664. DOI: 10.1007/10719839_14. Adresse: http://link.springer.com/10.1007/10719839_14.

[82] B. Courcelle, J. A. Makowsky und U. Rotics, „On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic“, *Discrete Applied Mathematics*, Jg. 108, Nr. 1-2, S. 23–52, Feb. 2001, ISSN: 0166218X. DOI: 10.1016/S0166-218X(00)00221-3. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S0166218X00002213.

[83] S.-I. Oum und P. D. Seymour, „Approximating clique-width and branch-width“, *Journal of Combinatorial Theory. Series B*, Jg. 96, Nr. 4, S. 514–528, Juli 2006, ISSN: 00958956. DOI: 10.1016/j.jctb.2005.10.006.

[84] *Parameter: cliquewidth*, 2020. Adresse: https://www.graphclasses.org/classes/par_12.html (besucht am 17. 11. 2020).

[85] T. Gallai, „Transitiv orientierbare graphen“, *Acta Mathematica Academiae Scientiarum Hungarica*, Jg. 18, Nr. 1-2, S. 25–66, März 1967, ISSN: 0001-5954. DOI: 10.1007/BF02020961. Adresse: https://link.springer.com/article/10.1007/BF02020961%20http://link.springer.com/10.1007/BF02020961.

[86] R. M. McConnell und J. P. Spinrad, „Modular decomposition and transitive orientation“, *Discrete Mathematics*, Jg. 201, Nr. 1-3, S. 189–241, Apr. 1999, ISSN: 0012365X. DOI: 10.1016/S0012-365X(98)00319-7.

[87] F. N. Abu-Khzam, S. Li, C. Markarian, F. Meyer auf der Heide und P. Podlipyan, „Modular-Width: An Auxiliary Parameter for Parameterized Parallel Complexity“, in *Lecture Notes in Computer Science (including subseries Lecture*

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Bd. 10336 LNCS, Springer Verlag, 2017, S. 139–150, ISBN: 9783319596044. DOI: 10.1007/978-3-319-59605-1_13.

[88] S.-i. Oum, „Rank-width: Algorithmic and structural results“, *Discrete Applied Mathematics*, Jg. 231, S. 15–24, Nov. 2017, ISSN: 0166218X. DOI: 10.1016/j.dam.2016.08.006. arXiv: 1601.03800.

[89] R. Rabinovich, „Complexity Measures for Directed Graphs“, Diss., Rheinisch-Westfälische Technische Hochschule Aachen, 2008, S. 123.

[90] N. Lodha, S. Ordyniak und S. Szeider, „SAT-Encodings for Special Treewidth and Pathwidth“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Bd. 10491 LNCS, Springer Verlag, 2017, S. 429–445, ISBN: 9783319662626. DOI: 10.1007/978-3-319-66263-3_27. Adresse: www.ac.tuwien.ac.at/tr.

[91] R. Ganian, N. Lodha, S. Ordyniak und S. Szeider, „SAT-encodings for treecut width and treedepth“, in *Proceedings of the Workshop on Algorithm Engineering and Experiments*, 2019. DOI: 10.1137/1.9781611975499.10.

[92] H. Dell, T. Husfeldt, B. M. P. Jansen, P. Kaski, C. Komusiewicz und F. A. Rosamond, „The First Parameterized Algorithms and Computational Experiments Challenge“, in *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, Bd. 63, 2017, S. 1–9, ISBN: 9783959770231. DOI: 10.4230/LIPIcs.IPEC.2016.30. Adresse: https://github.com/TomvdZanden/BZTreewidth].

[93] H. Tamaki, „Positive-instance driven dynamic programming for treewidth“, Apr. 2017. arXiv: 1704.05286. Adresse: http://arxiv.org/abs/1704.05286.

[94] H. Dell, C. Komusiewicz, N. Talmon und M. Weller, „The PACE 2017 Parameterized Algorithms and Computational Experiments challenge: The second iteration“, *Leibniz International Proceedings in Informatics, LIPIcs*, Jg. 89, Nr. 30, S. 1–12, Feb. 2018, ISSN: 18688969. DOI: 10.4230/LIPIcs.IPEC.2017.30.

[95] T. H. Cormen, C. E. Leiserson, R. L. Rivest und C. Stein, *Introduction to Algorithms, 3rd Ed.* Massachusetts Institute of Technology Press, 2009. Adresse: http://mitpress.mit.edu/sites/default/files/titles/content/9780262033848_pre_0001.pdf.

[96] P. D. Seymour und R. Thomas, „Graph Searching and a Min-Max Theorem for Tree-Width“, *Journal of Combinatorial Theory, Series B*, Jg. 58, Nr. 1, S. 22–33, Mai 1993, ISSN: 10960902. DOI: 10.1006/jctb.1993.1027.

[97] R. Tarjan, „Depth-First Search and Linear Graph Algorithms“, *SIAM Journal on Computing*, Jg. 1, Nr. 2, S. 146–160, Juni 1972, ISSN: 0097-5397. DOI: 10.1137/0201010. Adresse: http://epubs.siam.org/doi/10.1137/0201010.

[98] H. L. Bodlaender, A. M. C. A. Koster und F. Van Den Eijkhof, „Preprocessing rules for triangulation of probabilistic networks“, *Computational Intelligence*, Jg. 21, Nr. 3, S. 286–305, 2005, ISSN: 08247935. DOI: 10.1111/j.1467-8640.2005.00274.x.

[99] D. J. Rose, R. E. Tarjan und G. S. Lueker, „Algorithmic Aspects of Vertex Elimination on Graphs“, *SIAM Journal on Computing*, Jg. 5, Nr. 2, S. 266–283, Juni 1976, ISSN: 0097-5397. DOI: 10.1137/0205021. Adresse: http://epubs.siam.org/doi/10.1137/0205021.

[100] F. V. Fomin, P. A. Golovach, D. Lokshtanov und S. Saurabh, „Intractability of Clique-Width Parameterizations“, *SIAM Journal on Computing*, Jg. 39, Nr. 5, S. 1941–1956, Jan. 2010, ISSN: 0097-5397. DOI: 10.1137/080742270. Adresse: http://epubs.siam.org/doi/10.1137/080742270.

[101] ——, „Algorithmic Lower Bounds for Problems Parameterized by Clique-width“, in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, PA: Society for Industrial und Applied Mathematics, Jan. 2010, S. 493–502, ISBN: 978-0-89871-701-3. DOI: 10.1137/1.9781611973075.42.

[102] S. H. Sæther und J. A. Telle, „Between Treewidth and Clique-Width“, *Algorithmica*, Jg. 75, Nr. 1, S. 218–253, Mai 2016, ISSN: 0178-4617. DOI: 10.1007/s00453-015-0033-7. Adresse: http://link.springer.com/10.1007/s00453-015-0033-7.

[103] M. Samer, „Hypertree-decomposition via Branch-decomposition“, Institute of Information Systems (DBAI), Vienna University ofTechnology, Austria, Wien, Techn. Ber.

[104] G. Gottlob, N. Leone und F. Scarcello, „Comparison of structural CSP decomposition methods“, *Artificial Intelligence*, Jg. 124, Nr. 2, S. 243–282, 2000, ISSN: 00043702. DOI: 10.1016/S0004-3702(00)00078-3.

[105] ——, „Hypertree Decompositions and Tractable Queries“, *Journal of Computer and System Sciences*, Jg. 64, Nr. 3, S. 579–627, Mai 2002, ISSN: 00220000. DOI: 10.1006/jcss.2001.1809.

[106] J. Jeong, E. J. Kim und S.-i. Oum, „Finding branch-decompositions of matroids, hypergraphs, and more“, *Leibniz International Proceedings in Informatics, LIPIcs*, 2018. DOI: 10.4230/LIPIcs.ICALP.2018.80.

[107] E. Cohen und M. Tarsi, „NP-Completeness of graph decomposition problems“, *Journal of Complexity*, Jg. 7, Nr. 2, S. 200–212, Juni 1991, ISSN: 10902708. DOI: 10.1016/0885-064X(91)90006-J.

[108] F. Dorn, E. Penninkx, H. L. Bodlaender und F. V. Fomin, „Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Decompositions“, Jg. 58, S. 790–810, 2010. DOI: 10.1007/s00453-009-9296-1.

[109] C. Gutwenger und P. Mutzel, „A linear time implementation of SPQR-trees“, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Bd. 1984, Springer Verlag, 2001, S. 77–90, ISBN: 3540415548. DOI: 10.1007/3-540-44541-2_8.

[110] A. M. C. A. Koster, H. L. Bodlaender und S. P. M. van Hoesel, „Treewidth: Computational experiments“, *Electronic Notes in Discrete Mathematics*, Jg. 8, Nr. December, S. 1–4, 2001, ISSN: 15710653.

[111] J. Gramm, J. Guo, F. Hüffner und R. Niedermeier, „Data reduction and exact algorithms for clique cover“, *ACM Journal of Experimental Algorithmics*, Jg. 13, Nr. 2, Feb. 2009, ISSN: 1084-6654. DOI: 10.1145/1412228.1412236.

[112] C. Wang, C. Xu und A. Lisser, „Bandwidth Minimization Problem“, in *Optimisation et Simulation*, 2014. Adresse: https://hal.archives-ouvertes.fr/hal-01166658.

[113] J. E. Hopcroft und R. Tarjan, „Algorithm 447: efficient algorithms for graph manipulation“, *Communications of the ACM*, Jg. 16, Nr. 6, S. 372–378, Juni 1973, ISSN: 0001-0782. DOI: 10.1145/362248.362272. Adresse: https://dl.acm.org/doi/10.1145/362248.362272.

[114] J. E. Hopcroft und R. E. Tarjan, „Dividing a Graph into Triconnected Components“, *SIAM Journal on Computing*, Jg. 2, Nr. 3, S. 135–158, Sep. 1973, ISSN: 0097-5397. DOI: 10.1137/0202012.