Technische Universität Berlin
Fakultät IV: Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Algorithmik und Komplexitätstheorie (AKT)

Parametrisierte Algorithmen zum Finden von Petersen-Teilgraphen

Bachelorarbeit

von Thomas Jaron-Strugala

zur Erlangung des Grades „Bachelor of Science“ (B. Sc.)
im Studiengang Informatik

Erstgutachter:	Prof. Dr. Rolf Niedermeier
Zweitgutachter:	Prof. Dr. Sabine Glesner
Betreuer:	Dr. Matthias Bentert, Prof. Dr. Rolf Niedermeier

September 2021

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbstständige und eigenhändige Ausfertigung versichert an Eides statt:

Berlin, den _____ _____

Datum Unterschrift

Zusammenfassung

Der Petersen-Graph ist ein gängiges Gegenbeispiel für Aussagen in der Graphentheorie und für vermeintlich auf alle Graphen anwendbare Algorithmen. Demzufolge ist es nützlich zu wissen, ob Petersen-Teilgraphen in einer Eingabeinstanz vorhanden sind. Das Finden eines Petersen-Teilgraphen ist in einer Zeit von $O(n^{10})$ bei n Knoten bzw. $O(m^5)$ bei m Kanten und somit in polynomieller Zeit möglich. In dieser Arbeit entwickeln wir zu diesem Problem parametrisierte Algorithmen und erzielen mit diesen Laufzeiten, welche für Instanzen mit kleinen Parameterwerten schneller als die unparametrisierten Algorithmen sind. Somit leistet diese Arbeit einen Beitrag zum Themengebiet **FPT in P**, welches sich mit parametrisierten Algorithmen zu Polynomzeitproblemen beschäftigt. Konkret beschreiben wir unter anderem einen generischen parametrisierten Algorithmus für Distanzparameter wie `Distance to Clique` oder `Distance to Bipartite` sowie ein dynamisches Programm für den Parameter `Clique-Width` k, welcher eine Laufzeit von $O(10^k \cdot n)$ erzielt. Weiterhin zeigen wir ℓ-Grundproblem-Härte für die Parameter `Minimum Degree`, `Bisection Width` und `Maximum Dominating Set`, sowie alle Parameter in der Graphparameterhierarchie, welche unter diesen Parametern liegen. Mit diesem Härtebegriff lässt sich zeigen, dass für einen Parameter keine parametrisierten Algorithmen existieren, welche schneller als die unparametrisierten Algorithmen sind.

Inhaltsverzeichnis

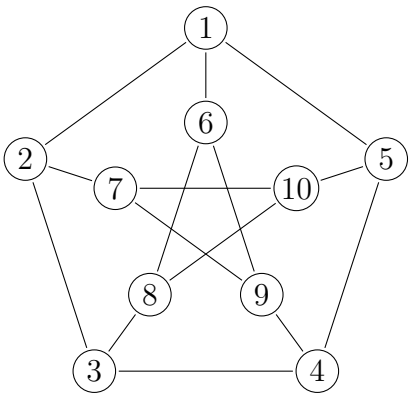

Abbildung 1.1: Der Petersen-Graph

1 Einführung

Der Petersen-Graph, benannt nach Julius Petersen, einem dänischen Mathematiker, ist ein drei-regulärer Graph mit zehn Knoten. Zur Veranschaulichung ist in Abbildung 1.1 ein Petersen-Graph dargestellt. Konstruiert wurde dieser Graph 1898 von Petersen als Gegenbeispiel für die Aussage, dass ein verbundener, brückenloser, kubischer Graph eine Kantenfärbung mit drei Farben besitzt. Der Petersen-Graph ist der kleinste Graph dieser Art ohne eine Drei-Färbung [HS93].

Wir wollen mit dieser Arbeit einen Beitrag zum Themengebiet der Graphenanalyse leisten. Dieses Themengebiet setzt sich mit der Struktur und den Eigenschaften von Graphen auseinander und wie sich diese auf Algorithmen, die mit Graphen arbeiten, auswirken. Dabei liegt bei der Analyse von Algorithmen neben der Laufzeit auch die Korrektheit des Algorithmus im Fokus: Vergleicht man zwei Algorithmen miteinander und erhält für eine bestimmte Eingabeinstanz zwei verschiedene Ergebnisse, so ist eine genauere Betrachtung dieser Instanz sinnvoll. Möglicherweise verursacht eine in der Instanz vorhandene Struktur die unterschiedlichen Ergebnisse. Wenn wir allgemeine Graphen betrachten, so kann es nützlich sein, zu erkennen, ob ein Graph einen induzierten Petersen-Teilgraphen enthält. Da der Petersen-Graph in der Graphentheorie vor allem als Gegenbeispiel verwendet wird, können Algorithmen möglicherweise eine bessere Laufzeit erzielen, wenn diese davon ausgehen können, dass der Eingabegraph Petersen-Graph-frei ist.

In der nachfolgenden Arbeit befassen wir uns mit dem Problem Petersen Graph Detection, das wie folgt definiert ist.

Petersen Graph Detection
Eingabe: Ein ungerichteter Graph G.
Frage: Enthält G einen Petersen-Graphen als induzierten Teilgraphen?

Einen unparametrisierten, „naiven“ Algorithmus, der Petersen Graph Detection löst, beschreiben wir in Kapitel 2. Wenn in der nachfolgenden Arbeit vom *naiven Algorithmus* die Rede ist, so wird von diesem Algorithmus gesprochen.

Im weiteren Verlauf der Arbeit versuchen wir durch parametrisierte Algorithmen bessere Laufzeiten zu erzielen und schauen uns an, welche Parameter diesbezüglich geeignet sind und welche nicht. Im Fokus dieser Arbeit liegt die Graphparameter-Hierarchie [Sch20] in Abbildung 1.2. In dieser Hierarchie werden Abhängigkeiten zwischen Graphparametern hergestellt: Es gilt, dass Parameter, welche hierarchisch über anderen Parametern liegen, eine obere Schranke für diese Parameter darstellen. Weiterhin deutet die Färbung der Knoten auf die Anwendbarkeit der Parameter für unser Problem hin. Die Bedeutung der Färbungen kann der Abbildung entnommen werden.

1.1 Verwandte Literatur

Forschung zu parametrisierten Algorithmen bzgl. des Petersen-Graphen gibt es bisher nicht. Allerdings gibt es im Zusammenhang mit dem Petersen-Graphen zahlreiche Arbeiten und Forschungen. Beispielsweise bilden die 1969 von Watkins [Wat69] benannten generalisierten Petersen-Graphen eine Graphklasse, welche den Petersen-Graphen enthalten und einen der Konstruktionswege des Petersen-Graphen generalisieren.

Diese Arbeit leistet einen Beitrag für das Themengebiet **FPT in P**, welches unter anderem durch Giannopoulou et al. [GMN17] erarbeitet wurde. Dabei geht es um parametrisierte Algorithmen für Probleme, welche in polynomieller Laufzeit gelöst werden können. Ziel hierbei ist es, effizientere Algorithmen für Probleme mit unattraktiven Laufzeiten zu entwickeln. In diesem Kontext haben Bentert et al. [Ben+19] dazu die *ℓ-Grundproblem-Härte* definiert, welche im Verlauf dieser Arbeit zum Einsatz kommen wird, um aufzuzeigen, welche Parameter für unser Problem nicht von Nutzen sind. Al-Hajjana [AH19] hat eine Arbeit zur Erkennung von Diamanten, also vollständigen Graphen über vier Knoten mit einer fehlenden Kante, mit parametrisierten Algorithmen verfasst. Ähnlich wie in dieser Arbeit deckt er dabei bezüglich dieses Problems einige Parameter der Graphparameter-Hierarchie ab. Ebenso wie in dieser Arbeit nutzt er hierfür das Konzept der *ℓ-Grundproblem-Härte*.

Weiterhin beschäftigten sich Abboud et al. [AWW16] im Themengebiet **FPT in P** mit parametrisierten Algorithmen zur Berechnung der Graphparameter `Radius` und `Diameter` in dünnbesetzten, also kantenarmen Graphen. Hierbei beschreibt `Radius` die kleinste maximale Distanz, die ein Knoten v zu allen anderen Knoten des Graphen aufweist. Im Gegensatz dazu beschreibt `Diameter` die größte Distanz zwischen zwei Knoten. Im Allgemeinen lassen sich diese Parameter für einen Graphen in $O(n^3)$ Zeitschritten berechnen. Jedoch wird gezeigt, dass es mithilfe von parametrisierten Algorithmen möglich ist, diese Parameter in subquadratischer Laufzeit zur Eingabegröße zu berechnen, wenn der gegebene Graph dünnbesetzt ist.

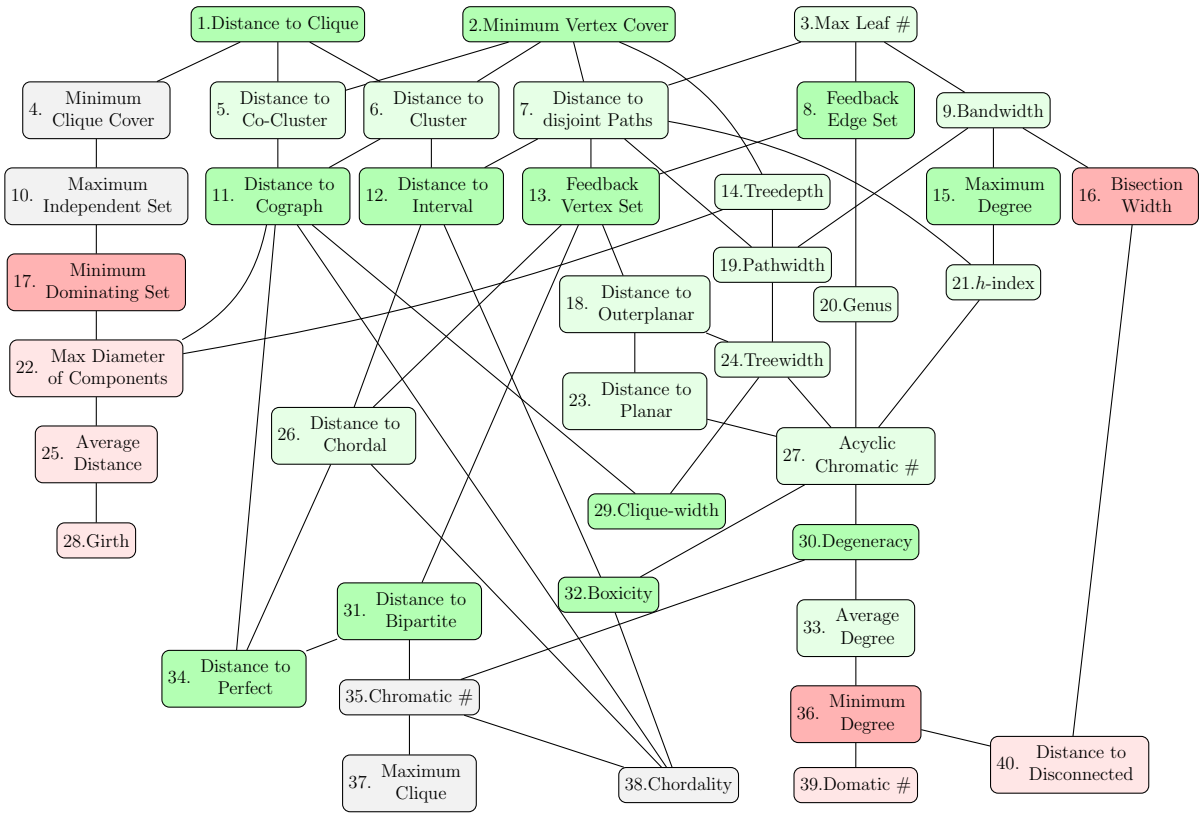

Abbildung 1.2: Die Graphparameter-Hierarchie.
Besteht eine Kante zwischen zwei Parametern, so bedeutet dies, dass der höher liegende Parameter k_1 den darunterliegenden Parameter k_2 in der Größe von oben beschränkt: Es gibt eine Funktion f, sodass $k_2 \leq f(k_1)$ für alle Graphen gilt. Diese Eigenschaft ist transitiv. Für den Parameter `Average Degree` gilt zum Beispiel, dass dieser immer größer oder gleich dem Parameter `Minimum Degree` für eine gegebene Eingabeinstanz ist.
Ist ein Knoten grün gefärbt, so kann dieser Parameter genutzt werden, um bei kleinen Parameterwerten eine bessere Laufzeit als der naive Algorithmus zu erreichen.
Ist ein Knoten hellgrün gefärbt, so wissen wir als Konsequenz aus der Existenz anderer Algorithmen, dass es einen parametrisierten Algorithmus zu diesem Parameter geben muss. Diese Parameter werden in dieser Arbeit nicht oder nicht im Detail betrachtet.
Eine rote Färbung sagt aus, dass wir mit diesem Parameter unter bestimmten Voraussetzungen keine bessere Laufzeit erzielen können (Näheres dazu in Kapitel 2.3).
Hellrot gefärbte Knoten stehen in Abhängigkeit von rot gefärbten Knoten (obere Schranke), weshalb die Verwendung dieser Parameter ebenfalls keine bessere Laufzeit erzielen kann.
Graue Knoten deuten darauf hin, dass wir in dieser Arbeit nicht sagen können, ob ein Parameter hinsichtlich unseres Problems nützlich zur Verwendung für parametrisierte Algorithmen ist oder nicht.

1.2 Resultate dieser Arbeit

Im Verlauf dieser Arbeit werden wir die Parameter in der eingangs dargestellten Graphparameter-Hierarchie in Algorithmen für Petersen Graph Detection nutzen oder zeigen, dass mit ihnen keine besseren Laufzeiten erzielt werden kann (*ℓ-Grundproblem-Härte*). Konkrete Ergebnisse können wir beispielsweise für die sogenannten „Distanzparameter“, welche im folgenden Kapitel definiert werden, erzielen. Hierfür beschreiben wir einen generellen Algorithmus, der für all diese Parameter genutzt werden kann. Außerdem beschreiben wir ein dynamisches Programm, welcher Petersen Graph Detection parametrisiert mit dem Parameter `Clique-Width` löst. Letztendlich resultiert die Arbeit darin, dass ein sehr großer Teil der Graphparameter-Hierarchie klassifiziert wurde und die Anzahl der Parameter, für die weiterhin eine Klassifizierung offen ist, im Grunde niedrig ist: Die Schwelle zwischen *ℓ-Grundproblem-harten* Parametern und Parametern, welche einen Nutzen im Bezug auf unser Grundproblem bieten, wird deutlich aufgezeichnet.

2 Notation und Grundlegendes

In diesem Kapitel definieren wir die grundlegende Notation und beschreiben die in dieser Arbeit näher betrachteten Parameter. Außerdem gehen wir auf das Konzept der ℓ-*Grundproblem-Härte* ein, welches ein nützliches Werkzeug darstellt, um zu zeigen, dass eine bessere Laufzeit mit einem bestimmten Parameter unter bestimmten Bedingungen nicht erzielt werden kann. Das Problem, auf welches wir in dieser Arbeit eingehen, erhält einen ungerichteten Graphen $G = (V, E)$ als Eingabe, wobei V die Knotenmenge und $E \subseteq \{\{v, w\} \mid v, w \in V, v \neq w\}$ die Kantenmenge bezeichnet. Weiterhin nennen wir

n	die Anzahl $\lvert V\rvert$ der Knoten;
m	die Anzahl $\lvert E\rvert$ der Kanten;
$G[V']$	den durch die Knotenmenge V' induzierten Teilgraphen von G;
$\boldsymbol{C}_x$	einen Kreisgraphen mit x Knoten;
$\boldsymbol{C}_{\geq x}$	einen Kreisgraphen mit mindestens x Knoten und
$\boldsymbol{P}_{\geq x}$	einen Pfad mit x Knoten.

Parametrisierte Komplexität. Wir verwenden Standardnotation aus der parametrisierten Komplexität und Algorithmik [Nie06]. Hierfür erläutern wir ein paar zentrale Begriffe:

- Ein **parametrisiertes Problem** ist ein Problem, welches neben der klassischen Instanz einen Parameter als Eingabe erhält. Formal definiert ist das parametrisierte Problem als eine Sprache $L \subseteq \Sigma^* \times \mathbb{N}$, wobei Σ ein endliches Alphabet darstellt. Die natürliche Zahl stellt den Parameter dar.
- **FPT** steht für *fixed parameter tractability* bzw. die Komplexitätsklasse der Probleme, welche im Bezug zu einem Parameterwert k und einer Funktion f in einer Laufzeit von $f(k) \cdot |x|^{O(1)}$ gelöst werden können. Mithilfe von Parametern wird versucht, bessere Laufzeiten für Probleme zu finden, welche nicht in polynomieller Zeit gelöst werden können. Ist der Parameterwert für eine Eingabe klein, oder nimmt man den Parameterwert als konstant an, so lässt sich das Problem schneller lösen als im unparametrisierten, nicht-polynomiellen Fall.
- In dem Themengebiet **FPT in P** wird versucht, Werkzeuge aus dem Bereich **FPT** auf polynomzeitlösbare Probleme anzuwenden. Mit diesen ist es unter Umständen möglich, durch den Einsatz von parameterabhängigen Faktoren die polynomiellen Faktoren in der Eingabegröße zu verbessern.

2.1 Der Unparametrisierte Algorithmus

Der folgende Algorithmus löst das Problem Petersen Graph Detection. Der unparametrisierte oder naive Algorithmus rät zehn Knoten der Knotenmenge des Eingabegraphen und überprüft, ob diese Teilmenge einen Petersen-Graphen induziert. Diese Überprüfung geschieht mittels der Funktion check-graph, welche später in anderen Algorithmen ebenfalls referenziert wird: Als Eingabe kann diese Funktion Graphen oder Knotenmengen erhalten. Bei Knotenmengen wird der durch diese zehn Knoten induzierte Subgraph betrachtet. Wenn die Eingabe einen Petersen-Graph darstellt, dann gibt check-graph True zurück, ansonsten False. Diese Funktion hat eine konstante Laufzeit, wenn eine Adjazenzmatrix gegeben ist. Soweit nicht anders beschrieben, stellen der unparametrisierte Algorithmus und alle weiteren Algorithmen dieser Arbeit zunächst eine solche Adjazenzmatrix in $O(n^2)$ Zeitschritten auf. Anschließend rät der Algorithmus in $O(n^{10})$ Zeitschritten die zehnelementige Knotenteilmenge und überprüft diese mit check-graph. Eine Lösung für Petersen Graph Detection wird mit diesem Algorithmus somit in $O(n^2 + n^{10}) = O(n^{10})$ Zeitschritten gefunden. Analog dazu gibt es einen unparametrisierten Algorithmus mit einer Laufzeit von $O(m^5)$: Anstelle von zehn Knoten können fünf Kanten geraten werden, welche nicht zueinander inzident sind. Da diese fünf Kanten *knotendisjunkt* sind, lässt sich aus diesen Kanten eine zehnelementige Knotenmenge ableiten, welche durch check-graph überprüft werden kann. Dies ist möglich, weil der Petersen-Graph ein *perfektes Matching* besitzt: Es ist möglich, alle Knoten des Petersen-Graphen durch eine Wahl solcher fünf Kanten abzudecken. Ein Beispiel wäre die Wahl der Kanten $\{1, 6\}, \{2, 7\}, \{3, 8\}, \{4, 9\}$ und $\{5, 10\}$ (siehe Abbildung 1.1). Die $O(m^5)$-Variante des Algorithmus kann in der Praxis eine bessere Laufzeit aufweisen: Ein Graph kann höchstens n^2 viele Kanten besitzen und besitzt in den meisten Fällen weniger. Im schlimmsten Fall liefert dieser Algorithmus also in $O(m^5) = O((n^2)^5) = O(n^{10})$ Zeitschritten eine Lösung, ist in den meisten Fällen aber deutlich schneller.

2.2 Graphparameter

Im Folgenden gehen wir auf die in dieser Arbeit genutzten Graphparameter ein. Dabei lassen sich einige Parameter aufgrund ihrer hierarchischen Abhängigkeiten oder Ähnlichkeiten zueinander gruppieren. Beispielsweise kann man die distanzbasierten Parameter wie `Distance to Interval` oder `Distance to Cograph` für einen parametrisierten Algorithmus bezüglich Petersen Graph Detection auf die selbe Weise anwenden. „Distanz“ bezieht sich hierbei auf eine Menge von Knoten, welche aus dem Eingabegraphen entfernt werden müssen, damit der Restgraph Teil der im Parameter genannten Graphklasse wird. Im Folgenden definieren wir die in dieser Arbeit betrachteten Parameter:

- `Chromatic Number` Als `Chromatic Number` bezeichnet man die kleinste Anzahl an nötigen Farben, um einen Graphen zu färben, sodass keine zwei Nachbarn die

gleiche Farbe haben. Die Berechnung der `Chromatic Number` eines Graphen ist **NP**-schwer [Kar72].

- `Maximum Independent Set` Das `Maximum Independent Set` eines Graphen bezeichnet die größtmögliche Knotenmenge eines Graphen, für welche gilt, dass keine Kanten zwischen den Knoten dieser Menge existieren. Die Berechnung des `Maximum Independent Sets` eines Graphen ist **NP**-schwer [Kar72].

- `Distance to Clique` Als Clique bezeichnet man einen Graphen, für den gilt, dass alle Knoten des Graphen paarweise mit einer Kante verbunden sind. Das Berechnen einer größtmöglichen Clique eines Graphen ist **NP**-schwer [GJ79; Kar72].

- `Distance to Interval` Die Klasse der Intervallgraphen beschreibt Graphen, welche durch ein Intervallmodell dargestellt werden können. Dabei stellen die Knoten des Graphen V die Intervalle I_v innerhalb des Intervallmodells dar. In diesem Modell existiert eine Überschneidung von Intervallen I_x und I_y genau dann, wenn $\{x, y\} \in E$. Weiterhin enthalten Intervallgraphen keine induzierten $\boldsymbol{C}_{\geq 4}$. Für jeden Intervallgraphen gilt, dass jeder Subgraph dieses Graphen auch ein Intervallgraph ist: Ein Graph der Intervallgraphklasse „vererbt“ die Graphklassenzugehörigkeit an all seine Subgraphen. Lewis und Yannakakis haben gezeigt, dass die Berechnung der Distanzmenge zu Graphklassen, die so eine „Erbschaftseigenschaft“ aufweisen und nicht-trivial sind, **NP**-schwer ist [LY80]. Eine Eigenschaft ist nicht-trivial, wenn diese Eigenschaft für unendlich viele Graphen zutreffend und für unendlich viele Graphen nicht zutreffend ist. Da beide Fälle für Intervallgraphen gelten, ist die Berechnung dieses Parameters **NP**-schwer.

- `Distance to Cograph` Die Klasse der Co-Graphen beschreibt Graphen, welche frei von induzierten $\boldsymbol{P}_4$'s sind. Die Berechnung der Distanzmenge ist **NP**-schwer, da alle Subgraphen eines Co-Graphen auch Co-Graphen sind [LY80].

- `Distance to Perfect` Ein perfekter Graph ist ein Graph, für welchen die chromatische Zahl jedes induzierten Subgraphen gleich der Größe der größten Clique dieses Subgraphen ist. Die Berechnung der Distanzmenge ist **NP**-schwer, da alle Subgraphen eines perfekten Graphen auch perfekte Graphen sind [LY80].

- `Distance to Bipartite` Ein bipartiter Graph ist ein Graph, dessen Knotenmenge in zwei Partitionen geteilt werden kann, sodass jede Partition ein Independent Set bildet und Kanten nur zwischen den Partitionen existieren. Die Berechnung der Distanzmenge ist **NP**-schwer, da alle Subgraphen eines bipartiten Graphen auch bipartite Graphen sind [LY80].

- `Minimum Vertex Cover` Als `Minimum Vertex Cover` bezeichnet man eine Knotenteilmenge des Graphen, sodass jede Kante des Graphen mindestens einen Endknoten in dieser Knotenteilmenge besitzt. Für einen Graphen gilt, dass alle Knoten, welche nicht im `Minimum Vertex Cover` vorhanden sind, ein `Maximum Independent`

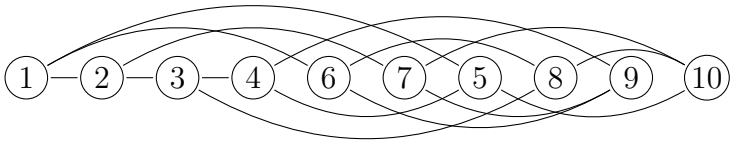

Abbildung 2.1: Eine zum Petersen-Graphen zugehörige *degeneracy order*.

`Set` des Graphen bilden. Somit kann man diesen Parameter als „Distance to Independent Set“ interpretieren. Die Berechnung des `Minimum Vertex Covers` eines Graphen ist **NP**-schwer [Kar72]. Allerdings ist es möglich, eine 2-Approximation des `Minimum Vertex Covers` in Linearzeit zu berechnen [Åst+09].

- `Feedback Vertex Set` Als `Feedback Vertex Set` wird die kleinste Menge der Knoten bezeichnet, welche aus einem Graphen entfernt werden müssen, damit der Restgraph einen Wald darstellt. Demzufolge stellt dieser Parameter die Distanz zu Wäldern dar. Die Berechnung des `Feedback Vertex Sets` ist **NP**-schwer [Kar72].

- `Feedback Edge Set` Als `Feedback Edge Set` wird die kleinste Menge der Kanten bezeichnet, welche aus einem Graphen entfernt werden müssen, damit der Restgraph einen Wald darstellt. Das `Feedback Edge Set` eines Graphen kann in Linearzeit berechnet werden, indem man mithilfe der Tiefensuche oder Breitensuche einen Spannbaum des Graphen sucht und alle restlichen Kanten dem `Feedback Edge Set` zuweist. Die Größe des `Feedback Edge Sets` kann außerdem durch $m - n + |C|$ berechnet werden, wobei C die Anzahl der Zusammenhangskomponenten des Graphen darstellt.

- `Maximum Degree` Der `Maximum Degree` Δ beschreibt den größten Knotengrad eines Knotens im Graphen. Dieser Parameter lässt sich in Linearzeit berechnen.

- `Degeneracy` Die `Degeneracy` eines Graphen ist die kleinste Zahl d, sodass der Graph und jeder Subgraph des Graphen einen Knoten mit maximal d Nachbarn besitzt. Eine dazugehörige *degeneracy order* $\leq_d$ für die Knoten des Graphen ordnet die Knoten so, dass $|N(v) \cap \{w | v \leq_d w\}| \leq d$ für jeden Knoten v gilt. Diese *degeneracy order* lässt sich in linearer Zeit berechnen [MB83]. Abbildung 2.1 stellt eine mögliche *degeneracy order* für den Petersen-Graphen dar.

- `Average Degree` Der `Average Degree` $a = 2m/n$ beschreibt den durchschnittlichen Knotengrad der Knoten im Graphen. Dieser Parameter lässt sich in Linearzeit berechnen.

- `Minimum Degree` Der `Minimum Degree` δ beschreibt die kleinste Nachbarschaft eines Knotens im Graphen. Dieser Parameter lässt sich in Linearzeit berechnen.

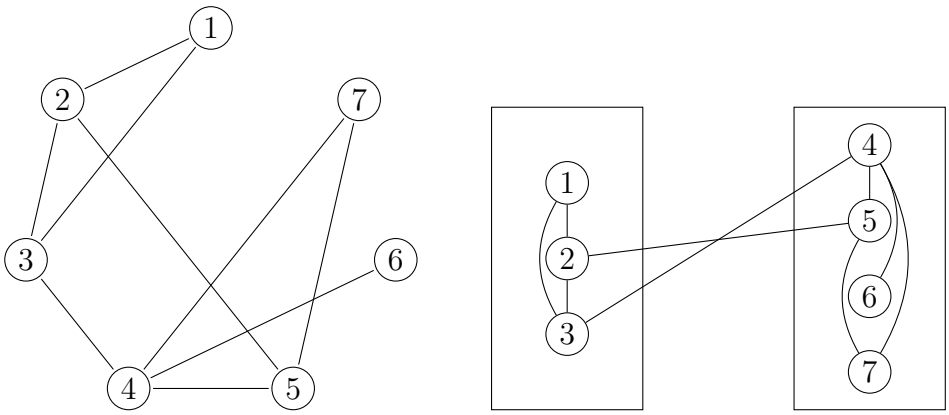

Abbildung 2.2: Ein Graph mit `Bisection Width` zwei. Die Boxen stellen jeweils eine Partition dar. Die *width* dieser Partitionierung wird durch die Anzahl der Kanten zwischen den Partitionen dargestellt, und da diese Partitionierung gleichzeitig die geringste *width* besitzt, stimmt sie mit der `Bisection Width` dieses Graphen überein.

- `Bisection Width` Betrachtet man eine Partitionierung eines Graphen in zwei Mengen, so bezeichnet man die Anzahl der Kanten, welche die Partitionen miteinander verbinden, als *width* dieser Partitionierung. In diesem Zusammenhang ist die `Bisection Width` eines Graphen die kleinstmögliche *width*, die eine Zwei-Partitionierung von G in zwei gleich große oder um eins unterschiedlich große Knotenmengen hat. Eine Veranschaulichung dieses Parameters ist in Abbildung 2.2 zu finden. Die Berechnung der `Bisection Width` eines Graphen ist **NP**-schwer [GJS76].

- `Minimum Dominating Set` Das `Minimum Dominating Set` beschreibt die kleinstmögliche Knotenmenge eines Graphen, sodass jeder Knoten des Graphen Teil dieser Menge ist oder zu einem Knoten in dieser Menge benachbart ist. Die Berechnung des `Minimum Dominating Sets` eines Graphen ist **NP**-schwer [GJ79].

- `Minimum Clique Cover` Das `Minimum Clique Cover` eines Graphen ist eine Partitionierung der Knotenmenge in die kleinstmögliche Anzahl an Cliquen. Die Berechnung des `Minimum Clique Covers` eines Graphen ist **NP**-schwer [Kar72].

- `Clique-Width` Die `Clique-Width` eines Graphen G ist ein Parameter, welcher eine Aussage über die strukturelle Komplexität eines Graphen trifft. Die `Clique-Width` ist definiert als die kleinste Anzahl an *Farben*, die benötigt wird, um G nach folgenden vier Operationen zu konstruieren:
  - (R1) Erstelle einen neuen Knoten v mit *Farbe* i
  - (R2) Ersetze die *Farbe* i durch *Farbe* j
  - (R3) Führe eine disjunkte Vereinigung zweier unterschiedlich *gefärbter* Graphen durch

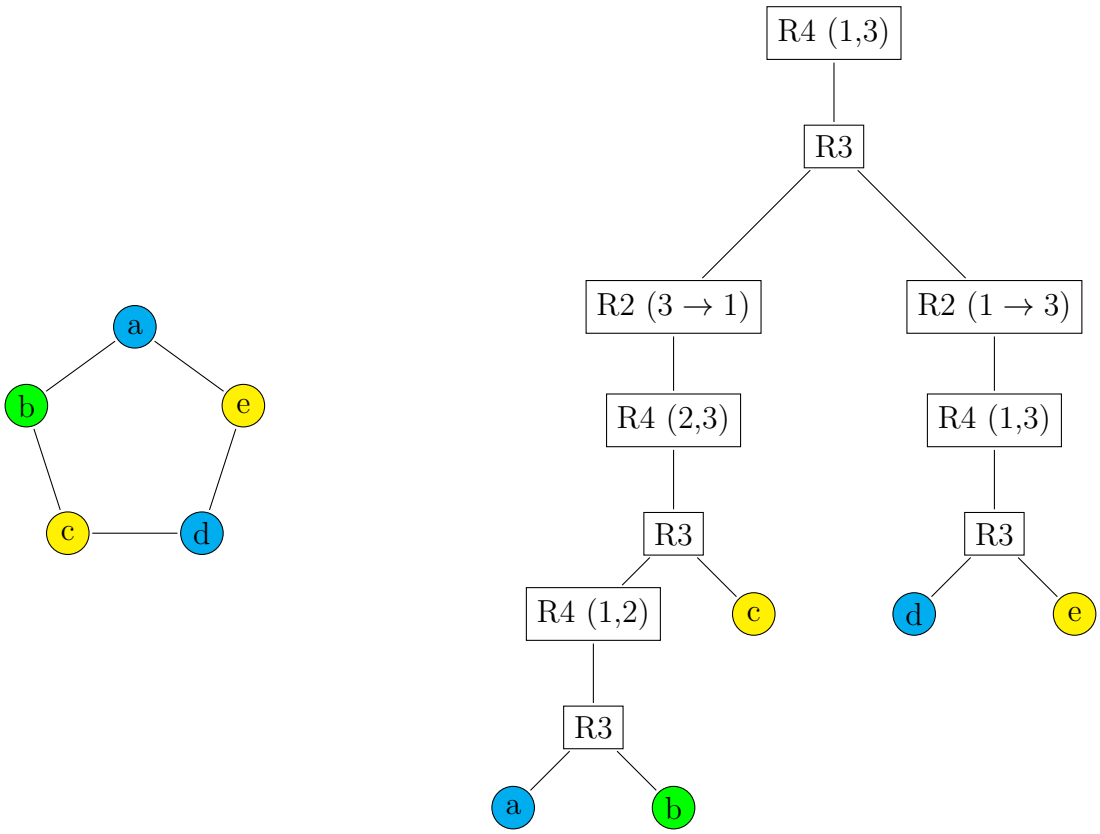

Abbildung 2.3: Auf der linken Seite ist ein $\boldsymbol{C}_5$ dargestellt. Dieser Graph besitzt eine `Clique-Width` von drei: Die rechts dargestellte `Clique-Width`-Expression zugehörig zu dem Graphen muss insgesamt nur drei Farben nutzen, um den Graphen zu rekonstruieren.

  - (R4) Verbinde jeden Knoten der *Farbe* i mit jedem Knoten der *Farbe* j durch eine Kante, wobei $i \neq j$

  Die Berechnung der `Clique-Width` eines Graphen ist **NP**-schwer [Fel+09]. Ein Beispiel zu diesem Parameter kann Abbildung 2.3 entnommen werden.

- `Maximum Clique` Die `Maximum Clique` bezeichnet die größtmögliche Clique in einem Graphen. Die Berechnung der `Maximum Clique` eines Graphen ist **NP**-schwer [Kar72].

- `Boxicity` Die `Boxicity` beschreibt die kleinste Anzahl d an nötigen Dimensionen, um einen Graphen als Schnittgraph achsenparalleler *d-dimensionaler Boxen* darzustellen. Ein Beispiel für die Darstellung eines Graphen mit `Boxicity` zwei kann Abbildung 2.4 entnommen werden. Jede Dimension dieser Darstellung kann außerdem als Intervallgraph angesehen werden. Aus diesen Intervallgraphen lässt sich der ursprüngliche Graph herleiten, indem man eine Kante zwischen zwei Knoten x und y setzt, wenn in jedem Intervallgraphen eine Überschneidung von x und y existiert. Die Berechnung der `Boxicity` eines Graphen ist **NP**-schwer [CFS08].

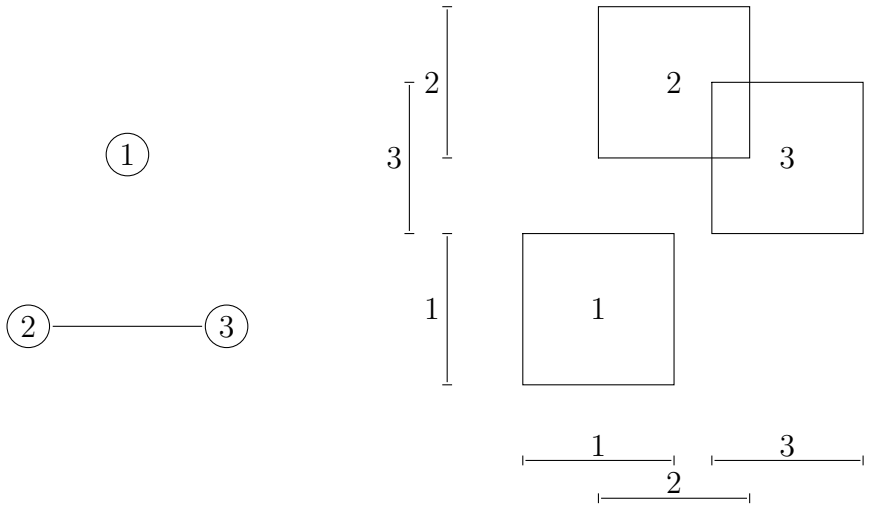

Abbildung 2.4: Ein Graph mit `Boxicity` zwei. Die zweidimensionalen Boxen, welche die Knoten darstellen, überschneiden sich nur, wenn im Graphen eine Kante zwischen den jeweiligen Knoten liegt. Außerdem sind die Intervalle in jeder Dimension dargestellt. Nur wenn in jedem Intervallgraphen eine Überlappung von Intervallen zweier Knoten existiert, so existiert auch eine Kante zwischen den zwei Knoten im Graphen.

- `Chordality` Die `Chordality` eines Graphen ist definiert als die kleinste Zahl k, für welche wir die Kantenmenge E des Graphen als Schnittmenge von k Kantenmengen $E = E_1 \cap ... \cap E_k$ darstellen können, sodass (V, E_i) einen triangulierten Graphen (engl. *Chordal Graph*) bildet. Ein triangulierter Graph ist ein Graph, bei dem jeder induzierte Kreis im Graphen ein Dreieck ist. Die Berechnung der Chordality eines Graphen ist **NP**-schwer [Kos+15].

2.3 ℓ-Grundproblem-Härte

Für die weitere Arbeit an parametrisierten Algorithmen bezüglich Petersen Graph Detection stellt sich die Frage, aus welchen Parametern letztendlich ein Nutzen gezogen werden kann und aus welchen nicht. Letzteres kann durch die *ℓ-Grundproblem-Härte(f)* [Ben+19] gezeigt werden, die wir im Folgenden definieren:

Definition 2.1. [Ben+19] Sei $P \subseteq \Sigma^* \times \mathbb{N}$ ein parametrisiertes Problem und $Q \subseteq \Sigma^*$ das dazugehörige unparametrisierte Problem. Sei $f : \mathbb{N} \to \mathbb{N}$ ein Polynom. Dann gilt für P *ℓ-Grundproblem-Härte(f)* (*ℓ-GP-Härte(f)*), wenn ein Algorithmus A existiert, der jede Eingabeinstanz x von Q in eine Instanz (x', k') von P übersetzt, sodass

(G1) A eine Laufzeit von $O(f(|x|))$ hat,

(G2) $x \in Q \Leftrightarrow (x', k') \in P$,

(G3) $k' \leq \ell$, und

(G4) $|x'| \in O(|x|)$.

Für P gilt *Grundproblem-Härte(f)* (*GP-Härte(f)*), wenn eine natürliche Zahl ℓ existiert, sodass für P *ℓ-GP-Härte(f)* gilt. Wenn f eine lineare Funktion ist, lassen wir die Laufzeit f weg und nennen P *ℓ-GP-hart*.

Wir wenden dieses Konzept zur Veranschaulichung auf das Problem Shortest Path an, welches wie folgt definiert ist:

Shortest Path
Eingabe: Ein ungerichteter Graph G und zwei Knoten s und t.
Frage: Welche Länge besitzt der kürzeste Pfad zwischen s und t?

Wir verwenden hierfür den Parameter **Minimum Degree**. Um bezüglich dieses Parameters *ℓ-GP-Härte* zu zeigen, müssen wir nun den Eingabegraphen so umformen, dass eine Lösung hinsichtlich des Problems nicht verändert wird und der Minimum Degree für alle möglichen Eingabegraphen einen konstanten Wert annimmt. Eine solche Umformung wäre beispielsweise das Hinzufügen eines neuen Knotens x, welchen wir ausschließlich mit dem Knoten s verbinden. Durch diese Umformung wird die Länge des kürzesten Pfades zwischen s und t nicht beeinflusst, allerdings wird der Minimum Degree des Eingabegraphen auf eins gesetzt. Diese Umformung geschieht in konstanter Zeit und der neue Graph hat eine Größe in Abhängigkeit zum Eingabegraphen. Somit sind alle vier Bedingungen für das Vorliegen einer *ℓ-GP-Härte* erfüllt und Shortest Path parametrisiert mit Minimum Degree ist 1-GP-hart. Eine Illustration der Konstruktion kann Abbildung 2.5 entnommen werden.

Durch dieses Werkzeug ist es möglich zu zeigen, dass ein parametrisierter Algorithmus zum Lösen von Petersen Graph Detection für bestimmte Parameter wahrscheinlich **keine** schnellere Laufzeit erzielen kann als der eingangs beschriebene unparametrisierte Algorithmus. Die Zahl ℓ ist eine konstante Obergrenze für die Größe von k'. Somit fällt der von k' abhängige Teil der Worst-Case-Laufzeit eines parametrisierten Problems weg: Die Laufzeit des parametrisierten Problems ist $f(k') \cdot |x|^c \in O(f(\ell) \cdot |x|^c)$ für eine Konstante c. Da $f(\ell)$ ein konstanter Wert ist und konstante Werte für die Worst-Case-Laufzeit keine Rolle spielen, folgt, dass das parametrisierte Problem eine Laufzeit von $O(|x|^c)$ besitzt, welche unabhängig vom gewählten Parameter ist. Alle Parameter, die in Abhängigkeit zum Parameter von P stehen, weisen demzufolge ebenfalls eine *ℓ'-Grundproblem-Härte* für eine natürliche Zahl ℓ' auf.

Im vorigen Absatz wurde bewusst das Wort *wahrscheinlich* verwendet: Die *ℓ-GP-Härte* ist in erster Linie eine Reduktion eines unparametrisierten Problems auf ein parametrisiertes Problem. Somit kann die *ℓ-GP-Härte* auch dafür genutzt werden, mithilfe eines parametrisierten Algorithmus einen besseren unparametrisierten Algorithmus zu

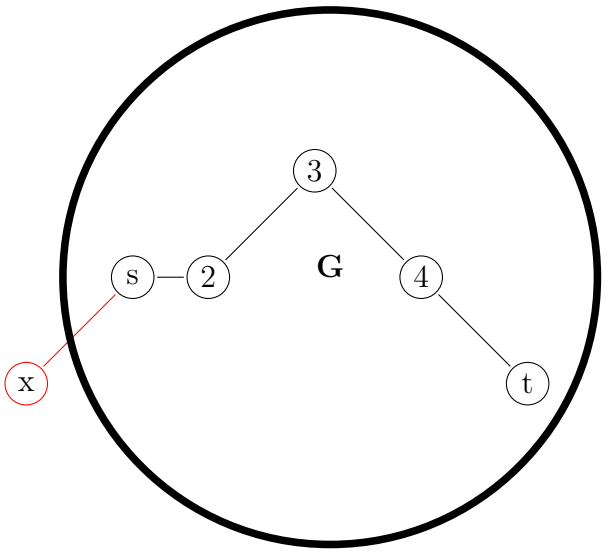

Abbildung 2.5: Der Eingabegraph besitzt für die Knoten s und t einen kürzesten Pfad über die Knoten 2, 3 und 4. Durch die ℓ-GP-Härtereduktion modifizieren wir die Instanz, indem wir einen zusätzlichen Knoten x hinzufügen, und diesen mit dem Knoten s verbinden. Wir stellen somit sicher, dass der Minimum Degree dieses Graphen nun höchstens eins ist.

finden. Für Petersen Graph Detection muss diese Reduktion eine schnellere Laufzeit als $O(n^{10})$ aufweisen, da sonst die Gesamtlaufzeit höher wäre als die unseres bereits beschriebenen Algorithmus. Haben wir nun einen parametrisierten Algorithmus mit einer Laufzeit von $O(f(k') \cdot n^d)$ gefunden, auf welchen wir unseren unparametrisierten Algorithmus in $O(n^c)$ Zeitschritten reduzieren können $(c, d < 10)$, und können wir zeigen, dass $k' \leq \ell$ für eine konstante Zahl ℓ gilt, so folgt ein nun unparametrisierter Algorithmus mit einer Gesamtlaufzeit von $O(n^{\max(c,d)}) \subset O(n^{10})$. Hierfür muss allerdings $n' \in O(n)$ gelten, wobei n' die modifizierte Größe des Eingabegraphen für den parametrisierten Algorithmus darstellt.
Da für Parameter, für die Petersen Graph Detection *GP-hart* ist, vermutlich gilt, dass die schnellsten parametrisierten Algorithmen für Petersen Graph Detection keine Verbesserung gegenüber den schnellsten unparametrisierten Algorithmen für Petersen Graph Detection darstellen, beschäftigen wir uns in dieser Arbeit nicht weiter mit parametrisierten Algorithmen für die Parameter **Minimum Degree**, **Bisection Width** und **Maximum Dominating Set** sowie den Parametern, welche in der Parameter-Hierarchie von den zuvor genannten Parametern von oben beschränkt werden.

3 Parametrisierte Algorithmen

Parameter	c	Worst-Case-Laufzeit
Distance to Clique	8	$O(\|S\|^8 \cdot n^2)$
Vertex Cover	6	$O(\|S\|^6 \cdot n^4)$
Distance to Cograph	4	$O(\|S\|^4 \cdot n^6)$
Distance to Interval	4	$O(\|S\|^4 \cdot n^6)$
Feedback Vertex Set	3	$O(\|S\|^3 \cdot n^7)$
Distance to Bipartite	3	$O(\|S\|^3 \cdot n^7)$
Distance to Perfect	3	$O(\|S\|^3 \cdot n^7)$

Tabelle 3.1: Übersicht über die Parameter und den entsprechenden Laufzeiten von Algorithmus 1. Dabei ist c die Größe der Distanzmenge für die jeweilige Graphklasse im Petersen-Graphen und S ist die Distanzmenge des Eingabegraphen.

In diesem Kapitel betrachten wir nun parametrisierte Algorithmen zu den im vorherigen Kapitel genannten Parametern. Bezogen auf die Graphparameter-Hierarchie behandeln wir die Parameter „von oben nach unten“, sodass wir zunächst funktionierende parametrisierte Algorithmen vorstellen und im Laufe des Kapitels an die Grenzen der Anwendung bestimmter Parameter stoßen werden. Dabei berücksichtigen wir in der Reihenfolge auch die Abhängigkeiten der Parameter zueinander. Wir nehmen an, dass die zu den Parametern zugehörigen Mengen gegeben sind. In den genannten Laufzeiten wird dann nicht auf die Berechnung des jeweiligen Parameters eingegangen. Wir beginnen mit den Distanzparametern.

3.1 Distanz-Parameter

Die Distanzparameter lassen sich jeweils auf die selbe Weise auf das Problem Petersen Graph Detection anwenden. Die Tabelle 3.1 zeigt auf, welchen Einfluss die jeweilige Distanzknotenmenge hat. Im Folgenden gehen wir auf den parametrisierten Algorithmus für Distanzparameter ein.

Als Eingabe erhält der Algorithmus 1 den Eingabegraphen G und die Menge S der Distanzknoten. Die Zahl c stellt hierbei die Mindestanzahl an Knoten dar, die wir aus

Algorithmus 1 für Distanz-Parameter

Input: Ein Graph $G = (V, E)$, die Distanz-Knotenmenge $S \subset V$ und eine Zahl c.
Output: TRUE, wenn G einen Petersen-Graph enthält. FALSE, sonst.

```
function DISTANCE-PARAMETER(G, S, c)
    if |S| < c then
        return FALSE
    end if
    for S' ⊆ S with |S'| = c do
        for V' ⊆ V \ S' with |V'| = 10 − c do
            G' ← G[V' ∪ S']
            if CHECK-GRAPH(G') then        ▷ TRUE, wenn G' ein Petersen-Graph ist
                return TRUE
            end if
        end for
    end for
    return FALSE
end function
```

der Distanzknotenmenge wählen müssen, damit ein Petersen-Graph in dem Eingabegraphen enthalten sein kann: c ist die Größe der Distanzmenge zu einer Graphklasse für den Petersen-Graphen. Ein Graph mit einer Distanzknotenmenge S mit $|S| < c$ kann keinen Petersen-Graphen enthalten. Die Werte, die c für jeden Parameter annimmt, sowie die resultierende Laufzeit kann der Tabelle 3.1 entnommen werden. Die Korrektheit des Algorithmus für alle Distanzparameter unter den gegebenen Werten für c wird nachfolgend erläutert. Desweiteren lässt sich jede Distanzknotenmenge innerhalb des Petersen-Graphen aus Abbildung 3.1 und Abbildung 3.2 entnehmen.

Dieser Algorithmus hat eine Laufzeit von $O(|S|^c \cdot n^{(10-c)})$. Der Algorithmus ist korrekt: Gibt der Algorithmus für einen Eingabegraphen TRUE zurück, so garantiert CHECK-GRAPH, dass die Eingabe einen Petersen-Teilgraphen enthält. Enthält der Eingabegraph einen Petersen-Teilgraphen, so hat die Distanzknotenmenge dieses Graphen zu einer Graphklasse eine Größe von mindestens c. Aus der Distanzknotenmenge werden nun c Knoten geraten, welche Teil des Petersen-Teilgraphen sein müssen, da der Petersen-Graph selbst eine Distanz von c zu dieser Graphklasse aufweist. Die restlichen $10 - c$ Knoten werden aus allen übrigen Knoten geraten. Da der Eingabegraph einen Petersen-Graphen enthält, wird CHECK-GRAPH diesen durch die insgesamt zehn geratenen Knoten erkennen und der Algorithmus wird TRUE zurück geben. Im Folgenden werden wir zeigen, dass der Petersen-Graph für die jeweilige Graphklasse eine Distanzknotenmenge der Größe c aufweist und leiten somit die Laufzeit der parametrisierten Algorithmen her. Wir beginnen mit dem Parameter `Distance to Clique`.

Lemma 3.1. PETERSEN GRAPH DETECTION *parametrisiert mit* ***`Distance to Clique`*** S *kann in* $O(|S|^8 \cdot n^2)$ *Zeit gelöst werden.*

Beweis. Als Clique bezeichnet man einen Graphen, zwischen dessen Knoten paarweise

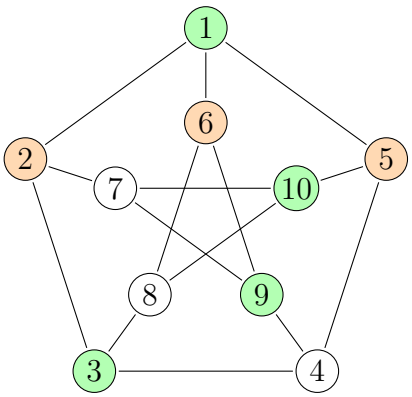

Abbildung 3.1: Farblich markiert sind hier die einzelnen Distanzknotenmengen innerhalb des Petersen-Graphen. Grün = Co-Graph, Orange = Bipartiter und Perfekter Graph

alle Kanten existieren. Der Petersen-Graph ist dreiecksfrei, und so besteht die größte Clique im Petersen-Graphen aus zwei Knoten. Somit ist die „Distance to Clique“ des Petersen-Graphen acht. Enthält der Eingabegraph einen Petersen-Graphen, so erkennt der Algorithmus diesen in $O(|S|^8 \cdot n^2)$ Zeitschritten für diesen Parameter. □

Auf der selben Ebene in der Graphparameter-Hierarchie wie `Distance to Clique` befindet sich der Parameter `Minimum Vertex Cover`, welchen das folgende Lemma behandelt.

Lemma 3.2. Petersen Graph Detection *parametrisiert mit* ***`Minimum Vertex Cover`*** S *kann in* $O(|S|^6 \cdot n^4)$ *Zeit gelöst werden.*

Beweis. Da ein Vertex Cover eines Graphen jede Kante im Graphen abdeckt, werden zur Abdeckung aller Kanten eines Petersen-Graphen mindestens sechs Knoten benötigt: Der Petersen-Graph besteht aus einem äußeren und einem inneren $\boldsymbol{C}_5$, und aus jedem dieser Kreise müssen zur Abdeckung aller Kanten drei Knoten im Vertex Cover liegen. Die Abbildung 3.2 zeigt ein Vertex Cover mit sechs Knoten. Somit besitzt der Petersen-Graph ein `Minimum Vertex Cover` der Größe sechs und es resultiert eine Laufzeit von $O(|S|^6 \cdot n^4)$. □

Als nächstes betrachten wir den Parameter `Distance to Cograph`.

Lemma 3.3. Petersen Graph Detection *parametrisiert mit* ***`Distance to Cograph`*** S *kann in* $O(|S|^4 \cdot n^6)$ *Zeit gelöst werden.*

Beweis. Ein Co-Graph zeichnet sich dadurch aus, dass jeder induzierte Subgraph frei von Pfaden der Länge vier ($\boldsymbol{P}_4$) ist. Um einen Petersen-Graphen frei von induzierten $\boldsymbol{P}_4$ zu machen, und somit zu einem Co-Graphen, müssen 4 Knoten aus dem Petersen-Graphen entfernt werden. Klar ist, dass aus jedem $\boldsymbol{C}_5$ des Petersen-Graphen mindestens zwei Knoten entfernt werden müssen, da sonst ein $\boldsymbol{P}_4$ aus den übrigen Knoten des $\boldsymbol{C}_5$ induziert werden kann. Für den äußeren und inneren $\boldsymbol{C}_5$ des Petersen-Graphen kommen wir

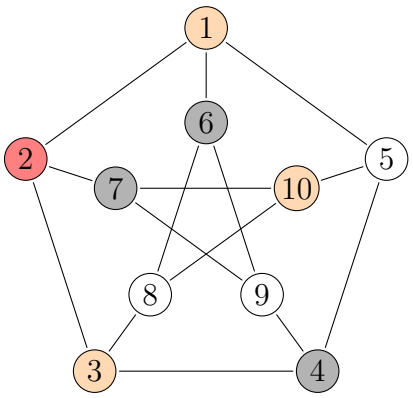

Abbildung 3.2: Die orange gefärbten Knoten bilden das `Feedback Vertex Set`. Die orange und grau gefärbten Knoten bilden zusammen ein `Minimum Vertex Cover` für den Petersen-Graphen. Die orange und rot gefärbten Knoten bilden zusammen die Distanzmenge zum Intervallgraphen für den Petersen-Graphen.

somit auf eine Mindestanzahl von vier zu entfernenden Knoten. Eine gültige Auswahl von zu entfernenden Knoten kann Abbildung 3.1 entnommen werden. Enthält G einen Petersen-Graphen, so erkennt Algorithmus 1 diesen in $O(|S|^4 \cdot n^6)$ Zeitschritten. □

Der Parameter `Distance to Cograph` wird von den beiden vorangegangenen Parametern in der Größe von oben beschränkt. Dies gilt auch für den nun folgenden Parameter `Distance to Interval`.

Lemma 3.4. Petersen Graph Detection *parametrisiert mit* ***`Distance to Interval`*** S *kann in* $O(|S|^4 \cdot n^6)$ *Zeit gelöst werden.*

Beweis. Ein Intervallgraph hat unter anderem die Eigenschaft, dass dieser frei von $\boldsymbol{C}_{\geq 4}$ ist. Da alle Kreise im Petersen-Graphen eine Länge von mindestens fünf haben, müssen also alle Kreise des Petersen-Graphen durch die Distanzknotenmenge abgedeckt werden. Demzufolge ist die Distanzknotenmenge mindestens so groß wie das `Feedback Vertex Set` des Petersen-Graphen. Allerdings kann man zeigen, dass sämtliche Kombinationen von drei zu entfernenden Knoten keinen Restgraphen des Petersen-Graphen bilden, welcher ein Intervallgraph ist. Beispielsweise bildet der Restgraph des `Feedback Vertex Sets` aus Abbildung 3.2 keinen Intervallgraphen, da das Intervall für den Knoten 9 zwangsläufig mit einem Knoten überlappen wird, mit welchem der Knoten 9 nicht verbunden ist. Durch das Entfernen eines weiteren Knotens kann allerdings aus dem Restgraphen ein Intervallgraph geformt werden. Die Distanzknotenmenge hat also Größe vier und es folgt eine Laufzeit von $O(|S|^4 \cdot n^6)$. □

Das nachfolgende Lemma ist für die kommenden Beweise nützlich.

Lemma 3.5. *Entfernt man zwei Knoten aus dem Petersen-Graphen, so enthält der Restgraph mindestens einen Kreis der Länge fünf.*

Beweis. Nehmen wir an, durch das Entfernen von zwei Knoten im Petersen-Graphen würde es im Restgraphen keinen $\boldsymbol{C}_5$ mehr geben. Insbesondere müsste ein Knoten aus dem äußeren $\boldsymbol{C}_5$ und ein Knoten aus dem inneren $\boldsymbol{C}_5$ des Petersen-Graphen entfernt werden. Aus Symmetriegründen legen wir ohne Beschränkung der Allgemeinheit für den weiteren Beweis fest, dass der Knoten 1 (siehe Abbildung 3.1) in der Distanzknotenmenge liegt. Es folgen nun drei verschiedene Möglichkeiten, wie der zweite Knoten der Distanzknotenmenge gewählt werden kann.

- Wählen wir den Knoten 6, so verbleibt ein $\boldsymbol{C}_5$ aus den Knoten 2, 7, 9, 4, 3.
- Wählen wir den Knoten 7, so verbleibt ein $\boldsymbol{C}_5$ aus den Knoten 3, 8, 10, 5, 4. Symmetrisch dazu ist die Wahl des Knotens 10.
- Wählen wir den Knoten 8, so verbleibt ein $\boldsymbol{C}_5$ aus den Knoten 2, 7, 9, 4, 3. Symmetrisch dazu ist die Wahl des Knotens 9.

Es folgt daraus, dass unabhängig von der Wahl der zwei Knoten immer ein $\boldsymbol{C}_5$ im Petersen-Graph zu finden ist. □

Diese Eigenschaft findet nun im folgenden Lemma zur Laufzeit des `Feedback Vertex Sets` Gebrauch.

Lemma 3.6. Petersen Graph Detection *parametrisiert mit* ***`Feedback Vertex Set`*** S *kann in* $O(|S|^3 \cdot n^7)$ *Zeit gelöst werden.*

Beweis. Das `Feedback Vertex Set` bezeichnet die Menge der Knoten, die entfernt werden müssen, um einen Graphen kreisfrei zu machen. Dafür müssen mindestens drei Knoten aus dem Petersen-Graphen entfernt werden, da sonst laut Lemma 3.5 in jedem Falle ein Kreis im Restgraphen vorhanden ist. In Abbildung 3.2 ist ein `Feedback Vertex Set` der Größe drei für den Petersen-Graphen abgebildet. Es folgt also eine Laufzeit von $O(|S|^3 \cdot n^7)$. □

Zuletzt betrachten wir die Parameter `Distance to Bipartite` und `Distance to Perfect` im nachfolgenden Lemma.

Lemma 3.7. Petersen Graph Detection *parametrisiert mit* ***`Distance to Bipartite`*** S *oder* ***`Distance to Perfect`*** S *kann in* $O(|S|^3 \cdot n^7)$ *Zeit gelöst werden.*

Beweis. Die Knotenmenge eines bipartiten Graphen lässt sich in zwei Partitionen aufteilen, sodass nur zwischen den Partitionen Kanten existieren und innerhalb der Partitionen nicht. So sind Kreise mit gerader Knotenanzahl bipartite Graphen, Kreise ungerader Länge jedoch nicht. Um aus dem Petersen-Graphen einen bipartiten Graphen zu formen, müssen also Knoten aus den Kreisen ungerader Länge entfernt werden, sodass die restlichen Knoten einen bipartiten Graphen bilden können. Aus dem Petersen-Graphen müssen insgesamt drei Knoten entfernt werden, damit der Restgraph bipartit ist, da sonst laut Lemma 3.5 mindestens ein $\boldsymbol{C}_5$ im Restgraphen vorhanden ist. Eine solche Distanzknotenmenge ist in Abbildung 3.1 dargestellt.

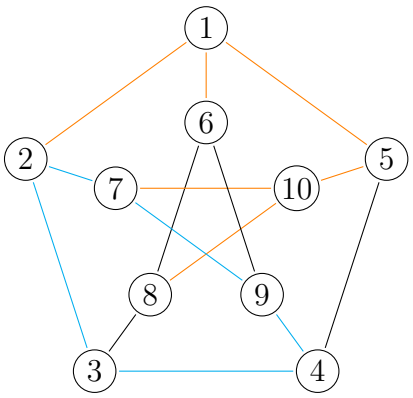

Abbildung 3.3

Abbildung 3.4: Aus den orange markierten Kanten lassen sich höchstens zwei nicht-adjazente Kanten wählen. Außerdem stellt diese Wahl an Kanten kein Feedback Edge Set dar: Die türkis markierten Kanten bilden einen Kreis.

Ein Graph ist perfekt, wenn jeder induzierte Subgraph k-färbbar ist, wobei k die Größe der größten Clique im Subgraphen darstellt. Die größte Clique des Petersen-Graphen hat die Größe zwei, der Petersen-Graph ist aber dreifärbbar. Es müssen also nun Knoten entfernt werden, sodass der resultierende Restgraph zweifärbbar wird. Eine Zweifärbung eines Graphen lässt sich als eine Bipartitionierung der Knoten des Graphen sehen, sodass Knoten einer Partition untereinander keine Kanten besitzen: Die Distanzknotenmenge zum perfekten Graphen ist im Petersen-Graphen also identisch zur Distanzknotenmenge zum bipartiten Graphen. Für beide Parameter ergibt sich somit eine Laufzeit von $O(|S|^3 \cdot n^7)$ Zeitschritten. □

3.2 Andere Parameter

In diesem Unterkapitel beschäftigen wir uns mit allen weiteren Parametern, für die wir schnelle parametrisierte Algorithmen gefunden haben. Wir beginnen mit dem Parameter `Feedback Edge Set`.

Das `Feedback Edge Set` beschreibt eine Menge von Kanten, welche entfernt werden muss, damit ein Graph kreisfrei wird. Demzufolge steht das `Feedback Edge Set` im direkten Zusammenhang mit dem `Feedback Vertex Set` aus dem vorigen Unterkapitel. Ein parametrisierter Algorithmus zu Petersen Graph Detection mit dem Parameter `Feedback Edge Set` arbeitet im Wesentlichen mit demselben Prinzip wie die distanzparametrisierten Algorithmen, jedoch nutzt der Algorithmus Kantenmengen statt Knotenmengen.

Lemma 3.8. Petersen Graph Detection *parametrisiert mit* ***`Feedback Edge Set`*** S *kann in* $O(|S|^3 \cdot m^2)$ *Zeit gelöst werden.*

Beweis. Damit ein Eingabegraph einen Petersen-Graphen enthalten kann, muss dieser ein `Feedback Edge Set` S der Größe mindestens sechs haben. Dies muss gelten, da der Restgraph beim Entfernen des `Feedback Edge Sets` einen Baum darstellt, und Bäume mit zehn Knoten neun Kanten besitzen. Da der Petersen-Graph insgesamt 15 Kanten besitzt, folgt ein Feedback Edge Set der Größe sechs für den Petersen-Graphen.

Ähnlich zum unparametrisierten Algorithmus raten wir nun drei zueinander nicht inzidente Kanten aus dem `Feedback Edge Set` und raten anschließend aus allen übrigen Kanten zwei weitere Kanten, welche nicht inzident zu den vorher geratenen Knoten sind. Diese Kanten kann man nun mit CHECK-GRAPH überprüfen. Es folgt eine Laufzeit von $O(|S|^3 \cdot m^2)$.

Es gilt nun zu zeigen, dass es immer möglich ist, drei solcher Kanten aus S zu wählen, wenn der Eingabegraph einen Petersen-Graphen enthält: Nehmen wir an, es ist nicht immer möglich, drei zueinander nicht inzidente Kanten aus dem `Feedback Edge Set` zu wählen. Dann gibt es für den Petersen-Graphen ein `Feedback Edge Set`, welches für zwei Knoten des Petersen-Graphen alle adjazenten Kanten beinhaltet. Außerdem sind diese Knoten nicht miteinander benachbart. Es folgt also eine Menge von sechs Kanten, aus welchen man nun höchstens zwei zueinander nicht inzidente Kanten wählen kann. Man kann nun allerdings zeigen, dass diese Kantenmenge kein `Feedback Edge Set` bildet, ein Beispiel dazu ist in Abbildung 3.4 abgebildet. □

Als Nächstes behandeln wir den Parameter `Maximum Degree`. Der `Maximum Degree` Δ bezeichnet den größten Knotengrad innerhalb eines Graphen. Mithilfe von Δ lässt sich eine Laufzeitabschätzung bezüglich des Findens eines induzierten Petersen-Teilgraphen machen.

Lemma 3.9. PETERSEN GRAPH DETECTION *parametrisiert mit* `Maximum Degree` Δ *kann in* $O(\Delta^9 \cdot n)$ *Zeit gelöst werden.*

Beweis. Ein mit `Maximum Degree` parametrisierter Algorithmus arbeitet mit einem Suchbaum, welcher zehn zusammenhängende Knoten rät und diese dann mittels CHECK-GRAPH darauf überprüft, ob diese einen Petersen-Graphen bilden. Zunächst wird ein Startknoten aus der Knotenmenge geraten. Anschließend wird ein Nachbarknoten dieses Knotens geraten. Für einen Knoten gibt es hierfür maximal Δ viele Möglichkeiten. Nachfolgend raten wir einen Nachbarn des zuletzt geratenen Knotens, welcher bisher noch nicht geraten wurde. Dieser Prozess wird solange durchgeführt, bis insgesamt zehn verschiedene Knoten geraten wurden. Da der Petersen-Graph einen Hamilton-Pfad enthält, ist es auf jeden Fall möglich, eine solche Folge von Knoten zu raten. Beispielsweise bildet die folgende Knotenfolge einen Hamilton-Pfad: 1, 2, 3, 4, 5, 10, 8, 6, 9, 7.

Der beschriebene Algorithmus ist korrekt: Enthält der Eingabegraph einen Petersen-Graphen, so wird dieser durch den Algorithmus gefunden, da der Petersen-Graph einen Hamilton-Pfad enthält, und mit CHECK-GRAPH überprüft. Durch die Worst-Case-Laufzeit des Suchbaums ergibt sich eine Laufzeit von $O(\Delta^9 \cdot n)$. □

Es folgt nun der Parameter `Degeneracy`.

Lemma 3.10. PETERSEN GRAPH DETECTION *parametrisiert mit* `Degeneracy` d *kann in* $O(d^6 \cdot n^4)$ *gelöst werden.*

Beweis. Gegeben eine **degeneracy order** des Eingabegraphen, also eine Ordnung sodass $|N(v) \cap \{w | v \leq_d w\}| \leq d$, raten wir vier Knoten aus dieser Ordnung, sodass diese vier Knoten nicht miteinander benachbart sind. Für jeden dieser Knoten gilt, dass diese jeweils höchstens d viele Nachbarn besitzen, welche in der Ordnung nach diesen Knoten folgen. Aus diesen insgesamt $4 \cdot d$ vielen Nachbarn raten wir nun sechs weitere Knoten und überprüfen dann mittels CHECK-GRAPH, ob diese zehn Knoten einen Petersen-Graphen darstellen. Es ergibt sich somit eine Gesamtlaufzeit von $O(d^6 \cdot n^4)$. □

Wir fahren fort mit dem Parameter `Average Degree`.

Lemma 3.11. PETERSEN GRAPH DETECTION *parametrisiert mit* `Average Degree` a *kann in* $O(a^5 \cdot n^5)$ *gelöst werden.*

Beweis. Der `Average Degree` eines Graphen lässt sich in konstanter Zeit durch $a = 2m/n$ berechnen. Der unparametrisierte Algorithmus besitzt eine Laufzeit von $O(n^{10})$ oder $O(m^5)$. Formt man die Gleichung zur Berechnung des `Average Degrees` eines Graphen um und wendet diese dann auf die Worst-Case-Laufzeit des unparametrisierten Problems an, erhalten wir eine Laufzeit von $O((\frac{a \cdot n}{2})^5) \in O(a^5 \cdot n^5)$. Es folgt somit die Existenz eines parametrisierten Algorithmus für diesen Parameter. □

Als nächstes betrachten wir den Parameter `Clique-Width`.

Lemma 3.12. PETERSEN GRAPH DETECTION *parametrisiert mit* `Clique-Width` k *kann in* $O(10^k \cdot n)$ *gelöst werden.*

Beweis. Wir nehmen hierbei an, dass wir die `Clique-Width` k sowie eine `Clique-Width`-Expression für den Graphen gegeben haben. Wir werden nun ein dynamisches Programm aufstellen, das mithilfe dieser `Clique-Width`-Expression und den vier Operationen zugehörig zur `Clique-Width` aus den einzelnen Knoten nach und nach den Eingabegraphen rekonstruiert. Dabei notieren wir in einer Tabelle, welche Subgraphen des Petersen-Graphen wir während des Rekonstruktionsprozesses finden. Insbesondere zählt der Petersen-Graph selbst als sein eigener Subgraph. Im finalen Schritt des Algorithmus haben wir den Eingabegraphen wieder rekonstruiert und können durch die Tabelleneinträge ablesen, ob im Eingabegraphen ein Petersen-Graph vorhanden ist. Zunächst definieren wir die Grundlagen des Algorithmus.

Zur besseren Verständnis nennen wir die Knoten des Eingabegraphen „Knoten“ und die Knoten der `Clique-Width`-Expression, welche Knoten enthalten kann, „Ecken“. Jede Ecke besitzt eine Tabelle, eine Knotenmenge und eine Kantenmenge. Jede Tabelle enthält einen Eintrag für jeden Subgraphen des Petersen-Graphen in jeder möglichen Färbung. Die Anzahl möglicher Farben wird durch den Parameter k bestimmt. Jede Tabelle hat somit $O(10^k)$ viele Einträge. Jeder Eintrag der Tabelle ist initial auf „0“ gesetzt. Wenn in unserer aktuellen Ecke einer dieser Subgraphen mit einer bestimmten Färbung vorhanden ist, so füllen wir den jeweiligen Eintrag mit einer „1“.

Wir beschreiben nun, wie die vier Operationen unsere Knoten, Ecken und Tabellen beeinflussen.

- **(R1) Erstelle einen neuen Knoten v mit Farbe i:** Diese Regel wählt einen Knoten aus der Knotenmenge des Graphen, welcher bisher nicht innerhalb einer Ecke vorhanden ist, und weist diesem eine Farbe zu. Dieser gefärbte Knoten wird nun einer neuen Ecke zugewiesen, welche keine anderen Knoten enthält. Die Tabelle dieser Ecke wird entsprechend der Färbung des Knotens gefüllt.

- **(R2) Ersetze die Farbe i durch Farbe j:** Diese Regel färbt jeden Knoten einer Ecke, welche die Farbe i besitzt, mit der Farbe j. Dementsprechend müssen nun alle Einträge in der Tabelle, welche Knoten der Farbe i enthalten, erneuert werden: Die Einträge mit Farbe i werden auf 0 gesetzt und die entsprechenden Einträge mit der Farbe j werden auf 1 gesetzt. Die Gesamtanzahl der Subgrapheneinträge mit einer 1 in der Tabelle erhöht sich hierbei nicht, kann jedoch sinken, wenn zuvor Einträge mit der Farbe j existiert haben, die äquivalent zu zuvor vorhandenen Einträgen mit der Farbe i sind.

- **(R3) Führe eine disjunkte Vereinigung zweier unterschiedlich gefärbter Graphen durch:** Diese Regel verbindet zwei Ecken miteinander: Die Knotenmengen und Kantenmengen werden jeweils vereinigt, und eine neue Tabelle wird aus den zwei Tabellen der Ecken erstellt. Jeder Eintrag, der in einer der beiden Ecken vorhanden ist, wird übernommen. Außerdem wird jeder Eintrag eines Subgraphen, welcher durch eine Kombination von Knoten aus beiden Knotenmengen der vereinigten Ecken entstehen kann, auf 1 gesetzt. Solche Subgraphen haben mindestens zwei Zusammenhangskomponenten.

- **(R4) Verbinde jeden Knoten der Farbe i mit jedem Knoten der Farbe j durch eine Kante, wobei $i \neq j$:** Diese Regel führt neue Kanten zwischen Knoten zweier verschiedener Farben in die Ecke ein. Demzufolge bleibt die Knotenmenge gleich und die Kantenmenge wird passend ergänzt. Wichtig ist nun, die Tabelleneinträge zu aktualisieren. Durch das Hinzufügen von neuen Kanten kann es sein, dass vorher existierende Subgraphen in bestimmten Färbungen nun nicht mehr in dieser Form existieren. Diese Einträge müssen entfernt werden. Außerdem müssen Einträge für Subgraphen, welche durch die neuen Kanten entstehen, hinzugefügt werden.

Nachdem diese Grundlagen erklärt sind, können wir nun den Algorithmus beschreiben.

Wir bearbeiten die `Clique-Width`-Expression „von unten nach oben“. Zunächst befinden sich alle Knoten einzeln gefärbt in einer Ecke. Demzufolge ist die Regel **R1** bereits angewendet worden. Die Anwendung dieser Regel benötigt n Zeitschritte.

Anschließend werden die Regeln **R2**, **R3** und **R4** solange angewandt, bis der Algorithmus zur Wurzelecke der `Clique-Width`-Expression angelangt ist. Das Zusammenführen oder Umfärben der Ecken geschieht in konstanter Zeit, und das Aktualisieren der Tabelleneinträge braucht maximal $O(10^k)$ Zeitschritte, konstant viele für jeden Eintrag.

In der Wurzelecke besitzen wir nun eine finale Tabelle, welche alle vorhandenen Subgraphen des Petersen-Graphen enthält. Gibt es einen Eintrag für den Petersen-Graphen

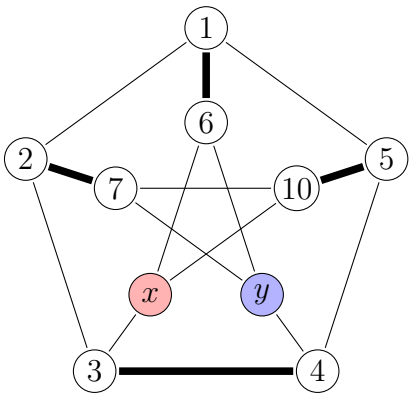

Abbildung 3.5: Knoten x ist mit den Knoten 3, 6 und 10 verbunden. Knoten y ist mit 4, 6 und 7 verbunden.

in irgendeiner Färbung, so wissen wir, dass ein Petersen-Graph Teil des Eingabegraphen ist. Eine `Clique-Width`-Expression enthält höchstens $O(n)$ viele Ecken, sodass das Programm insgesamt in einer Laufzeit von $O(10^k \cdot n)$ resultiert. □

Als nächstes betrachten wir den Parameter `Boxicity`.

Lemma 3.13. Petersen Graph Detection *parametrisiert mit* ***`Boxicity`*** k *kann in* $O(m^4 \cdot n \cdot k)$ *Zeit gelöst werden.*

Beweis. Für den folgenden Algorithmus nehmen wir an, dass wir die `Boxicity` in Form von k Intervallgraphen gegeben haben.

Zunächst raten wir vier Kanten aus der Kantenmenge so, dass diese Kanten ein Matching für acht Knoten bilden. Für die Wahl der Kanten muss außerdem gelten, dass die übrigen zwei Knoten, welche wir zum Erraten eines Petersen-Graphen noch benötigen, nicht mit einer Kante verbunden sind. Ein Beispiel hierfür bietet die Abbildung 3.5: Das Matching induziert den dargestellten Graphen **ohne** die Knoten x und y und x und y dürfen nicht miteinander verbunden sein. Die Knoten x und y besitzen jeweils drei Kanten zu je drei Knoten des Matchings, und besitzen insbesondere keine Kanten zu den restlichen dargestellten Knoten des Matchings.

Wir weisen nun alle Knoten, welche dieselben Kanten zum Matching besitzen wie der Knoten x aus der Abbildung, der Menge X zu. Alle Knoten, welche dieselben Kanten zum Matching besitzen wie der Knoten y, werden der Menge Y zugewiesen. Die übrigen Knoten sind für die weitere Bearbeitung nicht mehr relevant. Wir suchen nun in den Intervallgraphen ein Paar aus Knoten, wobei ein Knoten aus der Menge X und ein Knoten aus der Menge Y stammt. Dieses Paar ist nicht miteinander verbunden. Um so ein Paar zu finden, suchen wir in den Intervallgraphen der `Boxicity` nach einem Paar, für welche keine Überlappung der Kanten vorliegt und somit im Eingabegraphen keine Kante zwischen den beiden Knoten existiert (siehe Abbildung 3.6). Wenn es so ein Paar

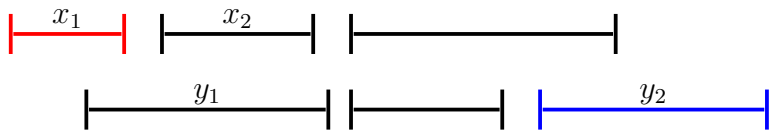

Abbildung 3.6: Ein Ausschnitt aus einem Intervallgraphen der `Boxicity` eines möglichen Eingabegraphen. Für den Algorithmus suchen wir ein Paar von Intervallen aus X und Y, sodass diese keine Überschneidung besitzen. Die unbenannten Intervalle stehen für Knoten, welche weder zu der Menge X noch der Menge Y gehören. Die farblich markierten Intervalle würden so ein Paar darstellen. Gibt es so ein Paar in einem der Intervallgraphen der `Boxicity`, dann sind diese zwei Knoten Teil eines Petersen-Graphen.

in einem der Intervallgraphen gibt, dann gibt es zwischen diesen beiden Knoten im ursprünglichen Graphen keine Kante. Diese würde nur existieren, wenn die beiden Knoten in jedem Intervallgraphen eine Überlappung hätten.

Für jeden der k Intervallgraphen iterieren wir über jedes Intervall. Jedes Intervall besitzt einen Start- und einen Endzeitpunkt und die Intervalle des Intervallgraphen sind nach Endzeitpunkt aufsteigend sortiert. Wenn wir das erste Mal ein Intervall zugehörig zu einem Knoten der Menge X finden, suchen wir nun ein Intervall zugehörig zur Menge Y, welches startet, nachdem das Intervall aus der Menge X beendet ist. Gibt es so ein Intervall nicht, so schauen wir uns das frühstendende Intervall aus der Menge Y an und wiederholen das Prozedere mit den Intervallen aus X. Gibt es für jeden Intervallgraphen kein Paar, welches nicht überlappt, dann gibt es auch keine Knoten, welche wie in der Abbildung 3.5 dargestellt mit dem Matching einen Petersen-Graphen bilden können. Ist das für jedes Matching der Fall, dann gibt es keinen Petersen-Graphen in dem Eingabegraphen. Insgesamt resultiert eine Worst-Case-Laufzeit von $O(m^4 \cdot (n + k \cdot n))$ für diesen Algorithmus, also eine Laufzeit von $O(m^4 \cdot n \cdot k)$. □

4 Härteresultate

In diesem Kapitel widmen wir uns Parametern, für die *ℓ-Grundproblem-Härte* gezeigt werden kann, welche wir in Kapitel 2 definiert haben. Zu zeigen ist also, dass wir in einer Laufzeit polynomiell zur Größe unseres Eingabegraphen, allerdings von oben beschränkt durch die Laufzeit des naiven Algorithmus, diesen so umformen können, dass der Parameter einen konstanten Wert annimmt und unsere Umformung die Korrektheit der Eingabe bezüglich Petersen Graph Detection nicht beeinflusst. Desweiteren muss die Größe des modifizierten Eingabegraphen in linearer Abhängigkeit zur ursprünglichen Größe stehen (siehe Definition 2.1). Wir betrachten zunächt den Parameter `Minimum Degree`.

Lemma 4.1. Petersen Graph Detection *parametrisiert mit* `Minimum Degree` *ist 0-GP-hart.*

Beweis. Im Falle des Parameters `Minimum Degree` lässt sich die *GP-Härte* einfach zeigen, da das bloße Hinzufügen eines Knotens mit Knotengrad null bereits den Parameterwert für alle möglichen Eingabegraphen konstant auf null setzt. Da der Knoten mit dem Eingabegraphen nicht verbunden ist und selbst auch keinen Petersen-Graphen darstellt, beeinflusst dieser Knoten auch nicht die Lösung bezüglich Petersen Graph Detection. Klar ist, dass das Hinzufügen des Knotens in konstanter Zeit geschieht und die Größe des neuen Graphen durch $O(n)$ abgeschätzt werden kann. Somit ist gezeigt, dass Petersen Graph Detection parametrisiert mit `Minimum Degree` 0-*GP-hart* ist. □

Die Parameter `Domatic Number` sowie `Distance to Disconnected` sind somit ebenfalls irrelevant für eine Laufzeitverbesserung unseres Problems, da diese in der Graphparameter-Hierarchie durch `Minimum Degree` von oben beschränkt sind.

Bezüglich des Parameters `Bisection Width` lässt sich ebenfalls *GP-Härte* zeigen. Zur Erinnerung: Die `Bisection Width` beschreibt die kleinstmögliche Anzahl an Kanten zwischen zwei gleich großen (oder bei ungerader Knotenanzahl fast gleich großen) Partitionen eines Graphen.

Lemma 4.2. Petersen Graph Detection *parametrisiert mit* `Bisection Width` *ist 0-GP-hart.*

Beweis. Unser Algorithmus erstellt für jeden Eingabegraphen eine Kopie des Graphen und fügt diese als neue Komponente(n) dem Graphen hinzu. Somit lässt sich der modifizierte Graph G' in zwei gleich große Partitionen teilen, zwischen denen keine Kanten existieren. Die *Width* zwischen diesen beiden Partitionen ist dann null, und da diese die geringste *Width* aller Bipartitionen des Graphen darstellt, ist auch die `Bisection`

`Width` dementsprechend null. Die Größe der neuen Instanz ist doppelt so groß wie die der Eingabeinstanz und steht somit in linearer Abhängigkeit zur Größe des Eingabegraphen. Das Erstellen einer Kopie des Graphen geschieht auch in linearer Zeit zur Eingabegröße. Da in G' die Kopie des Eingabegraphen keine Kanten zum Eingabegraphen selbst besitzt und somit eine unabhängige Komponente im Graphen darstellt, beeinflusst diese Komponente eine Aussage bezüglich Petersen Graph Detection nicht: Es existiert nur ein Petersen-Graph in G', genau dann, wenn im Eingabegraphen einer existiert. Somit gilt für Petersen Graph Detection parametrisiert mit `Bisection Width` 0-*GP-Härte*. □

Als nächstes befassen wir uns mit dem Parameter `Minimum Dominating Set`.

Lemma 4.3. Petersen Graph Detection *parametrisiert mit* ***`Minimum Dominating Set`*** *ist 1-GP-Hart.*

Beweis. Um *1-GP-Härte* zu zeigen, fügen wir dem Eingabegraphen einen Knoten hinzu, welcher eine Kante zu jedem anderen Knoten im Eingabegraphen besitzt. Ein `Minimum Dominating Set` des Graphen würde nun aus diesem neu hinzugefügten Knoten bestehen. Der Parameter kann somit konstant auf eins gesetzt werden. Die Größe des modifizierten Graphen steht in linearer Abhängigkeit zur Größe des Eingabegraphen. Das Hinzufügen dieses Knoten benötigt konstante Zeit, ebenso wie das Verbinden dieses Knotens mit allen anderen Knoten. Es gilt nun zu zeigen, dass der neu eingefügte Knoten v nicht Teil eines Petersen-Graphen sein kann. Da v Kanten zu jedem Knoten jeder möglichen Knotenteilmenge der Größe zehn besitzt und dies für keinen Knoten des Petersen-Graphen der Fall ist, ist somit auch ausgeschlossen, dass v Teil eines induzierten Petersen-Graphen sein kann. Somit beeinflusst das Hinzufügen des Knotens v auf die vorher beschriebene Weise die Lösung bezüglich Petersen Graph Detection nicht. Petersen Graph Detection parametrisiert mit `Minimum Dominating Set` ist somit *1-GP-Hart*. □

Damit einher erübrigt sich eine nähere Betrachtung der vom Parameter `Minimum Dominating Set` von oben beschränkten Parameter `Maximum Diameter of Components`, `Average Distance` und `Girth`.

5 Fazit

Innerhalb dieser Arbeit war es uns möglich, nahezu die gesamte Graphparameter-Hierarchie im Bezug auf das Problem Petersen Graph Detection zu untersuchen. Wir konnten für die Distanzparameter einen universellen parametrisierten Algorithmus beschreiben. Hierbei nahmen wir die Distanzknotenmengen als gegeben an. Jedoch ist die Nützlichkeit dieser Parameter im Einzelfall zu prüfen, denn alle bearbeiteten Distanzparameter können im Allgemeinen nicht in polynomieller Zeit berechnet werden. Es kann hierbei aber auf Konstantfaktor-Approximationen der Parameter zurückgegriffen werden: Die approximierten Parameter sind um einen konstanten Faktor „schlechter“ als die optimale Größe des Parameters. Ein Beispiel hierfür ist die mögliche Approximation des Parameters `Minimum Vertex Cover`, welche in Kapitel 2 angesprochen wird. Solche Approximationen haben keinen Einfluss auf die asymptotische Worst-Case-Laufzeit unserer Algorithmen: Sei $k' = c \cdot k$ ein approximierter Parameterwert, wobei c eine konstante Zahl ist. Dann gilt $O(k'^a \cdot n^b) = O(k^a \cdot n^b)$ für zwei Zahlen a und b.

Während der Bearbeitung haben wir festgestellt, dass der Petersen-Graph zu einigen Graphklassen eine gewisse Distanz aufweist. Wir konnten also implizit zeigen, dass zum Beispiel die Klasse der perfekten Graphen den Petersen-Graphen nicht enthält. Auch wissen wir, dass der Petersen-Graph nicht planar ist. Betrachtet man also für das Problem Petersen Graph Detection nur Graphen dieser Graphklassen, so kann man das Problem trivialer Weise in konstanter Zeit mit „Nein“ beantworten. Demzufolge wäre weitere Forschung zur Anwendung des Problems auf Graphklassen, welche nicht in dieser Arbeit behandelt wurden, möglich. Beispielsweise könnte man das Problem für dreiecksfreie Graphen oder reguläre Graphen, also Graphen für welche der `Maximum Degree` mit dem `Minimum Degree` übereinstimmt, untersuchen. Da der Petersen-Graph kein planarer Graph, aber ein biplanarer Graph (ein Graph bestehend aus der Kantenvereinigung zweier planarer Graphen) ist, wäre auch diese Graphklasse für eine solche Untersuchung interessant.

In dieser Arbeit betrachteten wir induzierte Petersen-Graphen. Nun mag man sich die Frage stellen, ob die Erkennung von nicht-induzierten Petersen-Teilgraphen, und somit die Betrachtung einer verallgemeinerten Variante unseres Problems Petersen Graph Detection, interessant sein könnte. Unserer Ansicht nach ist dieses Problem allerdings eher uninteressant, da beispielsweise eine Clique der Größe zehn innerhalb eines Eingabegraphen einen nicht-induzierten Petersen-Graphen enthalten würde. Eine Clique besitzt allerdings andere Eigenschaften als der Petersen-Graph, sodass die Erkennung dieses nicht-induzierten Petersen-Graphen nicht relevant zu sein scheint.

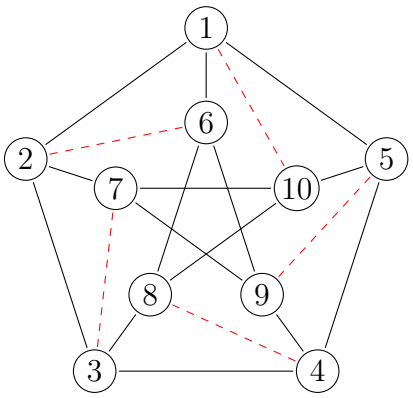

Abbildung 5.1: Die rot gestrichelten Linien stellen ein perfektes Matching der Knoten des Petersen-Graphen aus Kanten dar, die nicht im Petersen-Graphen vorhanden sind. Demzufolge existieren diese Kanten aber im Komplement des Petersen-Graphen, sodass diese Kanten ein Matching in dem Komplement bilden.

Für Petersen Graph Detection parametrisiert mit `Chromatic Number` haben wir keine Aussage treffen können. Es war uns weder möglich, einen Algorithmus aufzustellen, welcher unter Umständen schneller ist als der unparametrisierte Algorithmus, noch war es uns möglich, *ℓ-GP-Härte* für dieses Problem zu zeigen. Wir betrachteten außerdem das komplementäre Problem zu Petersen Graph Detection.

$\overline{PGD}$
Eingabe: Ein ungerichteter Graph G.
Frage: Enthält G das Komplement des Petersen-Graphen als induzierten Teilgraphen?

Petersen Graph Detection und $\overline{PGD}$ sind im unparametrisierten Fall gleich schwer, da das Komplement eines Graphen in n^2 Zeitschritten berechnet werden kann und man das Komplement des Petersen-Graphen in $O(n^{10})$ Zeitschritten finden kann. $\overline{PGD}$ hat also ebenso eine Laufzeit von $O(n^2+n^{10}) \in O(n^{10})$. Auch kann $\overline{PGD}$ in $O(m^5)$ Zeitschritten gelöst werden: Es gibt ein perfektes Matching im Komplement des Petersen-Graphen, da wir ein perfektes Matching aus nicht vorhandenen Kanten im Petersen-Graphen bilden können. Ein mögliches Matching dieser Art könnte aus den nicht existierenden Kanten $\{1,10\},\{2,6\},\{3,7\},\{4,8\}$ und $\{5,9\}$ wie in der Abbildung 5.1 abgebildet dargestellt werden. Dieses Matching bildet dann zwangsläufig im Komplementgraphen ein perfektes Matching. Wir vermuten, dass beide Probleme auch in den meisten parametrisierten Fällen gleich schwer zu lösen sind. Bezüglich $\overline{PGD}$ versuchten wir vergeblich eine Aussage zum Parameter Clique Cover zu treffen. Wir konnten allerdings folgende Beobachtung machen:

Beobachtung 5.1. $\overline{PGD}$ *parametrisiert mit* `Clique Cover` *entspricht* Petersen Graph Detection *parametrisiert mit* `Chromatic Number`.

Hierbei wird für den Eingabegraphen des parametrisierten Algorithmus für $\overline{PGD}$ der Komplementgraph gebildet und als Eingabegraph für den parametrisierten Algorithmus für Petersen Graph Detection verwendet. Es besteht also ein direkter Zusammenhang zwischen Petersen Graph Detection und $\overline{PGD}$ sowie zwischen den Parametern `Clique Cover` und `Chromatic Number`. Wir vermuten, dass beide Parameter *ℓ-GP-hart* sind: Kann man für Petersen Graph Detection parametrisiert mit `Chromatic Number` *ℓ-GP-Härte* zeigen kann, so vermuten wir, dass durch die Beziehung dieser Parameter zueinander die gleiche Idee genutzt werden kann, um *ℓ-GP-Härte* für Petersen Graph Detection parametrisiert mit `Minimum Clique Cover` zeigen kann. Ein ähnlicher Zusammenhang besteht zwischen den Parametern `Maximum Clique` und `Maximum Independent Set`: Das `Maximum Independent Set` eines Graphen stellt die Knoten der `Maximum Clique` des Komplementgraphen dar. Weiterhin wird der Parameter `Chordality` durch den Parameter `Chromatic Number` in der Graphparameter-Hierarchie von oben beschränkt, sodass bei einer *ℓ-GP-Härte* von `Chromatic Number` auch der Parameter `Chordality` *ℓ-GP-hart* ist.

In der Graphparameter-Hierarchie können wir für insgesamt fünf Parameter keine Aussagen treffen. Allerdings stehen diese Parameter bezüglich Petersen Graph Detection in einer so engen Beziehung, dass es sich hierbei im Prinzip um ein einziges Problem handelt. Somit verbleiben nicht fünf offene Probleme, sondern wahrscheinlich nur eines. Wir vermuten, dass für diese fünf Parameter *ℓ-GP-Härte* gezeigt werden kann. Diese Parameter können in zukünftigen Arbeiten weiter erforscht werden.

Eine weitere mögliche Fortführung unserer Arbeit liegt im Zählen oder Auflisten von Petersen-Graphen: Petersen Graph Detection zeigt nur auf, ob mindestens ein Petersen-Teilgraph in einem Graphen vorhanden ist, nicht aber wie viele Petersen-Graphen vorhanden sind und aus welchen Knoten des Graphen diese bestehen. Will man beispielsweise mit Petersen-Graph-freien Graphen arbeiten, so wäre das Entfernen von vorhandenen Petersen-Graphen, und damit die Entfernung von Knoten, aus welchen diese bestehen, durchaus interessant.

Literatur

[AH19] K. A. A. Al-Hajjana. *Diamond Detection in Graphs: Parameters, Algorithms, Hardness*. Bachelorarbeit. Technische Universität Berlin, 2019 (siehe S. 10).

[Åst+09] M. Åstrand, P. Floréen, V. Polishchuk, J. Rybicki, J. Suomela und J. Uitto. *A Local 2-Approximation Algorithm for the Vertex Cover Problem*. In: *Distributed Computing, Proceedings of the 23rd International Symposium (DISC 2009)*. Bd. 5805. Lecture Notes in Computer Science. Springer, 2009, S. 191–205 (siehe S. 16).

[AWW16] A. Abboud, V. V. Williams und J. R. Wang. *Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs*. In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*. SIAM, 2016, S. 377–391 (siehe S. 10).

[Ben+19] M. Bentert, T. Fluschnik, A. Nichterlein und R. Niedermeier. *Parameterized aspects of triangle enumeration*. In: *Journal of Computer and System Science* 103 (2019), S. 61–77 (siehe S. 10, 19).

[CFS08] L. S. Chandran, M. C. Francis und N. Sivadasan. *Boxicity and maximum degree*. In: *Journal of Combinatorial Theory, Series B* 98.2 (2008), S. 443–445 (siehe S. 18).

[Fel+09] M. R. Fellows, F. A. Rosamond, U. Rotics und S. Szeider. *Clique-Width is NP-Complete*. In: *SIAM Journal of Discrete Mathematics* 23.2 (2009), S. 909–939 (siehe S. 18).

[GJ79] M. R. Garey und D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979 (siehe S. 15, 17).

[GJS76] M. R. Garey, D. S. Johnson und L. J. Stockmeyer. *Some Simplified NP-Complete Graph Problems*. In: *Theory of Computer Science* 1.3 (1976), S. 237–267 (siehe S. 17).

[GMN17] A. C. Giannopoulou, G. B. Mertzios und R. Niedermeier. *Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs*. In: *Theory of Computer Science* 689 (2017), S. 67–95 (siehe S. 10).

[HS93] D. A. Holton und J. Sheehan. *The Petersen graph*. Bd. 7. Australian mathematical society lecture series. Cambridge University Press, 1993 (siehe S. 9).

[Kar72] R. M. Karp. *Reducibility among Combinatorial Problems*. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*. Hrsg. von R. E. Miller, J. W. Thatcher und J. D. Bohlinger. Boston, MA: Springer US, 1972, S. 85–103 (siehe S. 15–18).

[Kos+15] A. Kosowski, B. Li, N. Nisse und K. Suchan. *k-Chordal Graphs: From Cops and Robber to Compact Routing via Treewidth*. In: *Algorithmica* 72.3 (2015), S. 758–777 (siehe S. 19).

[LY80] J. M. Lewis und M. Yannakakis. *The Node-Deletion Problem for Hereditary Properties is NP-Complete*. In: *Journal of Computer and System Science* 20.2 (1980), S. 219–230 (siehe S. 15).

[MB83] D. W. Matula und L. L. Beck. *Smallest-Last Ordering and clustering and Graph Coloring Algorithms*. In: *Journal of the ACM* 30.3 (1983), S. 417–427 (siehe S. 16).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (siehe S. 13).

[Sch20] J. C. B. Schröder. *Comparing Graph Parameters*. Bachelorarbeit. Technische Universität Berlin, 2020 (siehe S. 10).

[Wat69] M. E. Watkins. *A theorem on Tait colorings with an application to the generalized Petersen graphs*. In: *Journal of Combinatorial Theory* 6.2 (1969), S. 152–164 (siehe S. 10).