ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Theoretische Informatik

Problemkern-Reduktion und Fixed Parameter Tractability von Graphenproblemen

Diplomarbeit an der Universität Ulm

Vorgelegt von:
Marcus Bombe
Marcus.Bombe@uni-ulm.de

Gutachter:
Prof. Dr. Jacobo Torán
Dr. Fabian Wagner

Betreuer:
Dr. Fabian Wagner

Abgabedatum: 28. Februar 2011

„Problemkern-Reduktion und Fixed Parameter Tractability von Graphenproblemen“
Fassung vom 28. Februar 2011

„Wer sich nicht auf der Schwelle des Augenblicks, alle Vergangenheit vergessend, niederlassen kann, der wird nie wissen, was Glück ist, und noch schlimmer: er wird nie etwas tun, was andere glücklich macht.“
(Nietzsche)

DANKSAGUNGEN: Mein Dank geht an die viele helfenden Menschen, die mir diese Diplomarbeit ermöglicht haben. Zunächst seien hier Prof. Jacobo Torán und Fabian Wagner genannt für ihre Betreuung und Mühe mit den vielen Korrekturen. Weiterhin geht ein großer Dank an Martin Bader für seine unschätzbar wertvolle fachliche Hilfe die Beweise zu MOO betreffend. In diesem Zuge auch einen ganz herzlichen Dank an Dominikus Krüger, ebenfalls für fachliche Unterstützung, Korrekturen und für die Auswahl des Problems MOO. Auch danke ich Oliver Gableske für seine zahlreichen hilfreichen Ratschläge und Korrekturen. Ich danke Guido de Melo für seine Unterstützung und wertvollen Ratschläge zum Schreiben der Diplomarbeit.

Mein ganz besonderer und alles überragender Dank geht an Finn Steglich. Ohne ihn und seine immerwährende fachliche, sowie freundschaftliche Hilfe und Unterstützung wäre mein Studium so nicht möglich gewesen.

Bei der Erstellung dieser Diplomarbeit wurde ausschließlich freie Software eingesetzt:

Satz: PDF-LaTeX 2_ε
Druck: KIZ Uni Ulm aus Studiengebühren

Problemkern-Reduktion und Fixed Parameter Tractability von Graphenproblemen

Marcus Bombe

Diplomarbeit an der Universität Ulm

Inhaltsverzeichnis

1 Einleitung

Wir betrachten in dieser Diplomarbeit das noch vergleichsweise junge Feld der parametrisierten Komplexitätstheorie, welches von Downey und Fellows [11] eingeführt wurde. Hierbei werden wir $\mathcal{NP}$-vollständige Graphenprobleme betrachten und wir werden untersuchen wie wir trotz ihrer $\mathcal{NP}$-Vollständigkeit mit ihnen umgehen können. Viele der Problemstellungen im Bereich der $\mathcal{NP}$-vollständigen Graphenprobleme sind aus praktischen Bedürfnissen von Gebieten wie der Bioinformatik entstanden. Hier besteht ein hoher Bedarf an Algorithmen für solche schwierigen Probleme. Bisher ist es üblich, diese schwierigen Probleme approximativ und durch Heuristiken zu lösen. In dieser Arbeit werden wir statt dessen eine Technik kennenlernen, exakte Lösungen möglichst schnell zu konstruieren.

Diese Arbeit beschäftigt sich mit parametrisierten Problemen. Dies sind Sprachen L der Form $L \subseteq \Sigma^* \times \mathbb{N}$ wobei Σ das Eingabealphabet ist und die zweite Komponente *Parameter* genannt wird. Ein Beispiel für ein parametrisiertes Graphenproblem ist das $\mathcal{NP}$-vollständige Vertex Cover:

> Vertex Cover:
> Gegeben: Ein einfacher Graph $G = (V, E)$ und $k \in \mathbb{N}_0$.
> Gesucht: Eine Knotenmenge $S \subseteq V$ mit $|S| \leq k$, sodass jede Kante aus E mindestens einen ihrer beiden Endknoten in S hat.

Der Parameter ist hierbei k, die maximal gestattete Anzahl von Knoten der Knotenmenge S. Ein Beispiel für Vertex Cover ist in Abbildung 1.1 dargestellt.

Viele dieser parametrisierbaren Graphenprobleme lassen sich trotz ihrer $\mathcal{NP}$-Vollständigkeit mit sogenannten Datenreduktionsregeln in polynomieller Zeit ver-

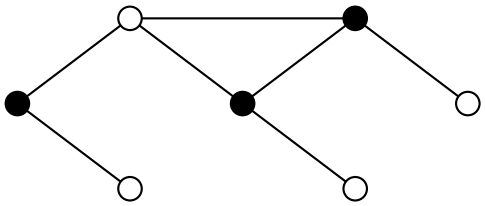


Abbildung 1.1: Ein Beispiel für Vertex Cover: Die schwarzen Knoten bilden die gesuchte Knotenmenge S.

einfachen. Eine triviale Regel für Vertex Cover ist beispielsweise Knoten von Grad 0 zu ignorieren.

Einige Graphenprobleme haben sehr geschickte polynomielle Datenreduktionsregeln: Die „kombinatorische Explosion“ in der Laufzeit eines Algorithmus für ein solches $\mathcal{NP}$-vollständiges Graphenproblem lässt sich auf den Parameter k einschränken. Anstatt dass es sich um eine exponentielle Laufzeit in der Eingabelänge n handelt, können manche parametrisierten Probleme in $f(k) \cdot n^{O(1)}$ Zeit gelöst werden, wobei die Funktion f berechenbar ist und lediglich vom Parameter k abhängt, nicht jedoch von der Eingabelänge n. Die zugehörige Komplexitätsklasse wird *fixed-parameter tractable*, oder kurz $\mathcal{FPT}$, genannt. Diese Probleme sind dann trotz $\mathcal{NP}$-Vollständigkeit auch bei großen Eingabeinstanzen gut lösbar, wenn nur der Parameter k hinreichend klein ist. Wir gewinnen durch diese Art der Betrachtung eine Möglichkeit mit Problemen umzugehen, die nach klassischer Anschauung für zu schwer gehalten werden, um sie exakt zu lösen.

Diese Arbeit stellt vor, dass parametrisierte Probleme genau dann in der Klasse $\mathcal{FPT}$ sind, wenn sie einen sogenannten *Problemkern* besitzen. Dies bedeutet, dass sich über Datenreduktionsregeln das Graphenproblem in polynomieller Zeit derart verkleinern lässt, dass der übrig bleibende Graph in seiner Größe nur noch vom Parameter k abhängt. Auf dieser Grundlage wird diese Diplomarbeit im Folgenden die Techniken der *Problemkern-Reduktion* auf verschiedenen Graphenproblemen untersuchen. Hierbei werden wir in Kapitel 3 das bereits ausführlich untersuchten Graphenproblem Vertex Cover betrachten und uns einen Über-

blick über die Bandbreite der möglichen Techniken für die *Problemkern-Reduktion* verschaffen.

Ein Schwerpunkt dieser Diplomarbeit ist die Untersuchung eines bisher noch nicht im Kontext der Parametrisierung betrachtetes Graphenproblem: In Kapitel 4 wird das Graphenproblem Minimum Outdegree Orientation analysiert, welches 2006 von Asahiro et al. [2] definiert wurde. Hierbei wird ein Graph untersucht, dessen Kanten jeweils ein *Kantengewicht* haben. Aufgabe ist es nun für jede Kante des Graphen eine Richtung festzulegen. Es wird pro Knoten die Summe der Kantengewichte von ausgehenden Kanten berechnet; wir bezeichnen diese Summe als *gewichteten Ausgangsgrad*. Ziel ist es, dass durch geschickte Wahl der Richtungen der Kanten ein möglichst kleiner Maximalwert für den gewichteten Ausgangsgrad zu Stande kommt. Formal lautet das Problem wie folgt:

> Minimum Outdegree Orientation:
> Gegeben: Ein gewichteter Graph $G = (V, E, w)$.
> Gesucht: Eine Orientierung Λ, sodass der maximale gewichtete Ausgangsgrad $\Delta_\Lambda(G)$ minimal ist.

Diese Diplomarbeit zeigt für zwei Parametrisierungen von Minimum Outdegree Orientation, dass diese in $\mathcal{FPT}$ liegen. Der erste Parameter, für den dieser Nachweis gelingt, ist die Anzahl von Kanten, die ein anderes Gewicht als c haben, wobei c ein für jeden Graphen individuelles Gewicht ist. Der zweite Parameter, für den gezeigt werden kann, dass Minimum Outdegree Orientation in $\mathcal{FPT}$ liegen, ist die Anzahl der Kanten, die den Graphen daran hindern, nur aus Zyklen und Pfaden zu bestehen.

Darüber hinaus wird auch ein neuer $\mathcal{NP}$-Vollständigkeitsbeweis für Minimum Outdegree Orientation angegeben. Dieser neue Beweis zeigt, dass das Problem Minimum Outdegree Orientation bereits dann $\mathcal{NP}$-vollständig ist, wenn der Grad der Knoten des Graphen auf 3 beschränkt wird und lediglich zwei verschiedene Kantengewichte verwendet werden.

Die in dieser Diplomarbeit gefundenen Ergebnisse zu Minimum Outdegree Orientation werden am Ende von Kapitel 4 diskutiert. In Kapitel 5 werden die Ergebnisse dieser Arbeit nochmals abschließend zusammengefasst und es wird auf offene Fragen verwiesen, welche sich durch die vorliegende Diplomarbeit ergeben haben.

2 Grundlagen

In diesem Abschnitt führen wir die Notationen und verwendeten Grundlagen innerhalb dieser Diplomarbeit ein. Die Notation entspricht weitestgehend der aus Niedermeiers Habilitationsschrift [23] und dem daraus entstandenem Buch [24].

Mengen und Zeichenketten

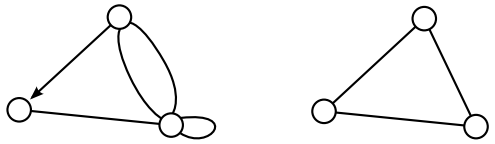
Die natürlichen Zahlen bezeichnen wir mit $\mathbb{N}$, sofern alle positiven Ganzzahlen gemeint sind und mit $\mathbb{N}_0$, wenn zudem die Null hinzu genommen wird. Mit $\mathbb{R}$ wird die Menge der reellen Zahlen bezeichnet. Mit $|M|$ wird die Anzahl der Elemente einer Menge M bezeichnet. Mit $A \subseteq B$ wird angegeben, dass A eine (nicht notwendigerweise echte) Teilmenge von B ist.

In der Regel bezeichnen wir mit Σ endliche Mengen zur Kodierung von Ein- bzw. Ausgabe und nennen Σ daher ein *Alphabet*. Eine *Zeichenkette* (auch Wort oder String genannt) ist die Aneinanderreihung von Elementen aus Σ. Mit Σ^* bezeichnen wir die Menge aller Zeichenketten die sich durch Aneinanderreihung beliebig vieler Elemente aus Σ ergeben.

2.1 Graphen, Planarität und Bäume

Ein (endlicher) *Graph* G ist gegeben als endliche Menge von *Knoten* V (englisch *vertex*) und einer Menge von *Kanten* E (englisch *edge*) zwischen je zwei dieser Knoten. Man schreibt $G = (V, E)$ mit $|V| < \infty,\ E \subseteq V \times V$.

Sofern nicht anders angegeben, betrachten wir nur Graphen, die *ungerichtet*, *ohne Mehrfachkanten* und *ohne Schleifen* vorliegen. In einem ungerichteten Graphen haben die Kanten keine Richtung; es gilt statt dessen $\forall v, u \in V : (u, v) \in E \Leftrightarrow (v, u) \in E$. Wir schreiben daher auch einfach $\{u, v\} \in E$, wobei $\{u, v\}$ ein ungeordnetes Paar von Knoten ist. Ohne Mehrfachkanten ist ein Graph, wenn zwischen zwei Knoten höchstens eine Kante verläuft. Ein Graph hat keine Schleifen, wenn es keine Kanten der Form $(u, v) \in E : u = v$ gibt, also Kanten von einem Knoten zu sich selbst verboten sind. Ein ungerichteter Graph ohne Mehrfachkanten und ohne Schleifen wird *einfach* genannt. Siehe hierzu auch Abbildung 2.1.



(a) (b)

Abbildung 2.1: Zwei Graphen mit verschiedenen Eigenschaften.
2.1(a) zeigt einen Graphen mit Mehrfachkanten, einer Schleife und einer gerichteten Kante. 2.1(b) zeigt hingegen einen einfachen Graphen

Weitere Grapheneigenschaften

Wenn ein Knoten in einer Kante enthalten ist, so ist dieser Knoten mit dieser Kante und die Kante mit dem Knoten *inzident*. Zwei verschiedene Knoten, die durch eine Kante verbunden sind, also mit der selben Kante inzident sind, heißen *adjazent*. Ebenso werden zwei verschiedene Kanten adjazent genannt, wenn sie mit dem selben Knoten inzident sind. Adjazente Knoten beziehungsweise Kanten heißen auch *benachbart*. Die *Nachbarschaft* $N(v)$ eines Knoten v ist die Menge seiner benachbarten Knoten, d.h. $N(v) = \{u \mid \{u, v\} \in E\}$. Der *Grad* eines Knotens ist die Anzahl der zu ihm inzidenten Kanten, wobei Schleifen doppelt gezählt werden. Bei einfachen Graphen gilt also $\forall v \in V : grad(v) = |N(v)|$. Zudem setzen wir $N[v] = N(v) \cap \{v\}$. Ein *Pfad* von einem Knoten v_0 zu einem Knoten v_k ist

eine Menge von Kanten $\{\{v_i, v_{i+1}\}|\{v_i, v_{i+1}\} \in E, v_0, v_1, \dots v_k \in V\}$. Ein Graph heißt *zusammenhängend*, falls es von jedem Knoten u des Graphen zu jedem anderen Knoten v des Graphen einen *Pfad* gibt. Ist der Graph hingegen *unzusammenhängend*, zerfällt er in mehrere *Zusammenhangskomponenten*.

Der Graph $G_1 = (V_1, E_1)$ ist ein *Teilgraph* von $G_2 = (V_2, E_2)$, falls gilt $V_1 \subseteq V_2$ und $E_1 \subseteq E_2$. Für einen Graphen $G = (V, E)$ und eine Knotenmenge $V_0 \subseteq V$ wird der durch V_0 *induzierte Teilgraph* durch $G[V_0] = (V_0, E_0)$ bezeichnet, wobei $E_0 = \{\{u, v\} \in E \mid u, v \in V_0\}$, siehe hierzu auch Abbildung 2.2(c). Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ heißen *isomorph*, wenn eine bijektive Abbildung $g : V_1 \to V_2$ existiert mit $\{u, v\} \in E_1 \Leftrightarrow \{g(u), g(v)\} \in E_2$. Eine *Knotenverschmelzung* in einem einfachen Graphen ist das Ersetzen zweier benachbarter Knoten $u, v \in V$ und der zu u und v inzidenten Kanten durch einen neuen Knoten w und Kanten entsprechend, sodass $N(w) = (N(u) \cup N(v)) \setminus \{u, v\}$ gilt. Eventuell auftretende Mehrfachkanten werden hierbei zu einer einzigen Kante zusammengefasst. Ein Graph G_1 heißt *Minor* eines Graphen G_2, falls G_2 nach Anwendung beliebig vieler Knotenverschmelzungen sowie Entfernung von beliebig vielen Kanten oder Knoten isomorph zu G_1 ist, betrachte hierfür auch Abbildung 2.2(b).

Planarität

Ein einfacher Graph $G = (V, E)$ heißt *bipartit*, falls sich V derart in zwei Knotenmengen V_1 und V_2 zerlegen lässt, sodass gilt $V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$. Zudem muss gelten, dass die inzidenten Knoten jeder Kante aus verschiedenen Mengen kommen, also $\forall\{u, v\} \in E : u \in V_1, v \in V_2$. Eine *vollständige Bipartition* eines Graphen, sofern existent, ist eine Zerlegung, sodass alle Knoten der einen Menge jeweils mit allen Knoten der anderen Menge adjazent sind. Einen solchen Graphen nennt man $K_{n,m}$ wobei n und m die Anzahl der Knoten der beiden Mengen sind. Für unsere Zwecke benötigen wir den $K_{3,3}$, siehe Abbildung 2.3(a).

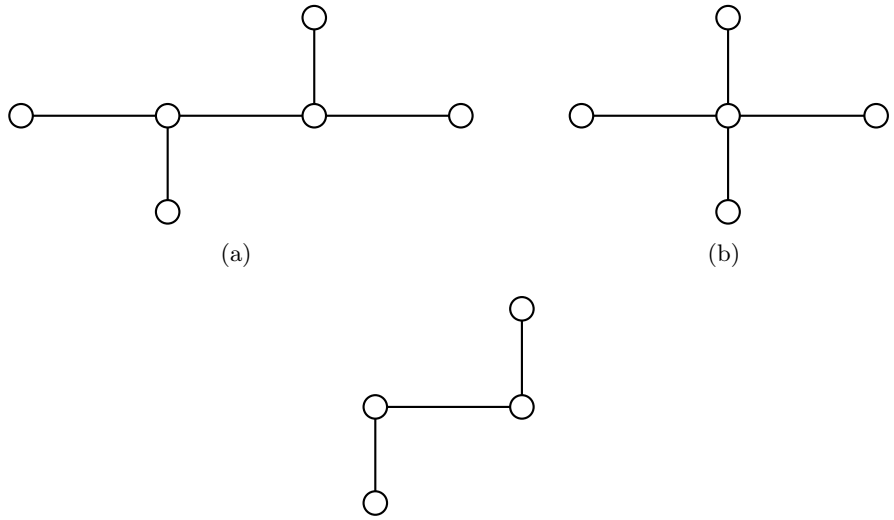
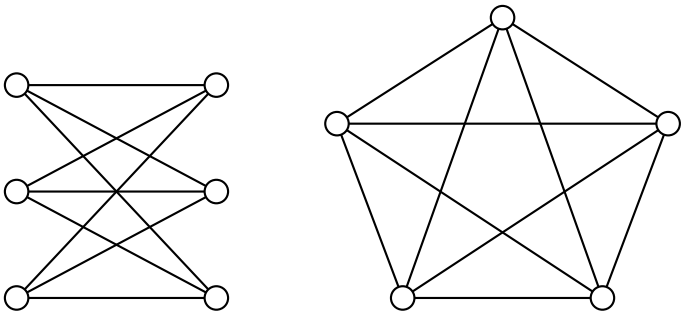


Abbildung 2.2: Beispiel für Minorenbildung und einen induzierten Teilgraphen. 2.2(a) zeigt den Ausgangsgraph. 2.2(b) ist ein Minor des Ausgangsgraphen (durch Verschmelzung der mittleren beiden Knoten) und 2.2(c) zeigt einen induzierten Teilgraphen des Ausgangsgraphen.

Als vollständigen Graphen bezeichnet man einen einfachen Graphen $G = (V, E)$ bei dem alle Knoten paarweise adjazent sind. Sei $|V| = n$, so gilt für alle Knoten $v \in V : grad(v) = n - 1$ und $|E| = n \cdot (n-1)/2$, letzteres zeigt sich leicht über vollständige Induktion. Einen solchen Graphen bezeichnet man auch K_n. Ein Vertreter ist hierbei der K_5, siehe Abbildung 2.3(b), welchen wir im Folgenden noch verwenden werden.

Planare Graphen sind spezielle Graphen die sich derart in eine Ebene zeichnen lassen, dass sich keine Kanten überkreuzen und keine Knoten überdecken. Zurückgehend auf Kuratowski gilt:

Satz 2.1 (Satz von Wagner, [28])**.** *Ein Graph G ist genau dann planar, wenn weder K_5 noch $K_{3,3}$ ein Minor des Graphen G ist.*



(a) (b)

Abbildung 2.3: Zwei wichtige Graphen für die Fragestellung der Planarität. Links der $K_{3,3}$, rechts der K_5

Einfache planare Graphen haben die Eigenschaft, dass die Anzahl ihrer Kanten linear in der Anzahl ihrer Knoten ist. Für einen planaren Graphen $G = (V, E)$ gilt $|E| \leq 3n - 6$ für $n = |V| \geq 3$. Ein Beweis dieser sogenannten *Eulerformel* (auch *eulerscher Polyedersatz* genannt) ist beispielsweise in [9] zu finden. Da allgemeine einfache Graphen hingegen quadratisch viele Kanten in der Anzahl ihrer Knoten haben können, bezeichnet man planare Graphen auch als *dünn besetzt*.

Bäume

Ein einfacher zusammenhängender azyklischer Graph wird *Baum* genannt. Ein einfacher zusammenhängender Graph heißt *azyklisch*, falls es zwischen je zwei seiner Knoten stets genau einen einzigen Pfad gibt. Ein Baum mit n Knoten hat stets $n - 1$ Kanten. Durch das Entfernen einer Kante zerfällt der Baum in zwei Bäume. Eine Menge von Bäumen wird auch *Wald* genannt. Einige Beispiele für Bäume sind in Abbildung 2.2 zu sehen.

Man kann bei Bäumen zudem einen Knoten speziell auswählen und ihn *Wurzel* nennen. Hierdurch erhalten die Kanten und Knoten im Baum eine Orientierung: Man unterscheidet die Bewegungsrichtung hin zur Wurzel und weg von der Wur-

zel. Von der Wurzel gibt es genau einen eindeutigen Pfad zu jedem anderen Knoten x. Die Knoten entlang des Pfades zwischen x und der Wurzel werden, mit Ausnahme von x selbst, als *Vorfahren* von x bezeichnet. Alle Knoten, zu denen x selbst ein Vorfahre ist, werden als die *Nachfahren* von x bezeichnet. Der adjazente Vorfahre eines Knotens heißt auch *Elternknoten*, die adjazenten Nachfahren eines Knotens werden auch dessen *Kinder* genannt. Ein Baum bei dem jeder Knoten nur höchstens 2 Kinder hat, wird Binärbaum genannt. Ein Knoten ohne Nachfahren heißt *Blatt* und hat Grad 1. Oft werden Bäume mit Wurzel wie in Abbildung 2.4 gezeichnet, sodass die Wurzel ganz oben zu finden ist und alle Kanten von einem Elternknoten zu dessen Kinder stets nach weiter unten reichen.

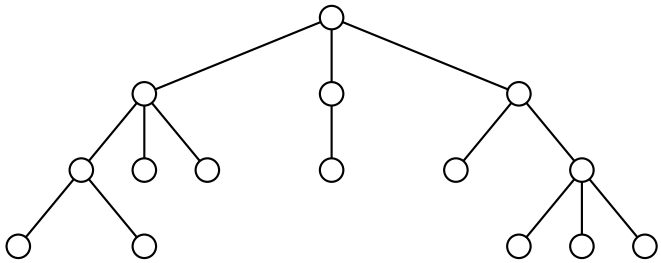


Abbildung 2.4: Ein Baum aus 15 Knoten mit 9 Blättern

Speicherungsarten von Graphen

Um Graphen in Algorithmen verwenden zu können, müssen sie in geeigneter Form vorliegen. Üblich sind in der Praxis hierbei zwei verschiedene Speicherungsarten für einen Graphen $G = (V, E)$:

Adjazenzmatrix: Zur Speicherung eines Graphen in einer Adjazenzmatrix wird eine quadratischen Matrix $M = (m_{ij})_{i,j=1}^{n}$ der Größe $n \times n$ mit $n = |V|$ aufgestellt. Der Matrixeintrag m_{ij} ist genau dann 1, wenn eine Kante zwischen den Knoten i und j existiert, andernfalls ist $m_{ij} = 0$. Die Matrix ist für ungerichtete Graphen symmetrisch. Der Vorteil der Adjazenzmatrix besteht darin, dass Zugriffe

in konstanter Zeit durchgeführt werden können. Zudem kann die Adjazenzmatrix leicht zur Speicherung weiterer Grapheigenschaften verwendet werden, etwa können Kantengewichte (siehe Kapitel 4) eingetragen werden. Der Nachteil der Adjazenzmatrix ist ihr quadratischer Platzbedarf in der Anzahl der Knoten.

Adjazenzliste: In einer Adjazenzliste wird ein Graph dadurch gespeichert, dass jeder Knoten eine verkettete Liste von Verweisen auf diejenigen Knoten enthält, zu denen er benachbart ist. Der Vorteil der Adjazenzliste besteht darin, dass ihr Speicherplatzbedarf nur linear in der Anzahl der Knoten und Kanten ist. In der Praxis haben Graphen oft wesentlich mehr Knoten als die durchschnittliche Anzahl inzidenter Kanten an einem Knoten. In solchen dünn besetzten Graphen wird dann lediglich linear viel Platz in der Anzahl der Knoten zur Speicherung der Adjanzenzliste benötigt. Aus diesem Grund wird die Adjazenzliste in vielen polynomialzeit Graphenalgorithmen der Adjazenzmatrix vorgezogen, siehe hierzu auch Schöning [27]. Der Nachteil der Adjazenzliste ist ihre schlimmstenfalls lineare Zugriffszeit.

2.2 Entscheidungs- und Optimierungsprobleme

Als *Entscheidungsproblem* bezeichnet man die Aufgabe festzustellen, ob für eine gegebene Eingabe die Antwort auf die problemspezifische Fragestellung „ja“ oder „nein“ zu lauten hat. Ein Entscheidungsproblem definiert hierbei eine Sprache L über einem Alphabet Σ, also $L \subseteq \Sigma^*$. Eine konkrete Eingabe x mit $x \in \Sigma^*$ wird *Instanz* genannt und das Entscheidungsproblem lautet „gilt $x \in L$?“. Ein Problem heißt *entscheidbar*, falls die Antwort auf „gilt $x \in L$?“ für jedes x in endlicher Zeit gefunden werden kann. Genauer: Ein Problem heißt entscheidbar, wenn die charakteristische Funktion

$$\chi_L(x) = \begin{cases} 1, & \text{falls } x \in L \\ 0, & \text{sonst} \end{cases}$$

für alle $x \in \Sigma^*$ *berechenbar* ist, d.h. es existiert ein Algorithmus, der die charakteristische Funktion (für beliebiges $x \in \Sigma^*$) in endlicher Zeit korrekt berechnet.

Neben den Entscheidungsproblemen gibt es noch die *Optimierungsprobleme*, bei denen die gesuchte Antwort die Minimierung oder Maximierung eines Wertes ist. Oft existiert zu einem Entscheidungsproblem auch ein Optimierungsproblem. Und mit der Fragestellung ob ein gewisser Wert optimal ist, existiert zu einem Optimierungsproblem auch immer ein Entscheidungsproblem.

Betrachten wir ein kleines Beispiel: Angenommen, gegeben sei ein Graph, dann wäre ein einfaches Entscheidungsproblem herauszufinden „existiert ein Pfad zwischen Knoten u und Knoten v?“ und ein einfaches Optimierungsproblem wäre „was ist die Länge des kürzesten Pfades zwischen u und v?“.

Laufzeiten und Berechnungsmodell

Um die Schwierigkeit eines Problems untersuchen zu können, betrachten wir die Laufzeit eines Algorithmus für dieses Problem abhängig von der Länge der Eingabe. Wir verwenden hierzu die *O-Notation*, auch Landau-Symbol genannt, und gehen dabei stets von der Worst-Case-Laufzeit aus. Das bedeutet, dass wir lediglich den von der Laufzeit längstmöglichsten Fall betrachten und außer Acht lassen, ob das Problem etwa für die allermeisten Eingaben wesentlich schneller gelöst werden kann.

Für Funktionen f und g und eine Eingabelänge n gilt die folgenden Definition:

$$g \in O(f(n)) \Leftrightarrow \exists\, c>0\ \exists\, n_0>0\ \forall\, n \geq n_0 \colon g(n) \leq c \cdot f(n)$$

Das bedeutet, dass g bis auf einen konstanten Faktor höchstens dasselbe Wachstum besitzt wie f. Zudem gilt für die O-Notation eine besondere Verwendung des Gleichheitszeichens: Anstatt $g \in O(f(n))$ schreibt man auch $g = O(f(n))$, also beispielsweise $3n^2 = O(n^2)$. Ebenso schreibt man für $O(g(n)) \subseteq O(f(n))$ auch

$O(g(n)) = O(f(n))$. Allerdings ist dieses Gleichheitszeichen nur von links nach rechts zu lesen, denn es gilt zwar beispielsweise $O(n^2) = O(n^7)$, nicht jedoch $O(n^7) = O(n^2)$. Weitere Details zur O-Notation werden etwa in Schöning [27] beschrieben.

Wir verwenden in dieser Arbeit das *Random Access Machine* (RAM) Berechnungsmodell um die Laufzeit von Algorithmen zu untersuchen. Im Wesentlichen beinhaltet das RAM-Modell, dass arithmetische Operationen wie Addition oder Multiplikation, sowie Zuweisungen und Vergleiche von Zahlen jeweils in einem einzigen Zeitschritt durchgeführt werden können. Zusätzlich benötigen alle Speicherzugriffe genau einen Zeitschritt. Beachtet werden muss hierbei jedoch, dass sowohl bei den Operationen als auch bei den Speicherzugriffen die Größe der beteiligten Zahlen nicht beliebig sein kann, sondern alle beteiligten Zahlen jeweils noch in eine konstante Anzahl von Speicherzellen (üblicherweise eine Speicherzelle) passen müssen. Andernfalls wäre eine missbräuchliche Verwendung des Berechnungsmodell möglich: Etwa ist die Berechnung von a^b nur für sehr kleine a und b durch $b - 1$ viele Multiplikationen zu je einem Zeitschritt, also in Linearzeit, möglich. Für große a und b benötigt bereits das Aufschreiben des Ergebnisses exponentielle Zeit.

2.3 Komplexitätstheorie

Im vorhergehenden Abschnitt haben wir die Laufzeit-Notation $O(p(n))$ kennen gelernt. Sei nun $p(n)$ ein beliebiges Polynom und n die Anzahl der Bits der Länge der Eingabe, so fällt nun jedes Problem per Definition in die Komplexitätsklasse $\mathcal{P}$, falls es für dieses Problem einen Algorithmus und ein Polynom p gibt, sodass dessen Laufzeit in $O(p(n))$ liegt. Eine solche Laufzeit nennen wir *polynomiell*. Algorithmen dieser Komplexitätsklasse werden im Allgemeinen als effizient angesehen.

Die Klasse $\mathcal{NP}$ beinhaltet alle Probleme L, deren Algorithmen nichtdeterministisch polynomielle Laufzeiten haben. Nichtdeterministisch polynomiell bedeutet, dass es mehrere Rechenwege im Algorithmus gibt, wobei jeder einzelne Rechenweg nur polynomielle Laufzeit haben darf. Für jedes $x \in L$ muss nun gelten, dass es jeweils *mindestens einen* Rechenweg gibt, bei dem der Algorithmus ebenfalls zum Ergebnis $x \in L$ gelangt. Zudem muss für alle $x \notin L$ gelten, dass der Algorithmus auf *jedem* seiner Rechenwege $x \notin L$ als Ergebnis ausgibt.

Da reale Computer jedoch deterministisch arbeiten, kann ein nichtdeterministisch polynomieller Algorithmus nach aktuellem Erfahrungsstand im Allgemeinen nur durch einen exponentiellen Algorithmus auf realen Computern simuliert werden. Algorithmen mit exponentieller Laufzeit sind nur für kleine Eingabegrößen in praxisrelevanter Zeit durchführbar. Deswegen werden solche Algorithmen als ineffizient angesehen. Es gilt $\mathcal{P} \subseteq \mathcal{NP}$. Ob es sich mit $\mathcal{P}$ um eine echte Teilmenge von $\mathcal{NP}$ handelt, ist eine offene Fragestellung.

Mit Hilfe der *polynomiellen Reduzierbarkeit* lassen sich verschiedene Probleme aus der Klasse $\mathcal{NP}$ genauer kategorisieren. Ein Entscheidungsproblem $A \subseteq \Sigma^*$ heißt polynomiell reduzierbar auf ein Entscheidungsproblem $B \subseteq \Sigma^*$, falls es eine berechenbare Funktion f gibt mit $x \in A \Leftrightarrow f(x) \in B, \forall x \in \Sigma^*$ mit f ist polynomiell in $|x|$. A ist damit in gewisser Hinsicht ein Spezialfall von B. Daher heißt ein Problem B *$\mathcal{NP}$-hart*, falls sich alle Probleme $A \in \mathcal{NP}$ in polynomieller Zeit auf B reduzieren lassen. Gilt zudem $B \in \mathcal{NP}$, so heißt B zudem *$\mathcal{NP}$-vollständig*. Eines der bekanntesten $\mathcal{NP}$-vollständigen Problemen ist das Erfüllbarkeitsproblem der Aussagenlogik SAT.

Siehe [27] für eine ausführlichere Einführung in die Klasse $\mathcal{NP}$. Eine Liste von über 300 $\mathcal{NP}$-vollständigen Problemen findet sich in Garey und Johnson [15].

2.4 Parametrisierte Komplexitätstheorie

Diese Arbeit beschäftigt sich mit *parametrisierten Entscheidungsproblemen*. Hierbei betrachten wir zusätzlich zur Eingabe x einen *Parameter* $k \in \mathbb{N}_0$. Die Problemstellung lautet dann herauszufinden, ob x eine problemspezifische Eigenschaft hat, die wiederum von k abhängt. Vertex Cover ist hierbei das Standardbeispiel für ein parametrisiertes Entscheidungsproblem auf Graphen:

> Vertex Cover:
> Gegeben: Ein einfacher Graph $G = (V, E)$ und $k \in \mathbb{N}_0$.
> Gesucht: Eine Knotenmenge $S \subseteq V$ mit $|S| \leq k$, sodass jede Kante aus E mindestens einen ihrer beiden Endknoten in S hat.

Siehe hierzu auch Abbildung 2.5 für zwei Beispiele eines Vertex Covers an einem konkreten Graphen.

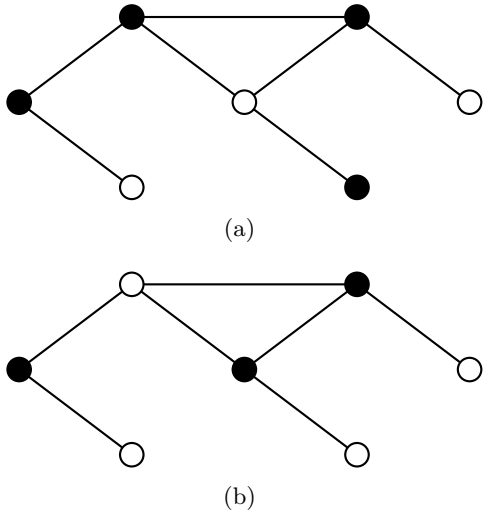


Abbildung 2.5: Beispiele für Vertex Cover.
Die schwarzen Knoten sind in der jeweiligen gesuchten Menge S enthalten. Oben ist ein Vertex Cover des Graphen der Größe $k = 4$ zu sehen, während unten ein optimales Vertex Cover des Graphen der Größe $k = 3$ abgebildet ist.

Eine formale Definition von parametrisierten Problemen folgt:

Definition 2.2. *Sei Σ ein endliches Alphabet. Ein parametrisiertes Problem ist eine Sprache $L \subseteq \Sigma^* \times \Sigma^*$. Die zweite Komponente wird hierbei Parameter genannt.*

Der Parameter ist für viele Probleme aus $\mathbb{N}_0$. Allerdings sind auch Strukturen wie Beispielsweise Teilgraphen, Minoren o.ä. potentiell als Parameter geeignet, daher unterbleibt hier eine Einschränkung auf $\mathbb{N}_0$. In der Literatur wird jedoch oft $L \subseteq \Sigma^* \times \mathbb{N}_0$ angenommen. Der Parameter wird, insbesondere wenn er aus $\mathbb{N}_0$ kommt, in der Regel unär kodiert [24].

Die Wahl des Parameters ist keinesfalls eindeutig. Im Laufe dieser Arbeit werden wir verschiedene Parametrisierungen kennen lernen. Oft werden Probleme auch auf verschiedene Arten parametrisiert um verschiedene Ergebnisse in der Komplexitätstheorie zu erlangen. So untersucht Krüger [20] beispielsweise das Graphenproblem Maximum Tree Orientation in sechs verschiedenen Parametrisierungsvarianten. Häufige Parameter bei Graphenproblemen sind die Anzahl der Knoten oder Kanten der zulässigen Lösung. Solche Parameter ergeben sich oft direkt aus der Problemdefinition. Als Parameter können aber auch strukturelle Eigenschaften des Graphen gewählt werden, etwa der maximale Knotengrad oder die Anzahl von Kanten, die den Graphen zyklisch machen.

Weitere spezielle Parametrisierungen existieren und werden etwa von Niedermeier [23] beschrieben. Beispielsweise gibt es für manche Probleme garantierte Lösungen. Etwa erhält man für Vertex Cover immer eine Lösung wenn alle Knoten gewählt werden. Das Problem Independent Set auf planaren Graphen, welches wir am Ende dieses Kapitel genauer betrachten, enthält als zu maximierende Lösung stets mindestens ein Viertel aller Knoten. Eine mögliche Parametrisierung ist der Abstand zu diesen garantierten Werten. Es gibt auch Probleme, für die eine Untersuchung von mehr als einem Parameter zur selben Zeit naheliegend ist. Beispielsweise untersuchen Fernau und Niedermeier [13] das Graphenproblem Constraint Bipartite Vertex Cover unter Verwendung zweier Parameter.

Die Komplexitätsklasse $\mathcal{FPT}$

Betrachten wir ein parametrisiertes Problem aus $\mathcal{NP}$ mit Eingabe x und Parameter k. Man kann nun die Beobachtung machen, dass es hierbei Probleme gibt, die deterministische Algorithmen haben, welche lediglich im Parameter k exponentielle Laufzeit vorweisen und polynomiell in der Eingabegröße $n = |x|$ sind. Die *kombinatorische Explosion* [23] dieser schwierigen Probleme lässt sich somit auf den Parameter k einschränken. Oft gilt für den Parameter zudem $k \ll n$. Vertex Cover ist ein solches Problem: Die derzeit besten bekannte parametrisierte Algorithmen für Vertex Cover haben eine bessere Laufzeit als $O(kn + 1{,}29^k)$ [7]. Ein solcher Algorithmus gilt für kleine k als effizient. Für Algorithmen dieser Art definiert man eine eigene Komplexitätsklasse:

Definition 2.3. *Sei L ein parametrisiertes Problem. L heißt* fixed-parameter tractable*, falls $(x, k) \in L$ mit $|(x, k)| = n$ in $f(k) \cdot n^{O(1)}$ Zeitschritten entschieden werden kann. Hierbei ist f eine beliebige berechenbare Funktion, die nur von k abhängt. Die zugehörige Komplexitätsklasse heißt $\mathcal{FPT}$.*

Wichtig ist, dass ein Problem stets nur bezüglich eines konkreten Parameters in $\mathcal{FPT}$ liegt. Wählt man für ein Problem, dass bezüglich eines Parameters in $\mathcal{FPT}$ liegt, eine andere Parametrisierung, so kann dies zu einer anderen Komplexität führen.

Wenn wir einen entsprechenden Algorithmus für ein schwieriges bzw. $\mathcal{NP}$-hartes Problem in $\mathcal{FPT}$ kennen und der Parameter k hinreichend klein ist, so ist eine Möglichkeit gefunden, ein eigentlich schwieriges Problem mit entsprechenden Einschränkungen doch effizient zu lösen. In der theoretischen Informatik sind hierfür auch weitere Vorgehensweisen bekannt, etwa approximative Algorithmen, heuristische Methoden oder etwa randomisierte Algorithmen. Ein zusätzlicher Ansatz, die Schwierigkeit einiger Probleme zu umgehen, ist die Betrachtung der Average-Case-Komplexität, also des Aufwandes für eine durchschnittliche Eingabe anstatt

der Worst-Case-Komplexität, die den Aufwand im ungünstigsten Fall der Eingabe betrachtet.

Um die Herangehensweise mittels parametrisierten Algorithmen für Probleme in $\mathcal{FPT}$ zwischen diesen verschiedenen Methoden einsortieren zu können, betrachten wir die wichtigsten Eigenschaften von Algorithmen für Probleme in $\mathcal{FPT}$:

- Vorteil: Die Algorithmen sind deterministisch und die Lösungen sind exakt und optimal.
- Vorteil: Man kann beweisbare obere Schranken für die Komplexität der Algorithmen angeben.
- Nachteil: Exponentielle Laufzeit im Parameter k.

Ähnlich der oben erklärten polynomiellen Reduktion lässt sich auch eine Reduktion auf parametrisierten Problemen definieren um auch hier die *relative Schwierigkeit* zweier Probleme abschätzen zu können:

Definition 2.4. *Seien L_1, L_2 parametrisierte Probleme. L_1 heißt* parametrisiert reduzierbar *auf L_2 falls es berechenbare Funktionen $f(k)$, $g(k)$ sowie eine Abbildung $\varphi : (x,k) \mapsto x'$ gibt mit $(x,k) \in L_1 \Leftrightarrow (x', f(k)) \in L_2$, sodass φ in $g(k) \cdot |(x,k)|^{O(1)}$ Zeit berechenbar ist.*

Es handelt sich also um eine Reduktion deren Laufzeitschranke an die Gegebenheiten von parametrisierten Problemen angepasst wurde. Die *meisten* ([23]) bekannten Reduktionen aus $\mathcal{P}$ und $\mathcal{NP}$ lassen sich nicht als parametrisierte Reduktionen durchführen. Dies liegt insbesondere daran, dass die Funktion $f(k)$ wieder lediglich vom Parameter k abhängen darf. In vielen Reduktionen aus $\mathcal{P}$ und $\mathcal{NP}$ hängt die Reduktion jedoch auch von der Eingabe x selbst ab. Wir führen als Beispiel hierfür folgendes Problem an:

Independent Set:
Gegeben: Ein einfacher Graph $G = (V, E)$ und $k \in \mathbb{N}_0$.

> Gesucht: Eine Knotenmenge $S \subseteq V$ mit $|S| \geq k$, sodass keine zwei Knoten aus S adjazent sind.

Die Probleme Vertex Cover und Independent Set hängen eng zusammen: Ein minimales Vertex Cover enthält genau die Knoten, die nicht in einem maximalen Independent Set liegen. Siehe hierzu auch nochmals Abbildung 2.5: Die grauen, nicht selektierten Knoten bilden jeweils ein Independent Set. Durch diesen engen Zusammenhang existiert folglich eine triviale Reduktion von Independent Set in einem Graphen $G = (V, E)$ mit Parameter k zu Vertex Cover auf einem Graphen G' mit Parameter k'. Setze hierzu schlicht $G' = G$ und $k' = |V| - k$. Diese Reduktion ist nicht parametrisiert, da k' eben von $|V|$ und somit von der Eingabe $x = G$ abhängt.

Tatsächlich liegt Independent Set auch nicht in $\mathcal{FPT}$, sondern in der Komplexitätsklasse $\mathcal{W}[1]$, welche wir im nächsten Abschnitt kennen lernen.

Die Komplexitätsklasse $\mathcal{W}[1]$

Nicht alle parametrisierten Probleme liegen in $\mathcal{FPT}$. Hierzu betrachten wir das k-Schritt-Halteproblem auf nichtdeterministischen Turingmaschinen. In der Literatur ist dieses Problem auch als Short Turing Machine Acceptance bekannt. Eine Beschreibung von Turingmaschinen ist etwa in [27] zu finden.

> k-Schritt-Halteproblem auf nichtdeterministischen Turingmaschinen:
> Gegeben: Ein Parameter $k \in \mathbb{N}$ und eine nichtdeterministische Turingmaschine M. M wird (inkl. Zustandsübergangstabelle) als Eingabewort x kodiert.
> Gefragt: Wird x von M in höchstens k Schritten akzeptiert?

Durch erschöpfende Simulation aller möglichen k-Schritt Berechnungswege der Turingmaschine kann das Problem in $O(n^{k+1})$ Zeit gelöst werden. n ist hierbei die Größe des Arbeitsalphabets der Turingmaschine und nicht beschränkt.

Das nicht parametrisierte Äquivalent des Problems mit höchstens polynomiell vielen Schritten in der Eingabelänge entspricht gerade dem generischen $\mathcal{NP}$-vollständigen Problem. Daher wäre es überraschend, wenn eine signifikante Verbesserung der hier genannten Laufzeit von $O(n^{k+1})$ - etwa hin zu einer Laufzeit von $f(k) \cdot n^{O(1)}$ - möglich wäre.

Mit dem k-Schritt-Halteproblem gelingt uns die Definition der Klasse $\mathcal{W}[1]$:

Definition 2.5. *Die Klasse aller parametrisierten Probleme, die sich parametrisiert auf das k-Schritt-Halteproblem reduzieren lassen, heißt $\mathcal{W}[1]$. Ein parametrisiertes Problem, auf das sich das k-Schritt-Halteproblem parametrisiert reduzieren lässt heißt $\mathcal{W}[1]$-*hart*. Ein $\mathcal{W}[1]$-hartes Problem heißt $\mathcal{W}[1]$-*vollständig*, falls es in $\mathcal{W}[1]$ liegt.*

Hiermit haben wir die *unterste* Klasse von nicht effizient lösbaren parametrisierbaren Problemen definiert. Analog zu den Klassen $\mathcal{P}$ und $\mathcal{NP}$ existiert die Vermutung $\mathcal{FPT} \neq \mathcal{W}[1]$. Darüber hinaus existiert eine ganze Hierarchie von parametrisierten Komplexitätsklassen $\mathcal{W}[t], t \geq 1$ mit $\mathcal{W}[t] \subseteq \mathcal{W}[t+1]$, welche von Downey und Fellows eingeführt werden [12]. Für uns relevant ist jedoch nur die Eigenschaft der $\mathcal{W}[1]$-Härte eines Problems L, da - sofern nicht $\mathcal{FPT} = \mathcal{W}[1]$ gilt - dann $L \notin \mathcal{FPT}$ gilt. Die Eigenschaft $L \in \mathcal{FPT}$ oder $L \notin \mathcal{FPT}$ wird für uns eine wichtige Rolle spielen.

2.5 Datenreduktion und Problemkerne

Aufbauend auf den bisherigen Abschnitten dieses Kapitels kommen wir zu den für diese Arbeit wichtigsten Definitionen. Ziel ist es, ein gegebenes Problem in polynomieller Zeit zu vereinfachen. Diese Vereinfachung wird über *Datenreduktionsregeln* vorgenommen:

Definition 2.6. *Sei L ein parametrisiertes Problem, x ein zugehörige Eingabe und k der zugehörige Parameter. Eine* Datenreduktionsregel *ist eine Abbildung $\phi : (x, k) \mapsto (x', k')$, welche folgende Eigenschaften erfüllt:*

- *ϕ ist polynomiell in $|(x, k)|$ berechenbar,*
- *$(x, k) \in L \Leftrightarrow (x', k') \in L$ und*
- *$|x'| \leq |x|$ sowie $|k'| \leq |k|$.*

In der Literatur ist oft der Hinweis zu finden, dass es technisch bereits genügt, wenn ϕ in $f(k) \cdot |(x, k)|^{O(1)}$ berechenbar ist und $|k'| \leq |g(k)|$ für eine beliebige berechenbare Funktion $g(k)$ gilt. Allerdings sind keine parametrisierten Probleme bekannt, bei denen diese freiere Definition benötigt werden würde. Siehe hierzu auch [23].

Ein Beispiel für eine Datenreduktionsregel bei Vertex Cover wäre das Ignorieren von Knoten mit Grad 0, da diese nichts zur Lösung beitragen können. Offensichtlich ist diese Regel korrekt und erfüllt die Anforderungen der Definition. Sind nun zu einem Problem L eine oder mehrere Datenreduktionsregeln bekannt, so erhalten wir durch die wiederholte und erschöpfende Anwendung dieser Regeln auf eine Instanz $(x, k) \in L$ eine *reduzierte Instanz* (x', k'). Hierauf aufbauend kommen wir zur Definition der *Problemkern-Reduktion*:

Definition 2.7. *Sei L ein parametrisiertes Problem. Eine* Problemkern-Reduktion *(englisch:* problem kernelization*) ist die Anwendung von Datenreduktionsregeln auf ein $(x, k) \in L$, sodass für die reduzierte Instanz (x', k') gilt: $|x'| \leq f(k)$ für eine beliebige berechenbare Funktion $f(k)$, die nur vom ursprünglichen Parameter k abhängig ist. Die Funktion $f(k)$ bezeichnet die Größe des Problemkerns.*

In einem Problemkern (englisch: *kernel*) bleibt der Teil der Eingabeinstanz übrig, den wir nicht effizient, d.h. in polynomieller Zeit, haben lösen können. Problemkerne sind nicht eindeutig, dennoch spricht man oft von dem Problemkern eines

parametrisierten Problems und meint damit insbesondere die Größe des Problemkerns. Wie wir später sehen werden, hat nicht jedes parametrisierte Problem auch einen Problemkern.

Vertex Cover ist das Standardbeispiel für ein parametrisiertes Problem mit einem Problemkern. Wir werden die zugehörigen Problemkern-Reduktionen in Kapitel 3 betrachten. Hier an dieser Stelle untersuchen wir statt dessen ein Problem mit einem trivialen Problemkern:

> Independent Set auf planaren Graphen:
> Gegeben: Ein einfacher planarer Graph $G = (V, E)$ und $k \in \mathbb{N}_0$.
> Gesucht: Eine Knotenmenge $S \subseteq V$ mit $|S| \geq k$, sodass keine zwei Knoten aus S adjazent sind.

Das nachfolgende Ergebnis ist ein gutes Beispiel, da wir im Vergleich zu anderen Problemkern-Reduktionen keine komplizierten Reduktionsregeln betrachten müssen. Durch den Vier-Farben-Satz und Robertson *et al.* [26] ist bekannt, dass die $n = |V|$ Knoten eines planaren Graphen in polynomieller Zeit derart in vier Teilmengen aufgeteilt werden können (d.h. *gefärbt* werden können), dass innerhalb einer Teilmenge keine Knoten adjazent sind. Diese vier Teilmengen bilden also jeweils ein Independent Set. Und da alle n Knoten auf die vier Teilmengen verteilt sind, muss es eine Teilmenge geben, die mindestens ein Viertel aller Knoten des Graphen enthält. Die Problemkern-Reduktion funktioniert dann wie folgt: Gilt für den Parameter $k \leq \lceil n/4 \rceil$, so ist die Antwort auf das Entscheidungsproblem „ja“ und die gesuchte Knotenmenge S lässt sich mit dem Algorithmus von Robertson *et al.* in polynomieller Zeit finden. Gilt hingegen $k > \lceil n/4 \rceil$, so folgt $n < 4k$ und die Eingabeinstanz selbst ist bereits ein Problemkern der Größe $4k$. In diesem Beispiel handelt es sich also um einen linearen Problemkern.

Ein technisches Detail muss zu diesem Beispiel noch besprochen werden: Wir haben das Problem im Falle $k \leq \lceil n/4 \rceil$ direkt entschieden anstatt es definitionsgemäß auf eine andere reduzierte Instanz abzubilden. Gelöst wird diese Ab-

weichung von der Definition dadurch, dass wir uns für das gegebene Problem je eine triviale „ja“-Instanz und „nein“-Instanz konstanter Größe suchen. Anstatt direkt zu entscheiden die Eingabe auf diese „ja“- bzw. „nein“-Instanz reduzieren. Wie Bodlaender [4] erwähnt, folgt daraus, dass ein Problem aus $\mathcal{P}$ einen Problemkern der Größe $O(1)$ besitzt. Ebenso gilt, dass ein Problem mit einem $O(1)$-Problemkern zu $\mathcal{P}$ gehört: Zunächst berechnen wir den Problemkern und sei dessen Größe höchstens c. Dann überprüfen wir, ob der Problemkern in der Menge aller „ja“-Instanzen mit Maximalgröße c enthalten ist und können das Problem somit entscheiden. Die Menge aller „ja“-Instanzen mit Maximalgröße c ist unabhängig von der Eingabe und kann somit in konstanter Zeit vorberechnet werden. Folglich gilt für ein $\mathcal{NP}$-hartes Problem, dass es keinen Problemkern der Größe $O(1)$ haben kann, sofern nicht $\mathcal{P} = \mathcal{NP}$ gilt.

2.6 Die Klasse $\mathcal{FPT}$ und die Existenz von Problemkernen

Nachdem uns nun die grundlegenden Definitionen dieser Arbeit bekannt sind, bringen wir sie noch in Zusammenhang mit der Klasse $\mathcal{FPT}$:

Satz 2.8. *Sei L ein parametrisiertes Problem, so gilt: $L \in \mathcal{FPT}$ genau dann, wenn L entscheidbar ist und einen Problemkern besitzt.*

Dieser Satz hat entscheidende Bedeutung für die vorliegende Diplomarbeit. Wir werden im Kapitel 4 eine Parametrisierung des Problems Minimum Outdegree Orientation betrachten und nachweisen, dass es bezüglich zweier Parameter in $\mathcal{FPT}$ liegt. Hieraus folgt direkt die Existenz eines Problemkerns für Minimum Outdegree Orientation.

Bodlaender [4] weist darauf hin, dass in der Literatur die Bedingung der Entscheidbarkeit des Problems oft vergessen wird. Diese Bedingung ist allerdings notwendig, wie folgendes Beispiel zeigt: Sei X unentscheidbar und das Problem $L = \{(x, k) \mid k \in X\}$, so lässt sich mit $(x, k) \mapsto (\varepsilon, k)$ ein trivialer linearer

Problemkern angeben. Gleichzeitig ist L unentscheidbar und aus der Unentscheidbarkeit folgt $L \notin \mathcal{FPT}$.

Der nachfolgende Beweis zu Satz 2.8 stammt von Flum und Grohe [17]:

Beweis zu Satz 2.8. Sei nun L entscheidbar und habe einen Problemkern. Folgenden Algorithmus können wir dann für L verwenden: Sei (x, k) die Eingabe der Größe n, so können wir über die Problemkern-Reduktion in $p(n)$ Zeit für ein Polynom p eine reduzierte Instanz (x', k') mit $|(x', k')| \leq f(k)$ erzeugen, wobei f eine beliebige berechenbare Funktion ist. Da L entscheidbar ist, gibt es einen Algorithmus, der (x', k') lösen kann. Sei die Laufzeit hiervon durch die beliebige berechenbare Funktion g beschränkt. Somit ist die Gesamtlaufzeit $p(n) \cdot g(f(k))$ und folglich gilt $L \in \mathcal{FPT}$.

Sei nun also $L \in \mathcal{FPT}$. Folglich ist L entscheidbar und es gibt einen Algorithmus, der Instanzen von L mit der Größe n in $f(k) \cdot n^c$ lösen kann, wobei c eine positive Konstante und f eine beliebige berechenbare Funktion ist. Diesen Algorithmus lassen wir n^{c+1} Schritte laufen. Hat der Algorithmus das Problem in dieser Zeit gelöst, so geben wir eine „ja“- bzw. „nein“-Instanz konstanter Größe als Problemkern an. Hat der Algorithmus das Problem hingegen nicht in n^{c+1} Schritten lösen können, so gilt:

$$n^{c+1} < f(k) \cdot n^c \Rightarrow n < f(k)$$

Folglich ist die Eingabeinstanz selbst bereits ein Problemkern. □

Durch diesen Satz haben wir den Zusammenhang zwischen der Klasse $\mathcal{FPT}$ und Problemkernen hergestellt. Zwar ist der Satz nicht geeignet, möglichst kleine Problemkerne zu finden, verwenden können wir ihn allerdings andersherum: Wenn ein Problem nicht in $\mathcal{FPT}$ liegt, so wissen wir auch, dass es keine Problemkern-Reduktion zu diesem Problem geben kann. Sofern nicht $\mathcal{FPT} = \mathcal{W}[1]$ gilt, reicht also der Nachweis dass ein Problem $\mathcal{W}[1]$-hart ist, um die Suche nach einem

Problemkern abbrechen zu können. Beispielsweise ist, wie bereits erwähnt, Independent Set (auf allgemeinen Graphen) $\mathcal{W}[1]$-vollständig und hat somit keinen Problemkern. Gleichzeitig haben wir im vorherigen Abschnitt gezeigt, dass Independent Set auf planaren Graphen einen Problemkern hat und folglich in $\mathcal{FPT}$ liegt.

3 Präsentation ausgewählter Techniken an Vertex Cover

In diesem Kapitel betrachten wir das Problem Vertex Cover in parametrisierter Variante eingehender und besprechen einige ausgewählte Techniken zur Problemkern-Reduktion. Zur Erinnerung noch einmal die parametrisierte Problemdefinition:

> Vertex Cover mit höchstens k Knoten (kurz: VC(k)):
> Gegeben: Ein einfacher Graph $G = (V, E)$ und $k \in \mathbb{N}_0$.
> Gesucht: Eine Knotenmenge $S \subseteq V$ mit $|S| \leq k$, sodass jede Kante aus E mindestens einen ihrer beiden Endknoten in S hat.

Die unparametrisierte Variante von Vertex Cover sucht eine Knotenmenge S mit $|S|$ minimal. Karp [18] zeigte 1972 bereits, dass die unparametrisierte Variante von Vertex Cover $\mathcal{NP}$-vollständig ist. Daraus folgt, dass auch die hier betrachtete Variante VC(k) $\mathcal{NP}$-vollständig ist.

In den folgenden Abschnitten werden wir verschiedene Problemkern-Reduktionen zu VC(k) sehen, somit gilt nach Satz 2.8, dass VC(k) $\in \mathcal{FPT}$ ist.

3.1 Problemkern-Reduktion nach Buss

Die nachfolgende Vorgehensweise zur Erzeugung eines Problemkerns für VC(k) stammt von Buss [6] und wird von Guo und Niedermeier [16] beschrieben. Es handelt sich hierbei um die drei nachfolgenden Datenreduktionsregeln. Zu beachten

ist, dass bei Datenreduktion, wie auch hier, in der Regel Lösungen ausgeschlossen werden. Wir versuchen mit nachfolgenden Regeln eine der Lösungen für VC(k) zu konstruieren, von denen es jedoch mehrere oder auch keine geben könnte.

Datenreduktionsregel Nr. 1: Knoten mit Grad 0 werden aus dem Graphen entfernt, k bleibt wie es ist. Die Korrektheit der Regel ist trivial, da Knoten vom Grad 0 niemals zur Lösung beitragen können. Ein Beispiel der Datenreduktionsregel ist in Abbildung 3.1 dargestellt.

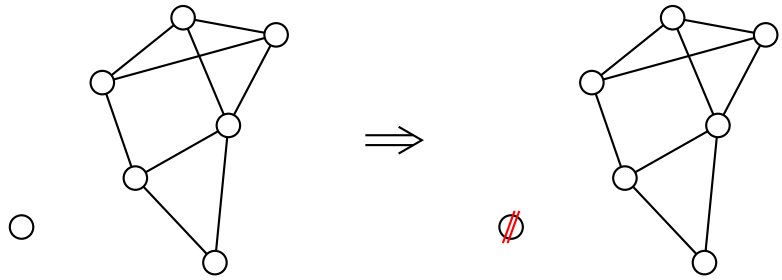


Abbildung 3.1: Vertex-Cover-Datenreduktionsregel Nr. 1: Entferne alle Knoten vom Grad 0.

Datenreduktionsregel Nr. 2: Für jeden Knoten v mit $grad(v) = 1$ wählen wir den einzigen Nachbarknoten u und fügen u der Lösungsmenge S hinzu. Wir verringern entsprechend k um 1. Die Knoten u und v sowie alle zu u inzidenten Kanten werden aus dem Graphen entfernt.

Diese Regel ist korrekt, weil die Kante $\{u, v\}$ abgedeckt werden muss. Folglich muss u oder v in der Lösungsmenge enthalten sein. Da v aber nur den Grad 1 hat, also nur die eine Kante $\{u, v\}$ abdecken könnte, wählen wir statt dessen u, da u möglicherweise einen höheren Grad hat und somit noch weitere Kanten abdecken könnte. Durch das Hinzufügen von u zu S wird S um ein Element größer, folglich können wir nur noch $k' = k - 1$ weitere Knoten wählen. Alle zu u inzidenten Kanten können nun entfernt werden, da sie durch u abgedeckt werden. Der Knoten v hat somit den Grad 0 und kann nach Datenreduktionsregel Nr. 1 ebenfalls entfernt werden. Eine beispielhafte Darstellung dieser Datenreduktionsregel ist in Abbildung 3.2 zu finden.

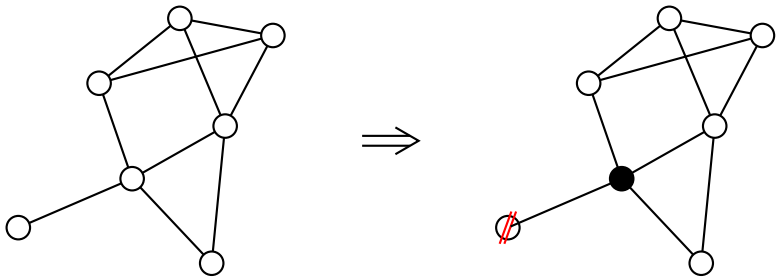


Abbildung 3.2: VERTEX-COVER-Datenreduktionsregel Nr. 2: Entferne alle Knoten vom Grad 1 und füge den jeweiligen Nachbarn zur Lösung hinzu.

Datenreduktionsregel Nr. 3: Für jeden Knoten v mit $grad(v) \geq k+1$ nehmen wir v in S auf, und entfernen v, sowie alle zu v inzidenten Kanten aus dem Graphen. Wir verringern k um 1.

Klar ist, dass wir durch das Hinzufügen von v zu S die Anzahl der noch wählbaren Knoten um eins verringern müssen, also $k' = k - 1$. Ebenso können wir alle zu v inzidenten Kanten und v selbst aus dem Graphen entfernen, wenn wir v zu S hinzufügen. Die Korrektheit der Wahl von v folgt aus $grad(v) \geq k + 1$: Da es sich beim vorliegenden Graphen nach Voraussetzung um einen einfachen Graphen handelt, sind Mehrfachkanten ausgeschlossen. Folglich hat v mindestens $k+1$ verschiedene Nachbarknoten. Würden wir v nun nicht in S aufnehmen, müssten die mindestens $k+1$ verschiedenen Nachbarn zu S hinzugefügt werden, um alle zu v inzidenten Kanten abzudecken. Dies ist allerdings nicht möglich, da somit $|S| > k$ gelten würde. Die Datenreduktionsregel ist in Abbildung 3.3 beispielhaft dargestellt.

Folgerung aus den drei Regeln: Alle drei Datenreduktionsregeln sind in polynomieller Zeit (bezüglich der Speicherungsgröße des Graphen) durchführbar. Zu Beachten ist, dass sich durch Regel 2 und 3 das k jeweils verringert. Die Regeln basieren stets auf der Verwendung des aktuellen, möglicherweise bereits verringerten k. So kann es vorkommen, dass Regel 3 erst anwendbar ist, wenn durch Regel 2 der Wert von k hinreichend verringert worden ist. Dies ist natürlich ab-

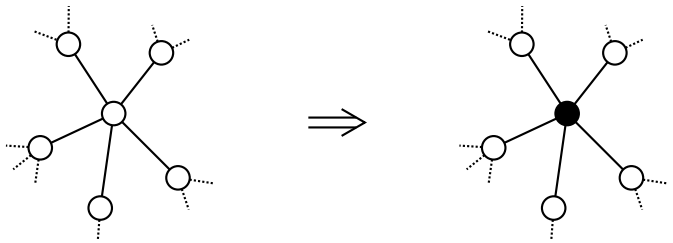


Abbildung 3.3: Vertex-Cover-Datenreduktionsregel Nr. 3: Nimm alle Knoten mit Grad $\geq k+1$ in die Lösung auf. In diesem Beispiel ist $k = 4$ und der mittlere Knoten hat den Grad 5, also kann die Regel angewandt werden.

hängig vom vorliegenden Graphen. Insbesondere erkennt man auch, dass es sich bei den Datenreduktionsregeln 1 und 2 um *parameterunabhängige Datenreduktionsregeln* handelt, während die Regel 3 abhängig vom Parameter k ist und somit als *parameterabhängige Datenreduktionsregel* bezeichnet wird.

Nachdem diese drei Regeln erschöpfend angewendet worden sind, hat der Graph nur noch Knoten v mit $2 \leq grad(v) \leq k$, wobei k möglicherweise durch die Regeln verringert worden ist. Aus $grad(v) \leq k$ folgt, dass jeder Knoten im verbleibenden Graphen höchstens k Kanten abdecken kann. Gleichzeitig können höchstens k Knoten in S aufgenommen werden. Somit folgt, dass höchstens k^2 Kanten von einer Lösung des verbleibenden Graphen abgedeckt werden können. Hat der verbleibende Graph mehr als k^2 Kanten, so kann es keine Lösung von VC(k) geben. Da jeder Knoten aber mindestens den Grad 2 hat, folgt aus der Feststellung, dass der verbleibende Graph höchstens k^2 Kanten hat, direkt, dass der verbleibende Graph auch nur höchstens k^2 Knoten haben kann.

Folglich ist der verbleibende Graph ein Problemkern der Größe k^2 Knoten und k^2 Kanten. Dies ist jedoch noch nicht das Optimum. In den folgenden Abschnitten werden wir Problemkern-Reduktionen von Vertex Cover kennen lernen, die einen Problemkern linearer Größe erzeugen. Bevor wir uns aber diesen Reduktionen zuwenden, betrachten wir noch zwei weitere Datenreduktionsregeln im

Stil der obigen drei. Durch die nun folgenden beiden Regeln, welche von Abu-Khzam et. al [1] vorgestellt werden, kann keine grundsätzliche Verbesserung in der Größenordnung des Problemkerns erreicht werden. Der Problemkern hat dann ebenfalls noch die Größe k^2 Knoten und k^2 Kanten. Allerdings sind die beiden nachfolgenden Regeln für die Praxis relevant, wie wir in Abschnitt 3.4 sehen werden. Zudem bilden sie ein gutes Beispiel, wie die bisherigen Reduktionsideen weiter ausgeschöpft werden können.

Datenreduktionsregel Nr. 4: Diese Regel wird auf alle Knoten mit Grad 2 angewendet, deren Nachbarn adjazent sind. Sei hierzu v ein Knoten mit $grad(v) = 2$ und seien u und w dessen Nachbarn, welche ebenfalls durch eine Kante verbunden sind. In diesem Fall nehmen wir u und w in die Lösungsmenge S auf und entfernen die Knoten u, v und w sowie alle zu ihnen inzidenten Kanten aus dem Graphen. Zudem verringern wir k um 2.

Diese Datenreduktionsregel ist ähnlich zu Regel Nr. 2. Zwischen den Knoten u, v und w befinden sich die Kanten $\{u, v\}$, $\{u, w\}$ und $\{v, w\}$. Um diese Kanten abzudecken, müssen zwei der drei Knoten u, v und w in S aufgenommen werden. Wählt man u und w, so können durch beide Knoten zudem jeweils noch eventuell existierende weitere Kanten mitabgedeckt werden, während eine Aufnahme von v in S lediglich die beiden inzidenten Kanten abdecken würde. Die Wahl von u und w deckt also potentiell mehr Kanten ab und ist daher zu bevorzugen. Durch die Aufnahme zweier Knoten in S muss $k' = k - 2$ gesetzt werden. Diese Knoten sowie adjazente Kanten können dann entfernt werden. Zudem kann wegen Datenreduktionsregel Nr. 1 auch v entfernt werden. In Abbildung 3.4 wird die Datenreduktionsregel Nr. 4 beispielhaft dargestellt.

Datenreduktionsregel Nr. 5: Die 5. Datenreduktionsregel beschäftigt sich ebenfalls mit Knoten vom Grad 2. Im Gegensatz zu Regel Nr. 4 sind in der hier betrachteten Situation aber die beiden Nachbarknoten nicht adjazent. Gegeben ist also ein Knoten v mit einem Nachbarn u und einem Nachbarn w, $grad(v) = 2$, $grad(u) \geq 2$, $grad(w) \geq 2$ und es gibt keine Kante zwischen u und w. In diesem

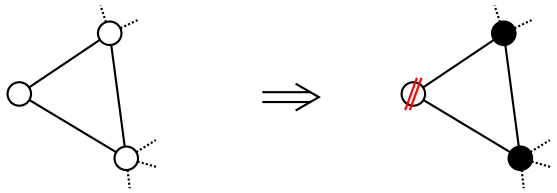


Abbildung 3.4: Vertex-Cover-Datenreduktionsregel Nr. 4: Sind die Nachbarn eines Knotens mit Grad 2 adjazent, so werden die beiden Nachbarn in S aufgenommen und alle drei Knoten sowie inzidente Kanten entfernt.

Fall können die Knoten u, v und w zu einem neuen Knoten v' knotenverschmolzen werden und k wird um 1 verringert. Ein Beispiel der Datenreduktionsregel ist in Abbildung 3.5 dargestellt.

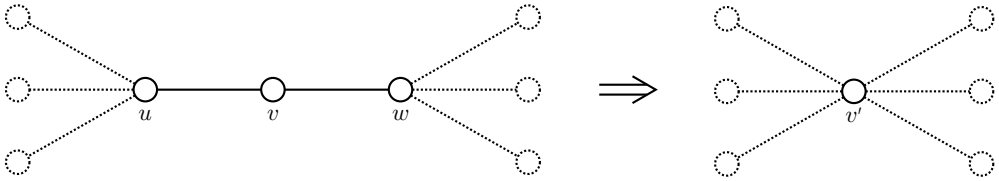


Abbildung 3.5: Vertex-Cover-Datenreduktionsregel Nr. 5: Gegeben ist ein Knoten v mit Nachbarn u und w, $grad(v) = 2$, $grad(u) \geq 2$, $grad(w) \geq 2$ und w ist nicht mit u benachbart, so können die Knoten u, v und w zu einem neuen Knoten v' knotenverschmolzen werden.

Die Korrektheit zeigt sich durch folgende Überlegung: Angenommen einer der beiden Nachbarn von v wird in die Lösungsmenge mit aufgenommen und daher entfernt, so hat v nur noch einen Nachbarn und kann daher nach Datenreduktionsregel Nr. 2 ebenfalls entfernt werden. In diesem Fall befinden sich also beide Nachbarn von v in der Lösung. Anderenfalls, wenn keiner der beiden Knoten u und w in die Lösung aufgenommen wird, so muss v in die Lösung aufgenommen werden um die Kanten $\{u, v\}$ und $\{v, w\}$ abzudecken. Welcher der beiden Fälle eintritt, lässt sich zu diesem Zeitpunkt allerdings noch nicht bestimmen. Aber am Ende soll vom Algorithmus der verschmolzene Knoten v' in die Lösung

aufgenommen werden oder nicht. An dieser Stelle unterbrechen wir die Ausführung und ersetzen v' wieder durch die Ausgangssituation und wir können nun die Entscheidung vornehmen, welche der beiden genannten Fälle zutrifft.

Wenn v' kein Teil der Lösung werden soll, so bedeutet das, dass alle zu v' inzidenten Kanten durch andere Knoten abgedeckt wurden. In diesem Fall müssen die Knoten u und w nicht zur Lösung hinzu genommen werden, da es ausreicht den Knoten v hinzuzunehmen um die beiden Kanten $\{u, v\}$ und $\{v, w\}$ abzudecken. In diesem Fall ist $k' = k - 1$ also korrekt.

Soll jedoch v' in die Lösungsmenge aufgenommen werden, so werden zumindest einige inzidente Kanten nur durch v' abgedeckt. Entsprechend muss mindestens einer der Knoten u oder w in die Lösung aufgenommen werden, um diese Kanten abzudecken. Nach der obigen Überlegung wird dann allerdings auch der andere der beiden Knoten zur Lösung hinzugefügt und v wird gestrichen. Auch in diesem Fall ist $k' = k-1$ korrekt, denn es kommen zwar am Ende zwei Knoten (u und w) zur Lösung hinzu, aber v' wird im Graphen per Knotenverschmelzung erstellt und muss für diesen Fall auch ausgewählt werden.

Die beiden Fälle werden in Abbildung 3.6 visualisiert.

Durch Anwendung der Regeln 4 und 5 ist im Graph nun für alle Knoten der Grad mindestens 3. Weitere Datenreduktionsregeln für Knoten mit höherem Grad sind ebenfalls denkbar. Eine Bewertung der Nützlichkeit dieser Regeln ist im Abschnitt 3.4 durchgeführt.

3.2 Problemkern-Reduktion nach Nemhauser-Trotter

Nemhauser und Trotter haben 1975 Ergebnisse im Bereich approximativer Algorithmen erzielt [22], welche sich nun ebenfalls als gute Problemkern-Reduktionsalgorithmen erwiesen haben. In diesem Abschnitt werden wir zwei Möglichkeiten

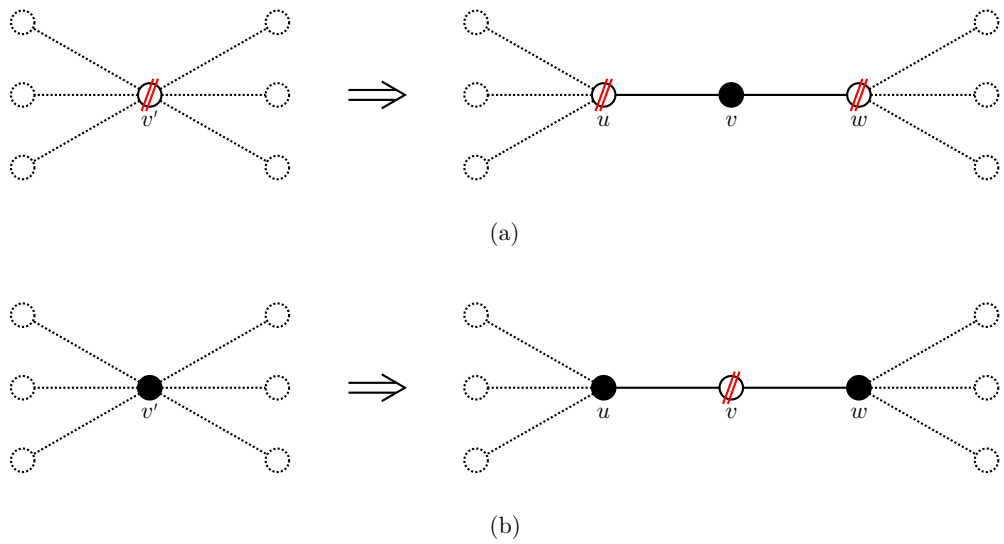


Abbildung 3.6: Fallunterscheidung bei Datenreduktionsregel Nr. 5: Soll v' nicht in die Lösung aufgenommen werden, so muss v aufgenommen werden und u sowie w können gestrichen werden. Würde hingegen v' zur Lösung gehören, so müssen u und w in die Lösungsmenge gesetzt werden und v wird gestrichen.

betrachten, wie ein Problemkern der Größe $2k$ auf Basis der Arbeit von Nemhauser und Trotter gefunden werden kann. Niedermeier [24] stellt diese vor.

Problemkern-Reduktion durch maximales Matching

Ein *Matching*, auch Paarung oder unabhängige Kantenmenge genannt, in einem Graphen $G = (V, E)$ ist eine Kantenmenge ohne adjazente Kanten. Das bedeutet, ein Knoten aus V kann höchstens zu einer Kante des Matchings inzident sein. Ein *maximales Matching* ist ein Matching in einem Graphen mit der größtmöglichen Anzahl von im Matching enthaltenen Kanten. Für die im Folgenden konstruierte Problemkern-Reduktion sind wir an maximalen Matchings in bipartiten Graphen interessiert, Abbildung 3.7 stellt dieses beispielhaft dar.

In einem bipartiten Graphen kann ein maximales Matching durch Algorithmen für den maximalen Fluss gefunden werden, siehe Schöning [27] für eine Einfüh-

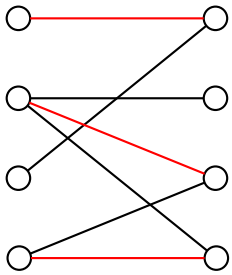


Abbildung 3.7: Beispiel eines maximalen Matchings auf einem bipartiten Graphen.

rung. Abu-Khzam et. al [1] schlagen hierfür insbesondere die Nutzung des Algorithmus von Dinic [10] vor. Der Algorithmus verbessert den bekannten Ford-Fulkerson-Algorithmus [14].

Die Problemkern-Reduktion für VC(k) auf einem Graphen $G = (V, E)$ auf Basis des maximalen Matchings verläuft nun wie folgt: Zunächst wird für den Graphen G ein bipartiter Graph $B = (V \cup V', E_B)$ erstellt. V' ist hierbei eine Kopie von V, also $V' = \{v' | v \in V\}$. Die Kantenmenge wird wie folgt gesetzt: $E_B = \{\{x, y'\}, \{x', y\} | \{x, y\} \in E\}$. Für jede Kante $\{x, y\} \in E$ werden also die zwei Kanten $\{x, y'\}, \{x', y\}$ in E_B eingefügt. Abbildung 3.8 zeigt diese Konstruktion an einem Beispiel.

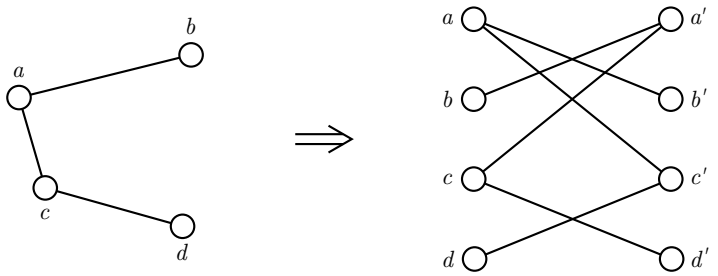


Abbildung 3.8: Beispiel für die Konstruktion des bipartiten Graphen.
Die Knoten werden dupliziert und für jede Kante $\{x, y\}$ im Ausgangsgraphen werden die Kanten $\{x, y'\}, \{x', y\}$ im zu erstellenden bipartiten Graphen eingefügt.

In diesem bipartiten Graphen B wird nun mit einem Algorithmus für den maximalen Fluss ein maximales Matching gefunden. Mit Hilfe des Satzes von König [21] kann über dieses maximale Matching in polynomieller Zeit ein Vertex Cover C_B für den Graphen B gefunden werden. Hiermit lassen sich zwei Knotenmengen definieren:

$$C_0 = \{x \in V | x \in C_B \text{ und } x' \in C_B\}$$

sowie

$$V_0 = \{x \in V | \text{ entweder } x \in C_B \text{ oder } x' \in C_B\}$$

Diejenigen Knoten $x \in V$, für die weder $x \in C_B$ noch $x' \in C_B$ gilt, werden an dieser Stelle nicht weiter betrachtet.

Nun gilt der folgende Satz, welcher auf Nemhauser und Trotters Arbeit zurückgeht, den wir hier aber nicht beweisen werden. Ein Beweis ist bei Niedermeier [24] zu finden.

Satz 3.1. *Sei $G = (V, E)$ der zu untersuchende Graph und C_0 sowie V_0 die oben genannten Knotenmengen. Es gilt: Ist C_1 ein minimales Vertex Cover des induzierten Teilgraphen $G[V_0]$, so ist $|C_1| \geq |V_0|/2$ und $C = C_0 \cup C_1$ ist ein minimales Vertex Cover von G.*

Uns gelingt es durch diesen Algorithmus mit C_0 eine Knotenmenge zu identifizieren, die sicher in einem minimalen Vertex Cover C für G enthalten ist. Zudem müssen alle zum minimalen Vertex Cover C fehlenden Knoten in der Menge V_0 enthalten sein, und mindestens die Hälfte aller Knoten aus V_0 muss in C enthalten sein. Über diese Eigenschaften lässt sich eine Problemkern-Reduktion für VC(k) durchführen:

Wir erzeugen, wie oben beschrieben, die beiden Mengen C_0 und V_0. Nun ist C_0 ein Teil der Lösung, somit ziehen wir diese Knoten von k ab, setzen also $k' = k - |C_0|$.

Gilt nun $|V_0| > 2 \cdot k'$, so kann es nach Satz 3.1 keine Lösung für VC(k) geben, da mindestens die Hälfte aller Knoten aus V_0 noch in die gesuchte Lösung aufzunehmen wären. Folglich gilt also $|V_0| \leq 2 \cdot k' \leq 2 \cdot k$, und somit ist der induzierte Teilgraph $G[V_0]$ ein Problemkern der Größe $2k$.

Die Laufzeit dieser Problemkern-Reduktion für VC(k) auf dem betrachteten Graphen $G = (V, E)$ wird von Niedermeier mit $O(k \cdot |V| + k^3)$ angegeben. Diese Laufzeit kommt zu Stande, indem der Graph G zunächst in $O(k|V|)$ Zeit durch die Problemkern-Reduktion nach Buss auf $O(k^2)$ Kanten und $O(k^2)$ Knoten reduziert wird. Die Laufzeit der Problemkern-Reduktion durch maximales Matching wird im Wesentlichen durch das Finden des maximalen Matchings bestimmt. Der Algorithmus von Dinic vermag dies in $O(\sqrt{|V|} \cdot |E|)$ Zeit durchzuführen, in unserem Fall also in $O(\sqrt{k^2} \cdot k^2) = O(k^3)$ Zeit.

Problemkern-Reduktion durch lineare Programmierung

Das eben gezeigte Ergebnis, dass VC(k) einen Problemkern der Größe $2k$ besitzt, lässt sich auch durch *lineare Programmierung* zeigen, was auch der ursprüngliche Anwendungsbereich der Arbeit von Nemhauser und Trotter war. Unter linearer Programmierung, oft auch *lineare Optimierung* genannt, versteht man das Optimieren einer linearen Funktion auf einer Menge. Diese Menge ist wiederum durch eine Anzahl von linearen Gleichungen oder Ungleichungen eingeschränkt. Der Begriff der Programmierung geht hierbei auf die Planung und die Aufstellung von Tabellen zurück und hat nichts mit dem Programmieren in einer Programmiersprache zu tun. Die bei der linearen Programmierung zu lösenden *linearen Programme (LP)* sind in polynomieller Zeit lösbar, siehe [22].

Bei der *ganzzahligen linearen Programmierung* (englisch: *integer linear programming*, ILP) schränkt man die Variablen dahingehend ein, dass sie nur ganzzahlige Werte annehmen dürfen. Ganzzahlige Lineare Programmierung ist $\mathcal{NP}$-vollständig, wie Karp [18] zeigt.

Das Optimierungsproblem Vertex Cover kann für einen Graphen $G = (V, E)$ als *Ganzzahliges Lineares Programm (ILP)* formuliert werden. Hierzu legen wir für jeden Knoten $v \in V$ eine Variable $x_v \in \{0, 1\}$ an. Hierbei bedeutet $x_v = 1$, dass der Knoten v in einem minimalen Vertex Cover C des Graphen G enthalten ist, während $x_v = 0$ heißt, dass er nicht in der Lösung liegt. Vertex Cover kann nun als folgendes Optimierungsproblem ausgedrück werden:

$$\text{Vertex Cover} \begin{cases} (1)\ x_v \in \{0,1\} \ \ \forall\, v \in V \\ (2)\ x_u + x_v \geq 1 \ \ \forall\, \{u,v\} \in E \\ (3)\ \text{Minimiere} \ \ \sum_{v \in V} x_v \end{cases}$$

Wie bereits erwähnt sind ILPs ebenfalls $\mathcal{NP}$-vollständig. Daher ändern wir im Folgenden die Bedingung (1) zu $x_v \geq 0 \ \ \forall\, v \in V$ und erzeugen so aus dem ILP ein in polynomieller Zeit lösbares LP. Dieses LP ist nun lediglich eine Näherung an die Lösung. Genauer: Der Lösungswert des LP ist eine untere Schranke für das ILP.

Wir definieren nun drei Knotenmengen:

$$P = \{v \in V | x_v > 0{,}5\}$$
$$Q = \{v \in V | x_v = 0{,}5\}$$
$$R = \{v \in V | x_v < 0{,}5\}$$

Nun gilt der folgende, auf Nemhauser und Trotter zurückgehende Satz:

Satz 3.2. *Sei $G = (V, E)$ der zu untersuchende Graph und die Knotenmengen P, Q und R wie oben definiert, so gilt: Es existiert ein minimales Vertex Cover C von G mit $P \subseteq C$ und $C \cap R = \emptyset$.*

Wir zeigen den Beweis in der Variante von Abu-Khzam et al. [1]:

Beweis zu Satz 3.2. Sei C dasjenige Vertex Cover, welches durch die Lösung des ILPs erstellt wird. Sei A eine Knotenmenge mit $A \subseteq P$, welche nicht C enthalten ist. Weiter sei B eine Knotenmenge mit $B \subseteq R$ und $B \subseteq C$. Wegen der Bedingung (2) gilt, dass die Nachbarn $N(r) \subseteq P \ \ \forall\, r \in R$ sind. Wir zeigen zunächst $|A| = |B|$:

Annahme: $|A| < |B|$. Wegen $N(r) \subseteq P \ \ \forall\, r \in R$ können wir die Knoten B in C durch die Knoten A ersetzen, ohne dass Kanten hierdurch nicht mehr abgedeckt werden. Wegen $|A| < |B|$ folgt dann jedoch, dass C nicht optimal war, ein Widerspruch zur Voraussetzung.

Annahme: $|A| > |B|$. Setze $\varepsilon = \min\{x_v - 0{,}5 | v \in A\}$ und ersetze x_v für alle $v \in B$ durch $x_v + \varepsilon$ und für alle $v \in A$ durch $x_v - \varepsilon$. Wegen $|A| > |B|$ entsteht durch diese neuen Werte für x_v eine kleinere Lösung des LP als die bestehende. Dies wäre ein Widerspruch zur Voraussetzung.

Folglich gilt $|A| = |B|$. In diesem Fall können wir wegen $N(r) \subseteq P \ \ \forall\, r \in R$ die Knoten B aus C durch die Knoten A ersetzen und erhalten ein minimales Vertex Cover mit den gesuchten Eigenschaften. □

Die Argumentation bezüglich der Größe des Problemkerns ist ähnlich wie im Fall des maximalen Matchings. Tatsächlich gilt auch hier, dass ein Vertex Cover des induzierten Teilgraphen $G[Q]$ mindestens $|Q|/2$ Knoten enthalten muss. Wäre dem nicht so, wäre die Lösung des LP nicht optimal gewesen, wie auch Niedermeier [24] beschreibt. Analog zum obigen Fall gilt daher für VC(k):

Die Knotenmenge P ist Teil der Lösung, also ziehen wir diese Knoten von k ab. Wir setzen also $k' = k - |P|$. Gilt nun $|Q| > 2 \cdot k'$, so kann es keine Lösung für VC(k) geben, da mindestens die Hälfte aller Knoten aus Q noch in die gesuchte Lösung aufzunehmen sind. Folglich gilt also $|Q| \leq 2 \cdot k' \leq 2 \cdot k$, und somit ist der induzierte Teilgraph $G[Q]$ ein Problemkern der Größe $2k$.

3.3 Problemkern-Reduktion durch Kronen

In diesem Abschnitt werden wir eine Problemkern-Reduktion kennen lernen, die lokale strukturelle Eigenschaften des betrachteten Graphen ausnutzt. Erinnern wir uns hierzu zurück an die Datenreduktionsregel Nr. 2 und 4 nach Buss. In beiden Fällen wurden die Nachbarknoten in das Vertex Cover aufgenommen, da die Nachbarn potentiell weitere Kanten abdecken konnten. Dieses Konzept wurde von Chor et al. [8] untersucht und zur Problemkern-Reduktion auf Basis von sogenannten *Kronen* ausgebaut.

Folgende Idee liegt dem zu Grunde: Sei I eine unabhängige Knotenmenge im Graphen, also eine Knotenmenge, in der keine Knoten adjazent sind. Sei N die Menge aller Nachbarn zu den Knoten aus I. Wir betrachten den Teilgraphen B, der nur aus den Knoten I und N und den Kanten zwischen diesen Knotenmengen besteht: Dieser Teilgraph ist bipartit. Angenommen das maximale Matching in diesem bipartiten Teilgraphen habe die Größe $|N|$, so gilt wegen des Satzes von König [21], dass ein minimales Vertex Cover von B ebenfalls die Größe $|N|$ haben muss. In diesem Fall können wir gerade die Knotenmenge N in das Vertex Cover aufnehmen und erreichen so, dass potentiell weitere Kanten (in G) abgedeckt werden, als wenn wir Knoten aus der Menge I im Vertex Cover verwenden würden. Diese zusätzlich abgedeckten Kanten sind gerade die Kanten zu den Nachbarn der Knoten aus N, die nicht in I liegen. Die Datenreduktionsregeln Nr. 2 ist eine direkte Konsequenz aus dieser Überlegung.

Die Definition von Kronen folgt nun dieser Überlegung:

Definition 3.3. *Eine* Krone *in einem Graphen $G = (V, E)$ besteht aus zwei Knotenmengen $I \subseteq V$ und $N \subseteq V$, für die folgende Eigenschaften gelten:*

- $I \cap N = \emptyset$
- $N = \{N(v) | v \in I\}$
- *I ist eine unabhängige Knotenmenge, das heißt:* $\forall v, u \in I : \{v, u\} \notin E$

- *Der bipartite Teilgraph B, bestehend aus den Knoten aus I und N, sowie den Kanten zwischen diesen beiden Knotenmengen, hat ein maximales Matching der Größe $|N|$.*

Zwei Beispiele für Kronen sind in Abbildung 3.9 dargestellt.

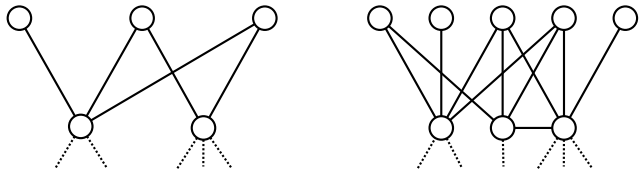


Abbildung 3.9: Zwei Beispiele für Kronen.
Die oberen Knoten sind jeweils die Knotenmenge I, die unteren Knoten bilden die Knotenmenge N. Diese Beispiele zeigen zudem, warum diese Struktur Krone genannt wird.

Die obige Diskussion zeigt, dass folgende Datenreduktionsregel für VC(k) auf dem Graphen $G = (V, E)$ angewendet werden kann: Wenn es eine Krone (I, N) in G gibt, so entferne die Knotenmengen I und N sowie inzidente Kanten aus G und setze $k' = k - |N|$.

Sowohl Niedermeier [24] als auch Abu-Khzam et al. [1] beschreiben nun Algorithmen, die auf Basis der Berechnung zweier spezieller maximalen Matchings solche Kronen in polynomieller Zeit finden. Ein Abbruchkriterium für die Suche des Vertex Cover ist, dass eines dieser maximalen Matchings größer als k ist. In diesem Fall kann nach dem Satz von König kein Vertex Cover der Größe $\leq k$ in G existieren. Dieses kann ausgenutzt werden um einen Problemkern zu konstruieren. Hierbei wird eine derart große Krone (I, N) konstruiert, dass lediglich maximal $3k$ Knoten nicht in ihr enthalten sind, wenn der Graph ein Vertex Cover der Größe $\leq k$ haben sollte. Durch das Entfernen dieser Krone (I, N) entsteht der induzierte Teilgraph $G[V \setminus (I \cup N)]$, welcher höchstens noch $3k$ Knoten besitzt und somit einen Problemkern dieser Größe darstellt.

3.4 Bewertung und Vergleich der Problemkern-Reduktionen

In den vorhergehenden Abschnitten wurden verschiedene Techniken der Problemkern-Reduktion für das Problem VC(k) vorgestellt. Vertex Cover ist ein in der Literatur sehr ausführlich untersuchtes Problem, siehe insbesondere Abu-Khzam et al. [1], Chen et al. [7] sowie Guo und Niedermeier [16].

Die hier gefundenen Problemkerne der Größe $2k$ und $3k$ Knoten werden oft als lineare Problemkerne bezeichnet. Tatsächlich ist das nicht gänzlich korrekt, da auch die Anzahl der Kanten mit in die Größe einzugehen hat. In einem allgemeinen Graphen ist die Anzahl der Kanten jedoch quadratisch in der Anzahl der Knoten.

Bemerkenswert ist, dass bei Vertex Cover mit den Problemkernen der Größe $2k$ ein Fixed-Parameter-Tractability-Ergebnis gezeigt werden kann, welches vermutlich minimal ist. Ein Problemkern mit $(2-\varepsilon)\cdot k$ Knoten, $\varepsilon > 0$, wäre eine Möglichkeit einen Faktor-$(2-\varepsilon)$ Approximationsalgorithmus für Vertex Cover zu erstellen. Das wird unter anderem von Niedermeier [24] und Bodlaender [4] als unwahrscheinlich dargestellt, weil ein solcher Approximationsalgorithmus ein ganz besonderer Durchbruch in der Approximationstheorie wäre. Khot und Regev [19] haben 2008 zu dieser Fragestellung publiziert und kommen ebenfalls zu der Erkenntnis, dass eine solche Approximation unwahrscheinlich ist.

Um die hier betrachteten Problemkern-Reduktionen miteinander zu vergleichen ziehen wir die Ergebnisse von Abu-Khzam et al. [1] zu Rate. Abu-Khzam et al. haben die oben genannten Problemkern-Reduktionen implementiert und die Laufzeit auf realen Daten zu Problemstellungen aus der Bioinformatik angewandt. Die hierbei betrachteten Graphen haben 726, 839 und 1683 Knoten, wobei der Parameter k den Wert 435, 246 beziehungsweise 1608 annimmt.

An diesen praxisrelevanten Beispielen wurden die folgenden Erkenntnisse gewonnen: Eine Problemkern-Reduktion nach Buss sollte auf alle Fälle zu Beginn durch-

geführt werden, da dieser Algorithmus sowohl sehr schnell im Vergleich zu den Nemhauser-Trotter-Ansätzen arbeitet. Ähnlich schnell arbeitet die Problemkern-Reduktion durch Kronen und liefert in manchen Fällen ähnlich gute Ergebnisse wie die lineare Programmierung oder der Ansatz über das maximale Matching. Abu-Khzam et al. raten sogar davon ab, die Nemhauser-Trotter-Ansätze zu verwenden, falls der Graph nach Anwendung der Problemkern-Reduktion nach Buss und durch Kronen sehr dicht ist. Im sehr dichten Graphen mit 1683 Knoten, $k = 1608$, haben sowohl lineare Programmierung und der Ansatz über maximale Matchings nach rund 40 Minuten Rechenzeit keine Reduktion durchführen können, während die Problemkern-Reduktion nach Buss in nur 7 Sekunden, also über 300 mal schneller, den Graphen bearbeitet hat und hierbei die Größe des Graphen fast halbieren konnte. Es zeigt sich also, dass für die algorithmische Praxis auch schwächere Problemkern-Reduktionen relevant sind.

Mit den hier gezeigten Techniken ist uns ein Einblick in die Vielfältigkeit der Ansätze der Problemkern-Reduktion gelungen. Zu vielen weiteren Graphenproblemen wurden Problemkerne bezüglich verschiedener Parametrisierungen gezeigt. Downey und Fellows haben 1995 eine umfangreiche Auflistung entsprechender Ergebnisse publiziert [12]. Bodlaender [4] verweist 2009 ebenfalls auf eine große Zahl von Ergebnissen im Bereich der Problemkern-Reduktion. Bodlaender et al. [5] zeigen im selben Jahr zudem Techniken, wie die Problemkern-Reduktion von Problemen auf planaren Graphen vereinheitlicht und erweitert werden kann.

4 Ergebnisse zu Minimum Outdegree Orientation

In diesem Kapitel befassen wir uns mit dem Graphenproblem Minimum Outdegree Orientation, zu dem ich Ergebnisse erzielen konnte, die wir in den folgenden Abschnitten betrachten werden. Wir werden einen neuen, verbesserten $\mathcal{NP}$-Vollständigkeitsbeweis für Minimum Outdegree Orientation untersuchen. Darüber hinaus gelingt es in diesem Kapitel, insbesondere für zwei verschiedene Parametrisierungen zu beweisen, dass Minimum Outdegree Orientation bezüglich des jeweiligen Parameters in der Komplexitätsklasse $\mathcal{FPT}$ liegt. Zunächst folgt die Problembeschreibung:

4.1 Beschreibung des Problems

Minimum Outdegree Orientation, oder kurz MOO, ist ein Problem auf gewichteten Graphen. Ein *gewichteter Graph* $G = (V, E, w)$ ist ein Graph bestehend aus einer Knotenmenge V und einer Kantenmenge E und einer *Gewichtsfunktion* $w : E \to \mathbb{N}$, die jeder Kante eine natürliche Zahl als *Kantengewicht* zuordnet. In Abbildung 4.1(a) ist ein gewichteter Graph zu sehen.

Eine *Orientierung* Λ eines Graphen G ist die Festlegung der Kantenrichtungen aus G. Für eine Kante $\{u, v\} \in E$ gilt entweder $(u, v) \in \Lambda$ oder $(v, u) \in \Lambda$, nicht aber beides zugleich. Eine Orientierung erzeugt damit aus einem ungerichteten Graphen einen gerichteten Graphen. Wenn ein Graph $G = (V, E, w)$ und eine Orientierung Λ vorliegt, unterscheidet man für Knoten $v \in V$ den *gewichteten Ein-*

gangsgrad δ^-_Λ und den *gewichteten Ausgangsgrad* δ^+_Λ. Der gewichtete Eingangsgrad ist die Summe der Gewichte inzidenter Kanten, die von einem beliebigen Knoten hin zum Knoten v zeigen, d.h. $\delta^-_\Lambda(v) = \sum_{u:(u,v)\in\Lambda} w(\{u,v\})$. Analog ist der gewichtete Ausgangsgrad die Summe der Gewichte der Kanten, die vom Knoten v wegführen, d.h. $\delta^+_\Lambda(v) = \sum_{u:(v,u)\in\Lambda} w(\{u,v\})$. Den maximalen gewichteten Ausgangsgrad in einem Graphen G bezeichnen wir mit $\Delta_\Lambda(G) = \max_{v\in V}\{\delta^+_\Lambda(v)\}$. Ein Beispiel für eine Orientierung, gewichtete Ausgangsgrade und den maximalen gewichteten Ausgangsgrad ist in Abbildung 4.1(b) zu finden.

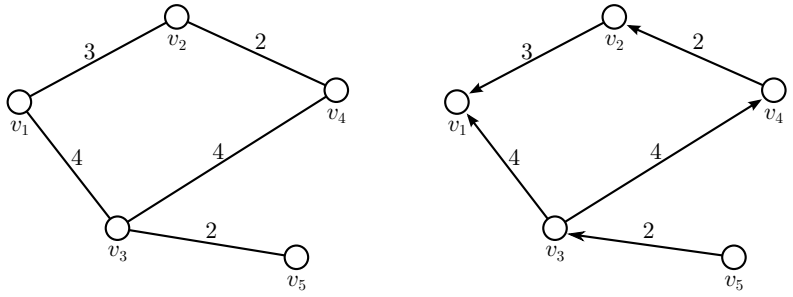


Abbildung 4.1: Gewichtete Graphen, einmal ungerichtet und einmal gerichtet.
4.1(a) zeigt einen Graphen G mit den Knoten $v_1, \ldots, v_5$ und Kantengewichten $w(\{v_1,v_2\}) = 3$, $w(\{v_1,v_3\}) = 4$, $w(\{v_2,v_4\}) = 2$, $w(\{v_3,v_4\}) = 4$, $w(\{v_3,v_5\}) = 2$.
In 4.1(b) ist derselbe Graphen G noch einmal abgebildet, diesmal allerdings mit der Orientierung $\Lambda = \{(v_2,v_1),(v_3,v_1),(v_4,v_2),(v_3,v_4),(v_5,v_3)\}$ und es gilt $\delta^+_\Lambda(v_1) = 0$, $\delta^+_\Lambda(v_2) = 3$, $\delta^+_\Lambda(v_3) = 8$, $\delta^+_\Lambda(v_4) = 2$, $\delta^+_\Lambda(v_5) = 2$ und somit $\Delta_\Lambda(G) = \delta^+_\Lambda(v_3) = 8$.

Auf Basis dieser eben erläuterten Begriffe betrachten wir die Definition des Optimierungsproblems MOO:

> Minimum Outdegree Orientation:
>
> Gegeben: Ein einfacher und gewichteter Graph $G = (V, E, w)$.
>
> Gesucht: Eine Orientierung Λ, sodass $\Delta_\Lambda(G)$ minimal ist.

Das Problem MOO besteht folglich darin, den *minimalen maximalen gewichteten Ausgangsgrad*

$$\min_{\Lambda} \Delta_{\Lambda}(G) = \min_{\Lambda} \max_{v \in V} \delta^{+}_{\Lambda}(v) = \min_{\Lambda} \max_{v \in V} \sum_{u:(v,u) \in \Lambda} w(\{u, v\})$$

zu ermitteln.

Definiert wurde das Problem zum ersten Mal 2006 durch Asahiro et al. [2]. Motiviert ist MOO durch O'Roukes *Art Gallery Problems* [25]. Das dort beschriebene *Guard Arrangement Problem* beschreibt die Aufgabe, in einer Kunstgalerie die Anzahl der Wachen zu minimieren. Die Wachen werden an Flur-Kreuzungen in der Kunstgalerie platziert, um angrenzende Flure zu überwachen. Jeder Flur muss überwacht werden. Hierbei handelt es sich um Vertex Cover. Die Variante *Capacitated Guard Arrangement* erlaubt Wachen an jeder Kreuzung; diese können jedoch nur eine begrenze Anzahl von Fluren überwachen, die selbst unterschiedlich schwer zu überwachen sind. Es handelt sich hierbei um Minimum Outdegree Orientation.

Als Hinweis sei noch erwähnt, dass es unerheblich ist, ob man den gewichteten Ausgangsgrad oder alternativ hierzu den gewichteten Eingangsgrad betrachtet. Durch Umdrehen jeder Kante in der Lösungsorientierung erhält man aus der Lösung des Einen eine Lösung für das Andere.

4.2 Verbesserter Nachweis der $\mathcal{NP}$-Vollständigkeit von MOO

Asahiro et al. [2] führen den Nachweis der $\mathcal{NP}$-Vollständigkeit von MOO über eine Reduktion von Partition, welches zu den 21 $\mathcal{NP}$-vollständigen Problemen von Karp [18] gehört. Wir sehen an dieser Stelle jedoch eine verbesserte Reduktion [3], ausgehend von 3-SAT, welches ebenfalls zu Karps 21 $\mathcal{NP}$-vollständigen Probleme zählt. Die Vorteile dieser neuen, von mir gezeigten Reduktion werden wir nach ihrer Darstellung untersuchen. Um die Reduktion durchzuführen,

betrachten wir in diesem Abschnitt die Definition des zu MOO gehörigen Entscheidungsproblems MOO(g):

> Minimum Outdegree Orientation(g):
> Gegeben: Ein einfacher und gewichteter Graph $G = (V, E, w)$ und ein Maximalgewicht g aus $\mathbb{N}$.
> Gefragt: Gibt es eine Orientierung Λ, sodass $\Delta_\Lambda(G) \leq g$ gilt?

Es gilt:

Satz 4.1. MOO(g) *ist $\mathcal{NP}$-vollständig.*

Beweis zu Satz 4.1. Offenbar ist MOO(g) $\in \mathcal{NP}$, da nichtdeterministisch jede mögliche Orientierung Λ erstellt werden kann und das Testen auf $\Delta_\Lambda(G) \leq g$ in polynomieller Zeit realisierbar ist. Es verbleibt für die $\mathcal{NP}$-Vollständigkeit nur noch die $\mathcal{NP}$-Härte zu zeigen. Wie üblich wird der Beweis der $\mathcal{NP}$-Härte durch Reduktion von einem bekannten $\mathcal{NP}$-vollständigen Problem durchgeführt. In diesem Fall greifen wir, wie bereits erwähnt, auf 3-SAT zurück.

3-SAT ist eine spezielle Variante des Erfüllbarkeitsproblems der Aussagenlogik, kurz SAT. Eine 3-SAT-Instanz wird *Formel* genannt und setzt sich aus den nachfolgend erläuterten Bestandteilen zusammen. Ein *Literal* ist die positive oder negative Version einer *boolschen Variable*, die entweder den Wert *wahr* oder *falsch* annehmen kann. Eine *Klausel* in 3-SAT besteht aus einer Oder-Verknüpfung von höchstens drei Literalen. Eine Formel besteht aus Und-verknüpften Klauseln. Diese Struktur von Und-verknüpften Klauseln mit Oder-verknüpften Literalen nennt man *konjunktive Normalform*. Die Fragestellung ist nun, ob eine Belegung der zu Grunde liegenden boolschen Variablen existiert, sodass die betrachtete Formel mit dieser Belegung zu *wahr* evaluiert wird. In diesem Fall heißt die Formel *erfüllbar*. Eine genaue Definition von SAT und 3-SAT findet sich unter anderem in Schönings Buch „Algorithmik“ [27]. Wir definieren das Entscheidungsproblem 3-SAT nun wie folgt:

3-SAT:
Gegeben: Eine aussagenlogische Formel F in konjunktiver Normalform mit höchstens 3 Literalen pro Klausel.
Gefragt: Ist F erfüllbar?

Ein Beispiel für eine 3-SAT-Instanz ist folgende Formel:

$$F = (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z)$$

Wir führen nun eine Reduktion von 3-SAT auf MOO(g) durch, indem wir in polynomieller Zeit aus einer 3-SAT-Instanz einen Graphen konstruieren werden, der genau dann einen minimalen maximalen gewichteten Ausgangsgrad $\Delta_{OPT}(G) = 2$ hat, wenn die Instanz erfüllbar ist, und $\Delta_{OPT}(G) = 3$ sonst, wobei OPT eine optimale Orientierung ist.

Vorbereitender Schritt: Zunächst sorgen wir dafür, dass die betrachtete Formel aus ausschließlich Klauseln mit genau drei unterschiedlichen Literalen besteht. Klauseln, in denen eine Variable sowohl in positiver als auch in negativer Literalausprägung vorkommt, können gestrichen werden, da die Oder-Verknüpfungen stets zu wahr evaluieren. Kommt in einer Klausel ein Literal mehrfach vor, so können die Duplikate gestrichen werden.

Verbleibt dafür zu sorgen, dass es genau drei Literale pro Klausel gibt: Hierzu werden Klauseln mit nur einem oder zwei Literalen derart erweitert, dass sie durch Klauseln mit drei Literalen nachgebildet werden. Seien hierzu x_1 und x_2 Literale der Formel und y sowie z boolsche Variablen, die noch nicht in der Formel vorkommen. Sei zudem $\neg y$ bzw. $\neg z$ die entsprechend negative boolsche Variable. So wird die Erweiterung wie folgt durchgeführt:

$$(x_1 \vee x_2) \equiv (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$$
$$(x_1) \equiv (x_1 \vee y \vee z) \wedge (x_1 \vee y \vee \neg z) \wedge (x_1 \vee \neg y \vee z) \wedge (x_1 \vee \neg y \vee \neg z)$$

Dieser vorbereitende Schritt kann in linearer Zeit durchgeführt werden und vergrößert die Formel höchstens um den Faktor 4. Es gilt im Folgenden also ohne Beschränkung der Allgemeinheit, dass alle Klauseln der betrachteten Formel genau drei Literale mit jeweils drei unterschiedlichen boolschen Variablen haben.

Konstruktion des Graphen: Ziel der Konstruktion ist, dass am Ende für jede boolsche Variable $x_1, x_2, \ldots, x_n$ der Formel genau eine spezielle Kante im Graph vorhanden ist deren Orientierung die Belegung der Variable angibt. Zudem wird jeder Klausel k der Formel durch die Konstruktion genau ein Knoten v_k zugeordnet.

Zunächst erzeugen wir für jede boolsche Variable $x_1, x_2, \ldots, x_n$ der Formel einen eigenen Teilgraphen. Dieser Teilgraph hat folgende grundlegende Struktur, wie sie auch in der Abbildung 4.2 zu sehen ist: Man beginnt mit einem Teilgraphen, der in der Mitte eine Kante des Gewichts 2 hat und hierzu links und rechts jeweils zwei adjazente Kanten mit Gewicht 1. Die Orientierung der mittleren Kante spezifiziert, ob die boolsche Variable mit *wahr* (Orientierung von links nach recht) oder mit *falsch* (Orientierung von rechts nach links) zu belegen ist. An beiden Seiten wird der Teilgraph baumartig erweitert, bis hinreichend viele Endknoten, welche wir im Folgenden als *Konnektoren* bezeichnen, verfügbar sind, um die nachfolgende Konstruktion durchführen zu können. Die Erweiterung wird durchgeführt, indem an einen Konnektor eine Kante mit Gewicht 2 hinzugefügt wird, welche wiederum zwei neue adjazente Kanten mit Gewicht 1 mitbringt. Hierbei ist der ursprüngliche Konnektor kein Endknoten mehr und entfällt somit. Jedoch sind die Endknoten der beiden neuen Kanten mit Gewicht 1 neue Konnektoren.

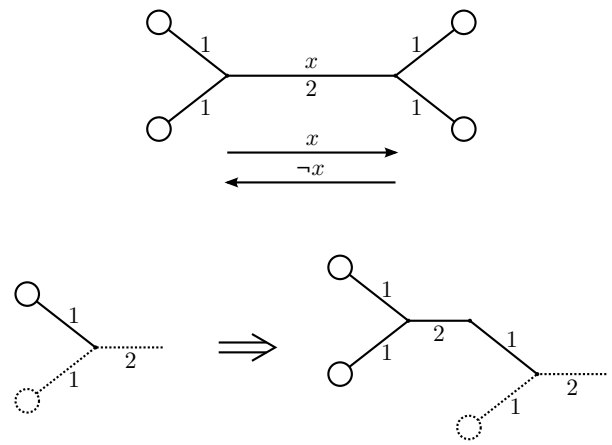


Abbildung 4.2: Teilgraph, wie er für jede boolsche Variable konstruiert wird. Begonnen wird bei der Konstruktion mit dem oben abgebildeten Graphen, welcher dann, wie darunter dargestellt, ggf. links oder rechts erweitert wird. Die Kante in der Mitte spezifiziert am Ende, ob die boolsche Variable mit *wahr* (Orientierung von links nach rechts) oder mit *falsch* (Orientierung von rechts nach links) belegt ist. Die Erweiterung wird so lange betrieben, bis genügend Konnektoren am Teilgraph vorhanden sind. Links werden so viele Konnektoren benötigt, wie es Klauseln mit der boolschen Variable in negativer Literalausprägung gibt, rechts hingegen so viele, wie es Klauseln mit der boolschen Variable in positiver Literalausprägung in der Formel gibt.

So kann sukzessive die Konnektorenanzahl auf beiden Seiten erhöht werden, wie in der Abbildung 4.2 gezeigt ist. Hierbei müssen die beiden baumartigen Seiten weder gleich groß noch balanciert sein, wie das Beispiel in Abbildung 4.3 zeigt. Links werden so viele Konnektoren benötigt, wie es Klauseln mit der boolschen Variable in negativer Literalausprägung gibt, rechts hingegen so viele, wie es Klauseln mit der boolschen Variable in positiver Literalausprägung in der Formel hat. Sollten einmal weniger als zwei Konnektoren auf einer Seite benötigt werden, so bleiben die überzähligen Konnektoren unbenutzt.

Für jede Klausel k der Formel wird nun je ein bisher unbenutzter Konnektor aus dem jeweiligen Teilgraphen der drei beteiligten boolschen Variablen ausgewählt. Wird die boolsche Variable in negativer Literalausprägung in der aktuellen

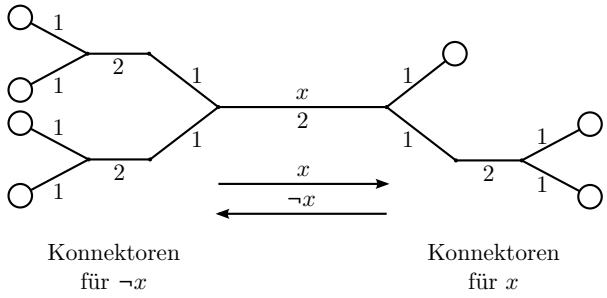


Abbildung 4.3: Beispiel eines Teilgraphen für eine boolsche Variable x.
Die boolsche Variable x kommt in diesem Beispiel in einer Formel viermal in negativer Literalausprägung und dreimal in positiver Literalausprägung vor. Entsprechend viele Konnektoren stehen zur Verfügung.

Klausel k verwendet, so wird ein ungenutzter Konnektor auf der linken Seite ihres Teilgraphen verwendet, ansonsten ein ungenutzter Konnektor auf der rechten Seite. Zu den drei so gefundenen Konnektoren pro Klausel werden die entsprechenden Knoten verschmolzen. Dieser Knoten, der aus der Knotenverschmelzung der drei Konnektoren entsteht, ist der Knoten v_k, den es für jede Klausel durch die Konstruktion zu erstellen gilt. In Abbildung 4.4 ist eine solche Auswahl anhand der drei Klauseln aus der obigen Beispielformel F zu sehen.

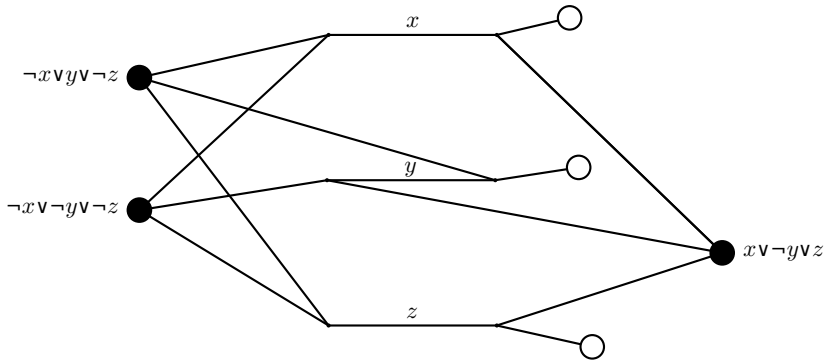


Abbildung 4.4: Beispielhafte Auswahl von Konnektoren anhand der Klauseln aus Formel F.
Zu beachten ist insbesondere, dass die ausgewählten Konnektoren pro Klausel knotenverschmolzen werden. Der Übersichtlichkeit wegen wurden die Kantengewichte nicht eingezeichnet.

Die Konstruktion des Graphen lässt sich in polynomieller Zeit durchführen, indem für jede boolsche Variable zuerst ihr positives und negatives Auftreten gezählt wird. Dann werden die Teilgraphen erstellt und zum Schluss jede Klausel direkt umgesetzt. Es werden dreimal so viele Konnektoren wie Klauseln benötigt, die Teilgraphengröße ist linear abhängig von der Anzahl der zur Verfügung zu stellenden Konnektoren. Somit ist die Größe des gesamten Graphen linear in der Größe der Formel.

Korrektheit: Sei F die 3-SAT-Formel, deren Erfüllbarkeit zu prüfen ist und sei G der mit der obigen Methode konstruierte Graph zur Formel F. Es ist zu zeigen, dass $\Delta_{OPT}(G) = 2$ genau dann gilt, wenn F erfüllbar ist und $\Delta_{OPT}(G) = 3$ sonst, wenn OPT eine optimale Orientierung ist. Dass dies der Fall ist, lässt sich erkennen, wenn wir ausgehend von einer Belegung der beteiligten boolschen Variablen versuchen, eine Orientierung Λ mit $\Delta_\Lambda(G) = 2$ zu konstruieren. Wird die boolsche Variable x auf *wahr* gesetzt, so haben alle Konnektoren der linken Seite, welche für Klauseln mit $\neg x$ stehen, zwingend eine ausgehende Kante mit Gewicht 1 in den Teilgraphen von x, da andernfalls im Inneren des Teilgraphen ein Knoten einen höheren gewichteten Ausgangsgrad als 2 haben müsste: Der Knoten, aus welchem die mittlere Kante ausgeht, hat durch diese Kante bereits den gewichteten Ausgangsgrad 2. Die beiden inzidenten Kanten mit Gewicht 1 müssen daher in diesen Knoten hinein führen, damit der gewichtete Ausgangsgrad 2 nicht übersteigt. Die Knoten aus denen diese Kanten mit Gewicht 1 ausgehen sind gegebenenfalls wieder inzident zu einer weiteren Kante mit Gewicht 2, welche nicht aus dem Knoten ausgehen können, da sonst auch hier der gewichtete Ausgangsgrad größer als 2 werden würde. Dies ist in Abbildung 4.5 zu sehen.

Auch lässt sich erkennen, dass innerhalb eines Teilgraphen TG, die Konnektoren ausgenommen, der minimale maximale gewichtete Ausgangsgrad $\Delta_{OPT}(TG) = 2$ ist: Kleiner als 2 kann $\Delta_{OPT}(TG)$ nicht sein, da bereits die mittlere Kante das Kantengewicht 2 hat. Gleichzeitig gibt es immer eine solche Belegung, indem

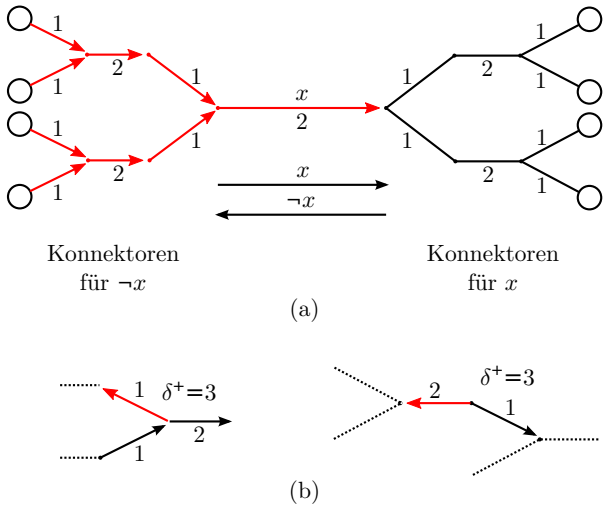


Abbildung 4.5: Konsequenz aus der Orientierung der mittleren Kante von links nach rechts.
Oben ist zu sehen, wie die Orientierung der mittleren Kante den weiteren Kanten ihre Orientierung aufzwingt. Die Orientierung der mittleren Kante in diesem Beispiel steht dafür, dass die boolesche Variable x *wahr* ist. Diese Orientierung sorgt dafür, dass alle Konnektoren auf der linken Seite, welche für Klauseln mit $\neg x$ stehen, zwingend eine ausgehende Kante mit Gewicht 1 in den Teilgraphen TG von x haben, sofern man $\Delta_\Lambda(TG) = 2$ erreichen möchte.
Unten ist zu sehen, wie ein Abweichen von der Orientierung automatisch zu inneren Knoten mit gewichtetem Ausgangsgrad von 3 führt.

alle Kanten entsprechend der mittleren Kante nach links bzw. rechts ausgerichtet werden. Somit gilt also $\Delta_{OPT}(TG) = 2$ für den Teilgraph TG ohne dessen Konnektoren.

Wird nun eine Klausel der Formel nicht erfüllt, so bedeutet das, dass alle drei in ihr vorkommenden boolschen Variablen genau entgegengesetzt zu der Verwendung in der Klausel belegt sind. Das bedeutet aber auch, dass die drei knotenverschmolzenen Konnektoren, die für diese Klausel stehen, für alle drei Variablen eine ausgehende Kante mit Gewicht 1 haben. Somit hat der Knoten, der aus der Verschmelzung der drei Konnektoren besteht, einen gewichteten Ausgangsgrad

von 3 oder innerhalb eines beteiligen Teilgraphen befindet sich ein Knoten mit gewichtetem Ausgangsgrad 3. Ein Beispiel hierfür ist in Abbildung 4.6 zu sehen.

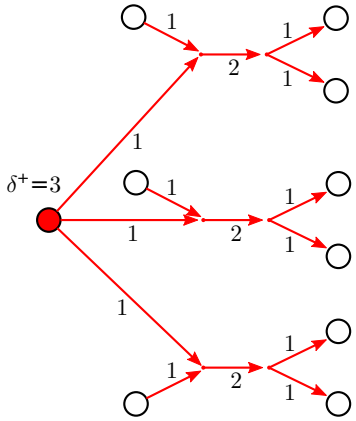


Abbildung 4.6: Durch unerfüllte Klauseln entstehen Knoten mit gewichtetem Ausgangsgrad von 3.
Der rot markierte Knoten ist die Verschmelzung dreier Konnektoren, die für die Klausel $(\neg x \vee \neg y \vee \neg z)$ steht. Wählt man nun x, y, z als *wahr*, so ist die Klausel offenbar nicht erfüllt. Gleichzeitig werden dadurch die drei mittleren Kanten per Definition nach rechts orientiert. Die Konsequenz ist, dass entweder der schwarz markierte Knoten einen gewichteten Ausgangsgrad von 3 haben muss, oder alternativ, falls eine der von ihm wegführenden Kanten umgedreht wird, in einem der Teilgraphen ein Knoten mit gewichtetem Ausgangsgrad von 3 vorliegt.

Die Unerfüllbarkeit einer 3-SAT-Formel bedeutet, dass unter jeder Belegung der boolschen Variablen mindestens eine Klausel der Formel zu *falsch* evaluiert wird. Und eine unerfüllte Klausel führt zu einem Knoten mit gewichtetem Ausgangsgrad von 3. Da innerhalb der Teilgraphen TG der minimale maximale gewichtete Ausgangsgrad $\Delta_{OPT}(TG) = 2$ ist und die verschmolzenen Konnektoren jeweils nur drei inzidente Kanten mit Gewicht 3 haben, gilt somit gleichzeitig $\Delta_{OPT}(G) \leq 3$. Zusammengesetzt ergibt dies: Ist die dem Graphen G zu Grunde liegende Formel F unerfüllbar, so gilt $\Delta_{OPT}(G) = 3$. OPT sei hierbei eine jeweils optimale Lösung.

Es verbleibt die Rückrichtung zu zeigen: $\Delta_{OPT}(G) = 3$, OPT optimal, führt dazu, dass F unerfüllbar ist. Hierzu äquivalent: F erfüllbar führt zu $\Delta_{OPT}(G) \neq 3$, OPT optimal. Und $\Delta_{OPT}(G) \neq 3$ bedeutet nach der vorhergehenden Überlegung $\Delta_{OPT}(G) = 2$.

Sei nun also F erfüllbar. So existiert eine Belegung der boolschen Variablen, sodass jede Klausel zu *wahr* evaluiert. Wir wählen eine solche Belegung und orientieren die korrespondierenden mittleren Kanten der jeweiligen Teilgraphen entsprechend. Weiter richten wir alle Kanten der jeweiligen Teilgraphen in dieselbe Richtung aus, wie ihre jeweilige mittlere Kante, siehe hierzu Abbildung 4.7. Wir nennen diese Orientierung Λ. Innerhalb eines jeden Teilgraphen TG gilt mit dieser so gewählten Orientierung, dass $\Delta_\Lambda(TG) = 2$ ist. Aber auch für jede drei miteinander verschmolzenen Konnektoren ist der gewichtete Ausgangsgrad höchstens 2: Nach Konstruktion befinden sich die für eine Klausel verwendeten Konnektoren zumindest bezüglich einer der verwendeten boolschen Variable auf derjenigen Seite des Teilgraphen, sodass die Verbindungskante vom entsprechenden Teilgraphen zu den verschmolzenen Konnektoren hin verläuft. Somit können jeweils höchstens noch zwei Kanten aus den verschmolzenen Konnektoren ausgehen und der gewichtete Ausgangsgrad ist höchstens 2. Folglich gilt $\Delta_\Lambda(G) = 2$.

Damit ist auch die Rückrichtung gezeigt und somit der Beweis erbracht.

□

Durch diesen Beweis ist nicht nur gezeigt, dass MOO(g) $\mathcal{NP}$-vollständig ist, sondern auch, dass MOO(g) bereits dann $\mathcal{NP}$-vollständig ist, wenn man sich auf Graphen mit Knotengrad ≤ 3 und zwei unterschiedlichen Kantengewichte beschränkt. Dies ist deshalb besonders, weil beide Bedingungen gleichzeitig untere Schranken für die $\mathcal{NP}$-Vollständigkeit von MOO(g) darstellen. Im folgenden Abschnitt werden wir zum einen sehen, dass Asahiro et al. [2] zeigen, dass MOO(g) für Graphen mit nur einem identischen Kantengewicht für alle Kanten in polyno-

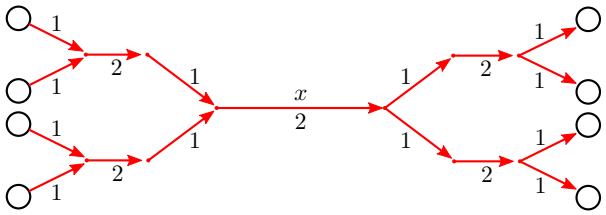


Abbildung 4.7: Gleich-Orientierung aller Kanten anhand der mittleren Kante. Diese Orientierung sorgt dafür, dass für den Teilgraphen gilt, dass der maximale gewichtete Ausgangsgrad 2 ist und zudem alle durch diese Variablenbelegung (in diesem Fall ist x *wahr* da die mittlere Kante von links nach rechts ausgerichtet ist) *wahr* werdenden Klausel knotenverschmolzenen Konnektoren entsprechen, die ebenfalls nur einen maximalen gewichteten Ausgangsgrad von 2 haben.

mieller Zeit lösbar ist. Zum anderen wird im darauffolgenden Abschnitt zu sehen sein, dass MOO(g) auf Graphen mit Knotengrad ≤ 2 polynomiell lösbar ist.

Weiter können wir aus dem obigen Beweis auch folgern, dass im Falle $\mathcal{P} \neq \mathcal{NP}$ kein approximativer polynomieller Algorithmus existieren kann, der MOO besser als Faktor 3/2 approximiert. Dies liegt daran, dass der minimale maximale Ausgangsgrad $\Delta_{OPT}(G)$ im obigen Algorithmus nur 2 oder 3 sein kann und damit das $\mathcal{NP}$-vollständige Problem 3-SAT entscheidet, wenn OPT eine optimale Orientierung ist. Gäbe es nun einen Algorithmus der polynomiell eine Approximation besser als 3/2 von MOO erzeugen könnte, so wäre das auch für den Graphen aus dem Beweis möglich. Dies würde aber sofort eine polynomielle Lösung der zu Grunde liegenden 3-SAT Instanz bedeuten, da die Approximation eben entweder ein Ergebnis kleiner als 3 hat und somit $\Delta_{OPT}(G)$ nur 2 sein kann ($\Rightarrow$ die 3-SAT-Instanz ist erfüllbar) oder aber das approximierte Ergebnis ≥ 3 ist und somit, unter der Annahme, dass die Approximation besser als Faktor 3/2 ist, auch bedeutet, dass $\Delta_{OPT}(G) = 3$ ist ($\Rightarrow$ die 3-SAT-Instanz ist unerfüllbar). Dies ist ein Widerspruch zur Annahme $\mathcal{P} \neq \mathcal{NP}$. In approximativen Komplexitätsklassen gesprochen bedeutet das: MOO $\notin \mathcal{PTAS}$. Da aber tatsächlich po-

lynomielle approximative Algorithmen für MOO existieren ([2]) gilt gleichzeitig MOO $\in \mathcal{APX}$. Beide Klassen werden beispielsweise in Schöning [27] definiert.

4.3 Ergebnis: MOO(k) liegt in $\mathcal{FPT}$

Wir untersuchen die folgende parametrisierte Variante von MOO:

> Minimum Outdegree Orientation mit k andersgewichteten Kanten (kurz MOO(k)):
> Gegeben: Ein einfacher und gewichteter Graph $G = (V, E_c \cup E_k, w)$. Hierbei haben alle Kanten aus E_c das Gewicht c, wobei c ein beliebiges aber festes Gewicht aus $\mathbb{N}$ ist. Die Kanten aus E_k mit $|E_k| = k$ haben jeweils ein beliebiges Gewicht aus $\mathbb{N}$.
> Gesucht: Eine Orientierung Λ, sodass $\Delta_\Lambda(G)$ minimal ist.

In der Abbildung 4.8 ist eine Beispielinstanz von MOO(k) visualisiert.

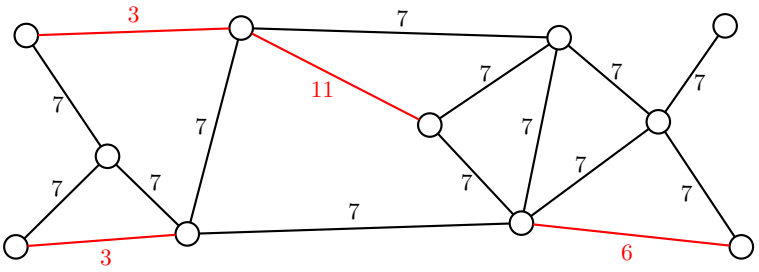


Abbildung 4.8: Beispiel einer Eingabeinstanz von MOO(k).
Der hier abgebildete Graph hat $k = 4$ andersgewichtete Kanten (in Rot eingezeichnet), die restlichen Kanten haben alle das Gewicht $c = 7$.

Offenbar gilt, dass auch MOO(k) $\mathcal{NP}$-vollständig ist, da MOO mit $k = |E| - 1$ ein Spezialfall von MOO(k) ist.

Für einen beliebigen einfachen und gewichteten Graphen mit der Kantenmenge E lässt sich eine Zerlegung von E in E_c und E_k in linearer Zeit durchführen,

sodass $|E_c|$ maximal ist. Hierzu betrachtet man jedes Kantengewicht und legt als Wert von c das am häufigsten auftretende Kantengewicht der Kanten aus E fest. Ist c bekannt, können die Kanten aus E direkt den Kantenmengen E_c und E_k zugeordnet werden

Es gelingt zu MOO(k) folgendes zu zeigen:

Satz 4.2. MOO(k) $\in \mathcal{FPT}$

Der Beweis orientiert sich an Asahiro et al. [2] die gezeigt haben, dass MOO auf einem Graphen, in welchem alle Kanten dasselbe Gewicht haben (also MOO(0) betrachtet wird), in polynomieller Zeit lösbar ist. Hierzu benötigen wir noch die Definition eines *gerichteten Pfades*: Ein gerichteter Pfad P der Länge l von einem Knoten v_0 zu einem Knoten v_k in einem gerichteten Graphen mit Orientierung Λ ist eine Menge von gerichteten Kanten $\{(v_{i-1}, v_i) | (v_{i-1}, v_i) \in \Lambda, 1 \leq i \leq k\}$, welche wir mit $P = \langle v_0, v_1, \ldots, v_k \rangle$ bezeichnen. $\bar{P} = \langle v_k, v_{k-1}, \ldots, v_0 \rangle$ bezeichnet den zu P entgegen gesetzten Pfad $\bar{P}$.

Beweis zu Satz 4.2. Der Beweis erfolgt über eine Konstruktion eines Algorithmus, der MOO(k) in $f(k) \cdot n^{O(1)}$ Zeit löst, wobei n die Größe der Eingabeinstanz ist. Wir iterieren hierzu über alle Orientierungen Λ_k der k andersgewichteten Kanten und lösen den verbleibenden Graphen mit einer modifizierten Variante des Algorithmus „Reverse“ von Asahiro et al. [2]. Die Hauptidee dieses Algorithmus „Reverse“ ist es, dass in einem Graphen mit gleichen Kantengewichten für jede Kante und Orientierung Λ das Austauschen eines gerichteten Pfades P durch den entgegengesetzten Pfad $\bar{P}$ in Λ nur den gewichteten Ausgangsgrad des Start- und Endknotens verändert, nicht jedoch den der Knoten dazwischen. Siehe hierzu Abbildung 4.9, in welcher dieses beispielhaft an einem Pfad demonstriert wird.

Der Algorithmus 1 nutzt diese Idee nun sukzessive, um eine optimale Lösung für MOO(k) zu konstruieren, immer in Abhängigkeit von der jeweils bereits gesetzten Orientierung Λ_k.

Algorithmus 1: MOO((G, k))

Eingabe : Ein Graph $G = (V, E_c \cup E_k, w)$ mit k Kanten E_k mit beliebigem Gewicht $\in \mathbb{N}$, $k \in \mathbb{N}$, sowie Kanten E_c mit $\forall e \in E_c : w(e) = c, c \in \mathbb{N}$

Ausgabe : Eine Orientierung Λ mit $\Delta_\Lambda(G)$ minimal

```
1  Λ_min := ∅
2  O := Menge aller möglichen Orientierungen der Kanten E_k
3  solange O ≠ ∅ tue
4      Wähle ein Λ_k ∈ O und setze O := O \ {Λ_k}
5      Setze Λ := Λ_k und vervollständige Λ zu einer beliebigen Orientierung
       von G
6      für alle v ∈ V tue berechne δ⁺_Λ(v)
7      wiederhole
8          Wähle einen Knoten u mit δ⁺_Λ(u) ≥ δ⁺_Λ(v) ∀v ∈ V
9          Suche einen gerichteten Pfad P = ⟨u, ..., p_l⟩, l ≥ 1 in E_c mit
           δ⁺_Λ(p_l) < δ⁺_Λ(u) − c
10         wenn ein solcher Pfad P existiert dann
11             Setze Λ := (Λ \ P) ∪ P̄
12             δ⁺_Λ(p_l) := δ⁺_Λ(p_l) + c
13             δ⁺_Λ(u) := δ⁺_Λ(u) − c
14     bis kein Pfad P gefunden wird
15     wenn Λ_min = ∅ oder Δ_{Λ_min}(G) > Δ_Λ(G) dann Λ_min := Λ
16 Gib Λ_min aus
```

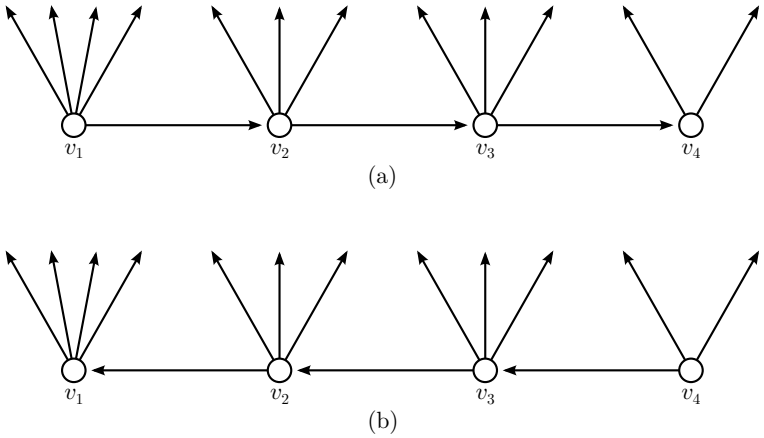


Abbildung 4.9: Beispiel eines umzudrehenden Pfades des Algorithmus „Reverse“. Seien die Kantengewichte alle 1.
In (a) gilt $\delta^+_\Lambda(v_1) = 5$, $\delta^+_\Lambda(v_2) = 4$, $\delta^+_\Lambda(v_3) = 4$ und $\delta^+_\Lambda(v_4) = 2$ und den Pfad $P = \langle v_1, v_2, v_3, v_4\rangle$.
In (b) ist dieser Pfad nun umgedreht, also $\bar{P} = \langle v_4, v_3, v_2, v_1\rangle$. Geändert hat sich jedoch nur $\delta^+_\Lambda(v_1)$ um -1 auf 4 und $\delta^+_\Lambda(v_4)$ um $+1$ auf 3. Die gewichteten Ausgangsgrade der Knoten innerhalb des Pfades haben sich nicht geändert.

Funktionsweise: Betrachten wir nun die Funktionsweise des Algorithmus. In Zeile 1 wird Λ_{min} eingeführt, welches am Ende in Zeile 16 eine optimale Orientierung enthalten wird. Wie angekündigt iteriert die äußere Schleife, siehe Zeile 3, des Algorithmus über alle möglichen Orientierungen $\mathcal{O}$ der Kanten von E_k. Diese Menge von $|\mathcal{O}| = 2^k$ Orientierungen wird in Zeile 2 eingeführt und in Zeile 4 wird in jedem Schleifendurchlauf eine Orientierung Λ_k gewählt. Λ_k wird hierbei aus $\mathcal{O}$ gestrichen, womit die äußere Schleife in Zeile 3 terminiert. In Zeile 5 wird die gewählte Orientierung Λ_k zu Λ vervollständigt, d.h. die Orientierung der Kanten E_k wird in Λ über Λ_k festgelegt während die Orientierungen der Kanten E_c frei gewählt werden können.

Die Schleife in Zeile 7 optimiert nun die gleichgewichteten Kanten. Hierzu wird in Zeile 8 der Knoten u bestimmt, der einen maximalen gewichteten Ausgangsgrad hat. Dieser Knoten muss nicht eindeutig sein. Sollte die innere Schleife bei diesem

Knoten u abbrechen, so ist die Optimierung auf jeden Fall zu Ende. Dies ist darin begründet, dass definitionsgemäß $\Delta_\Lambda(G) = \delta^+_\Lambda(u)$ gelten muss. Somit folgt aus der Nicht-Optimierbarkeit von $\delta^+_\Lambda(u)$ automatisch, dass weitere Optimierungen an anderen Knoten (möglicherweise mit selben gewichteten Ausgangsgrad) keine Verbesserung von $\Delta_\Lambda(G)$ bewirken können. Somit kann der Algorithmus in Zeile 8 einen beliebigen solchen Knoten u wählen.

Analog zum Algorithmus „Reverse“ suchen wir in der Zeilen 9 einen gerichteten Pfad P in den gleichgewichteten Kanten E_c. Beachte: Die Kanten E_k werden in diesem Schritt vollständig ignoriert, da ihre Orientierung durch Λ_k bereits fest gesetzt ist. Für $P = \langle u, \ldots, p_l \rangle$ muss nun gelten, dass die Differenz von $\delta^+_\Lambda(u)$ zu $\delta^+_\Lambda(p_l)$ größer als das Gewicht c der gleichgewichteten Kanten ist. Erfüllt der Pfad P diese Eigenschaft, wird Λ dadurch verbessert, dass alle Kanten des Pfades P in Λ umgedreht werden (Zeile 11). Dies verringert den gewichteten Ausgangsgrad von u um c, da der Pfad gerichtet und aus E_c war, und wir somit in diesem Schritt eine ausgehende Kante von u mit dem Gewicht c umgedreht haben. Analog erhöht sich der gewichtete Ausgangsgrad von p_l um c. Diese Änderungen werden in den Zeilen 12 und 13 vollzogen. Für alle anderen Knoten des Pfades ändert sich der gewichtete Ausgangsgrad nicht, da eine ausgehende Kante mit Gewicht c zu einer eingehenden wird aber gleichzeitig eine andere eingehende Kante mit dem Gewicht c zu einer ausgehenden gemacht wird. Abbildung 4.9 zeigt dies an einem Beispiel.

Die innere Schleife (Zeile 7) wird nun so lange durchgeführt, wie noch Pfade gefunden werden können, die einem Knoten mit dem jeweiligen maximalen gewichteten Ausgangsgrad einen geringeren Ausgangsgrad verschaffen. Gibt es keine solchen Pfade mehr, ist Λ bezüglich dem bereits gesetztem Λ_k optimal, wie wir gleich sehen werden. Daher wird in Zeile 15 überprüft, ob es sich bei Λ um eine bessere Orientierung als das bisherige Λ_{min} handelt, und Λ_{min} wird gegebenenfalls aktualisiert.

Korrektheit: Zunächst zeigen wir, dass der Algorithmus in der inneren Schleife, Zeile 7, tatsächlich ein optimales Λ zum gegebenen Λ_k findet. Sei hierzu Λ_k im Folgenden beliebig aber fest. Für den Nachweis der Korrektheit der inneren Schleife stellen wir nun fest, dass gilt [3]:

Lemma 4.3. *Sei Λ diejenige Orientierung für den zu untersuchenden Graphen $G = (V, E_c \cup E_k, w)$, welche das Ergebnis der inneren Schleife des Algorithmus zum gegebenen Λ_k ist. So gilt: Λ ist optimal unter dem gegebenen Λ_k.*

Beweis zu Lemma 4.3. Dass Λ das Ergebnis der inneren Schleife des Algorithmus ist, heißt mit anderen Worten, dass kein weiterer Pfad $P = \langle u, \ldots, p_l \rangle, l \geq 1$ in den Kanten E_c gefunden werden kann, für den gilt, dass $\delta^+_\Lambda(p_l) < \delta^+_\Lambda(u) - c$.

Annahme: Λ sei nicht optimal (unter nach wie vor fest gesetztem Λ_k).

Die Suboptimalität von Λ bedeutet, dass es eine Orientierung $\widehat{\Lambda}$ gibt, für die gilt: $\Delta_{\widehat{\Lambda}}(G) < \Delta_\Lambda(G)$. Die beiden Orientierungen unterscheiden sich dann in einer Menge von Kanten $\widehat{E} \subseteq E_c$, deren Umdrehen von Λ zu $\widehat{\Lambda}$ führt. Damit jedoch gelten kann, dass $\widehat{\Lambda}$ tatsächlich eine bessere Orientierung ist, müssen bei der Änderung von Λ zu $\widehat{\Lambda}$ alle Knoten $v \in V$ mit maximalem gewichteten Ausgangsgrad $\delta^+_\Lambda(v) = \Delta_\Lambda(G)$ in ihrem gewichteten Ausgangsgrad verringert werden. Halten wir einen solchen Knoten u fest. Folglich muss sich für den Knoten u mindestens eine Kante $\{u, p_1\}$ in $\widehat{E}$ befinden, welche unter Λ eine von u ausgehende Orientierung hat, da ansonsten beim Umdrehen der Kanten $\widehat{E}$ keine Verbesserung des gewichteten Ausgangsgrads von u möglich ist und somit $\Delta_{\widehat{\Lambda}}(G) \geq \Delta_\Lambda(G)$ gilt, was ein Widerspruch zur Annahme wäre.

Wir versuchen nun im Folgenden einen Widerspruch zu konstruieren, in dem wir einen Pfad $\widehat{P} = \langle u, \ldots, p_l \rangle, l \geq 1$ in den Kanten E_c finden, für den gilt, dass $\delta^+_\Lambda(p_l) < \delta^+_\Lambda(u) - c$ ist, was nach Voraussetzung nicht möglich sein sollte. Wir beginnen den Pfad $\widehat{P}$ durch die Kante (u, p_1), deren Existenz wir nach obiger Überlegung belegt haben.

Betrachten wir nun diesen adjazenten Knoten p_1:

Fall 1**:** Keine von p_1 unter Λ ausgehenden Kanten befinden sich in $\widehat{E}$. In diesem Fall gilt $\delta^+_{\widehat{\Lambda}}(p_1) \geq \delta^+_{\Lambda}(p_1) + c$, da mindestens das Gewicht c der Kante $\{u, p_1\} \in \widehat{E}$ zum gewichteten Ausgangsgrad von p_1 hinzu kommt. Nach Annahme muss jedoch gelten:

$$\Delta_{\Lambda}(G) > \Delta_{\widehat{\Lambda}}(G) \geq \delta^+_{\widehat{\Lambda}}(p_1) \geq \delta^+_{\Lambda}(p_1) + c$$

Folglich gilt mit der Voraussetzung, dass $\delta^+_{\Lambda}(u) = \Delta_{\Lambda}(G)$ ist:

$$\delta^+_{\Lambda}(p_1) < \Delta_{\Lambda}(G) - c = \delta^+_{\Lambda}(u) - c$$

Dann ist jedoch $\widehat{P} = \langle u, p_1 \rangle$ ein gültiger Pfad, den der Algorithmus hätte umdrehen müssen. Dies ist ein Widerspruch zur Voraussetzung, dass Λ das Ergebnis der inneren Schleife des Algorithmus ist, und kein solcher Pfad mehr existiert.

Fall 2**:** Es gibt inzidente Kanten zu p_1, welche unter Λ aus p_1 ausgehend orientiert sind und sich in $\widehat{E}$ befinden. Sei (p_1, p_2) eine solche Kante. Wir fügen diese Kante zu $\widehat{P}$ hinzu. Nun gilt für p_2 dieselbe Argumentation wie für p_1: Entweder es gilt $\delta^+_{\Lambda}(p_2) < \delta^+_{\Lambda}(u) - c$ und $\widehat{P} = \langle u, p_1, p_2 \rangle$ wäre ein gültiger Pfad, was einen Widerspruch darstellt, oder aber es gibt auch hier wieder einen adjazenten Knoten p_3 mit $\{p_2, p_3\} \in \widehat{E}$ und $(p_2, p_3) \in \Lambda$.

Dieses Argument lässt sich nun induktiv fortsetzen um einen Pfad $\widehat{P} = \langle u, \ldots, p_l \rangle$ mit $l \geq 1$ in den Kanten E_c zu konstruieren. Zu beachten ist, dass wir uns bei jedem Schritt auf die Verwendung einer bis zu diesem Zeitpunkt noch nicht verwendeten Kante beschränken können. Dies ist so, da ein Pfad, der einen Zyklus enthält, beim Umdrehen die gleiche Wirkung hat wie seine um den Zyklus bereinigte Version. Da der Graph jedoch eine endliche Größe hat, terminiert diese induktive Pfadkonstruktion. Und für den letzten Knoten p_l des Pfades kommt dann nur noch der obige Fall 1 in Frage. Somit ist entweder Λ doch optimal, was ein Wi-

derspruch zur Annahme wäre, oder für den Pfad $\widehat{P}$ gilt, dass $\delta^+_\Lambda(p_l) < \delta^+_\Lambda(u) - c$, was ein Widerspruch zur Voraussetzung wäre.

□

Die Korrektheit des gesamten Algorithmus folgt nun daraus, dass die äußere Schleife des Algorithmus (Zeile 3) alle möglichen Orientierungen Λ_k systematisch durchprobiert. Es befindet sich also irgendwann auch eine optimale Orientierung OPT_k der Kanten E_k unter den von der äußerem Schleife durchprobierten Orientierungen. Die innere Schleife findet nun, wie eben gezeigt, eine optimale vervollständigte Orientierung Λ für jedes beliebige Λ_k, also auch für OPT_k. Das Ergebnis Λ_{min} ist dann diese optimale vervollständigte Orientierung Λ zu OPT_k. Damit ist die Korrektheit gezeigt.

Laufzeit: Sei $E = E_k \cup E_c$. Zunächst einmal können wir davon ausgehen, dass G zusammenhängend ist und $|E| \geq |V|$. Ist G nicht zusammenhängend, wenden wir den Algorithmus auf die einzelnen Zusammenhangskomponenten an und konstruieren daraus die Gesamtlösung. Wenn G nun also zusammenhängend ist, so würde $|E| < |V|$ bedeuten, dass G ein Baum ist. Hier wäre die triviale Linearzeit-Lösung, dass wir dem Baum eine beliebige Wurzel geben und jede Kante stets vom Kindknoten zum Vaterknoten verläuft. Folglich können wir diese Annahmen ohne Beschränkung der Allgemeinheit voraussetzen.

Bei der Laufzeitbetrachtung des Algorithmus fällt weiter auf, dass $\mathcal{O}$ nicht explizit aufgestellt werden muss. Statt dessen ist es ausreichend systematisch alle $|\mathcal{O}| = 2^k$ Orientierungen zu testen, etwa über einen Suchbaum oder indem man für jede der k Kanten eine Richtung mit 0 identifiziert und die andere mit 1 und binär durchzählt. Die Laufzeit der äußeren Schleife (Zeile 3) ist wegen $|\mathcal{O}| = 2^k$ gerade $O(2^k) =: f(k)$. Betrachten wir nun also im Folgenden die Laufzeit innerhalb dieser Schleife.

Zeile 5 benötigt $O(|E_c|)$ Zeit und Zeile 6 benötigt in $O(|E|)$ Zeit, da jede Kante den gewichteten Ausgangsgrad von einem der Knoten erhöht. Bevor wir uns nun der inneren Schleife widmen, stellen wir noch fest, dass Zeile 15 in $O(|V|)$ Zeit berechnet werden kann und die Ausgabe (Zeile 16) in $O(|E|)$ Zeit passiert. Ein Durchlauf der äußeren Schleife benötigt also $O(|E_k \cup E_c| + |V|) = O(|E|)$ Zeit, zuzüglich dem Zeitbedarf der durch die innere Schleife entsteht.

Es verbleibt die innere Schleife des Algorithmus aus Zeile 7. Die einzelnen Schritte in dieser Schleife sind ebenfalls in linearer Zeit machbar: Einen geeigneten Knoten u finden wir in $O(|V|)$ (Zeile 8) und über eine Tiefensuche benötigt das Finden von einem geeigneten Pfad P $O(|E_c|)$ Zeit (Zeile 9). Die gegebenenfalls nötigen Aktualisierungen (Zeilen 11 bis 13) finden in $O(|E_c|)$ beziehungsweise $O(1)$ statt. Die innere Schleife benötigt also $O(|E_c| + |V|) = O(|E_c|) = O(|E|)$ Zeit.

Um nun die Wiederholungen der inneren Schleife abschätzen zu können, führen wir als Hilfskonstrukt eine Menge M ein mit $M = \{v | v \in V, \delta^+_\Lambda(v) \geq \Delta_\Lambda(G) - c\}$. Zu beachten ist, dass $\delta^+_\Lambda(v)$ und $\Delta_\Lambda(G)$ in M im Laufe der inneren Schleife durch die Pfadänderungen stets zu aktualisieren sind. Zu Beginn eines Schleifendurchlaufs wird somit ein Knoten $u \in M$ gewählt und, sofern ein geeigneter Pfad P gefunden wird, $\delta^+_\Lambda(u)$ wird um c verringert. u gehört dann aber immer noch M an. Für den Knoten p_l am Ende des Pfades P gilt im Gegenzug, dass zum einen $p_l \notin M$, zum anderen $\delta^+_\Lambda(p_l)$ um c vergrößert wird und der Knoten möglicherweise danach zu M hinzugefügt wird. Der Ausgangsgrad eines Knoten verringert sich somit sukzessive, sobald er M angehört und ein solcher Knoten wird M nie verlassen. Der Ausgangsgrad von Knoten außerhalb von M erhöht sich hingegen monoton, bis der Knoten in M liegt oder kein Pfad mehr gefunden wird.

Aus dieser Beobachtung mit der Menge M können wir schließen, dass jeder Knoten nur begrenzt oft Ausgangspunkt eines Pfades sein kann. Dies können wir mit dem Grad des Knotens nach oben abschätzen. Auch kann es vorkommen, dass alle Knoten irgendwann in die Menge M aufgenommen werden. Somit können

wir eine obere Schranke für die Anzahl der zu findenden Pfade und somit auch für die Durchläufe der inneren Schleife angeben als $\sum_{v \in V} grad(v) = 2 \cdot |E| = O(|E|)$.

Setzt man nun alle Teile des Algorithmus zusammen, so erhält man eine Gesamtlaufzeit von $f(k) \cdot O(|E| + |E| \cdot |E|) = f(k) \cdot O(|E|^2)$ Zeit. Dies liegt in $\mathcal{FPT}$. Somit ist auch die Laufzeitbehauptung bewiesen.

Mit der gezeigten Laufzeit und der Korrektheit des Algorithmus haben wir somit gezeigt: MOO(k) $\in \mathcal{FPT}$.

□

Hinweise: Gegenüber dem ursprünglichen Algorithmus von Asahiro et al. [2] haben wir einige Veränderungen am Algorithmus vornehmen müssen, die sich hauptsächlich auf die Komplikationen durch die Berücksichtigen des Kantengewichts c zurückführen lassen. Asahiro et al. dürfen in ihrem polynomiellen Spezialfall ohne Beschränkung der Allgemeinheit $c = 1$ annehmen. Dies ist für den hier betrachteten allgemeinen Fall nicht zulässig. Ein Beispiel dafür ist in Abbildung 4.10 gezeigt.

Asahiro et al. [2] haben zudem eine verbesserte Variante des Algorithmus „Reverse“ gezeigt, welche eine Verbesserung der Laufzeit auf $O(|E|^{3/2})$ verspricht. Der Kerngedanke ist hier, dass nicht nur ein Pfad pro Schleifendurchlauf getauscht werden soll, sondern alle auffindbaren knotendisjunkten Pfade auf einmal. Dies ließe sich ebenfalls auf den hier gezeigten Algorithmus für MOO(k) übertragen und sollte bei praktischen Anwendungen berücksichtigt werden. Ebenso sind an einigen Stellen Abkürzungen für Spezialfälle denkbar.

Zuletzt sei noch angemerkt, dass dieser Algorithmus auch derart erweitert werden kann, dass zusätzlich zu den Kantengewichten auch *Knotengewichte* $\in \mathbb{N}_0$ möglich sind, wie sie auch im folgenden Abschnitt betrachtet werden. Dies liegt insbesondere daran, dass es äquivalent ist, einerseits Kanten mit fester Orientierung zu betrachten oder andererseits den Knoten, aus den diese fest orientierten Kanten ausgehen, als Knotengewicht einfach diese Kantengewichte hinzuzufügen

und die Kanten dann zu löschen. Die Arbeit des obigen Algorithmus kann auch derart angesehen werden, dass 2^k verschiedene Belegungen von Knotengewichten im zu untersuchenden Graphen durchprobiert werden, eben abhängig vom jeweiligen Λ_k. Weitere feste Knotengewichte würden die Laufzeit nicht erhöhen.

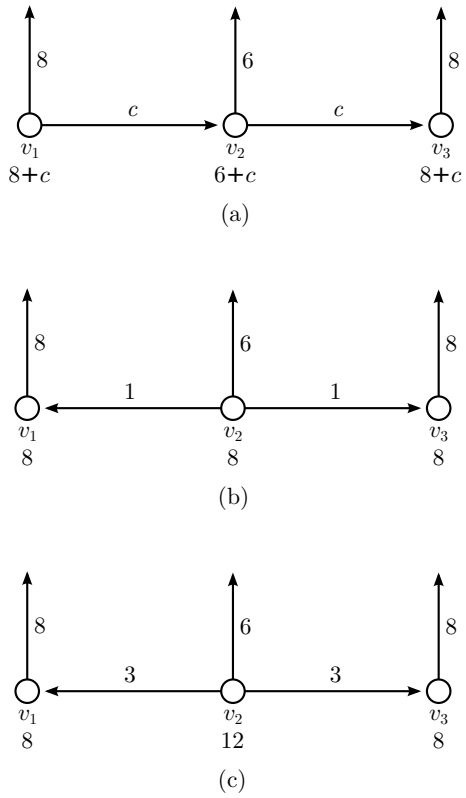


Abbildung 4.10: Gegenbeispiel zu Annahme, dass $c = 1$ gesetzt werden darf. Die nach oben weisenden Kanten gehören zu E_k und sind bereits fest gewählt. In (a) ist die Ausgangslage zu sehen. Es gilt: $\delta^+_\Lambda(v_1) = 8 + c$, $\delta^+_\Lambda(v_2) = 6 + c$ und $\delta^+_\Lambda(v_3) = 8$. In (b) wird nun $c = 1$ angenommen und über den Algorithmus zu MOO(k) die offensichtliche Lösung $\delta^+_\Lambda(v_1) = \delta^+_\Lambda(v_2) = \delta^+_\Lambda(v_3) = 8$ konstruiert. In (c) hingegen gilt $c = 3$ und die Lösung von $b)$ ist hier keine Lösung, da der maximale gewichtete Ausgangsgrad 12 ist aber die Situation in $a)$ mit $c = 3$ nur einen maximalen gewichteten Ausgangsgrad von 11 hat. Folglich muss c stets berücksichtigt werden.

4.4 Ergebnis: $\mathsf{MOO}_{d\leq 2}(k)$ liegt in $\mathcal{FPT}$

In diesem Abschnitt sehe wir, dass auch eine weitere Variante von MOO in $\mathcal{FPT}$ liegt. Hierzu betrachten wir zunächst jedoch den Spezialfall, dass der betrachtete Graph P pfadförmig ist, das heißt, alle Knoten von P hängen zusammen und haben den Grad 2 bis auf zwei Randknoten, welche nur den Grad 1 haben. Insbesondere ist also kein Zyklus enthalten. Die Kanten haben ein Gewicht $\in \mathbb{N}$. Wir bezeichnen die Kantengewichte mit einem β. Hinzu kommt, dass wir ebenfalls gestatten, dass die Knoten ein Gewicht $\in \mathbb{N}_0$ haben, welches dem gewichteten Ausgangsgrad hinzugefügt wird. Diese *Knotengewichte* bezeichnen wir mit einem α. In Abbildung 4.11 ist ein Beispiel für einen solchen Graphen zu sehen.

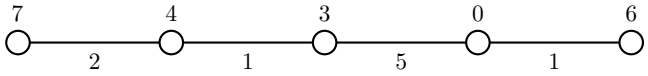


Abbildung 4.11: Ein pfadförmiger Graph mit fünf gewichteten Knoten. Sowohl Kanten als auch Knoten haben ein Gewicht. Das Kantengewicht zählt wie üblich zum gewichteten Ausgangsgrad desjenigen Knotens, aus dem die zugehörige Kante ausgeht. Das Knotengewicht wird zum gewichteten Ausgangsgrad hinzu addiert. Betrachten wir etwa den mittleren Knoten mit dem Gewicht 3. Sind beide inzidenten Kanten aus ihm ausgehend, so beträgt sein gewichteter Ausgangsgrad $1+5+3=9$. Sind beide inzidenten Kanten jedoch zu dem Knoten hin gerichtet, so beträgt der gewichtete Ausgangsgrad des Knotens immer noch 3.

Lemma 4.4. *Gegeben sei ein pfadförmiger Graph $P=(V,E,w_V,w_E)$ mit $|V|=n$ Knoten, $V=\{v_1,\ldots,v_n\}$, mit beliebigem Gewicht $\alpha_i := w_V(v_i) \in \mathbb{N}_0\ \forall v_i \in V$, sowie $|E|=n-1$ Kanten, $E=\{\{v_i,v_{i+1}\}|\ i\in\{0,\ldots,n-1\}\}$, mit beliebigem Gewicht $\beta_i := w_E(\{v_i,v_{i+1}\})$ aus $\mathbb{N}\ \forall i\in\{0,\ldots,n-1\}$. Es gilt: $\Delta_{OPT}(P)$ für eine optimale Orientierung OPT lässt sich in $O(|V|)$ finden.*

Beweis des Lemmas 4.4. Wir beweisen nachfolgend dieses Lemma indem wir mit Algorithmus 2 (MOO_Pfad(P)) einen Algorithmus angeben, der besagte Aufgabenstellung in $O(|V|)$ Zeit löst. Sei im Folgenden der pfadförmige Graph

stets in gerader Linie von links nach rechts so gezeichnet, dass auf der linken Seite der Graph mit dem Knoten v_1 beginnt und auf der rechten Seite mit dem Knoten v_n endet. Entsprechend sind die Graphiken dieses Abschnittes gezeichnet und die verwendeten Richtungen Links und Rechts zu verstehen.

Algorithmus 2: MOO_Pfad(P)

Eingabe : Ein Graph $P = (V, E, w_V, w_E)$ mit $|V| = n$ Knoten,
$V = \{v_1, \ldots, v_n\}$, mit beliebigem
Gewicht $\alpha_i := w_V(v_i) \in \mathbb{N}_0 \; \forall v_i \in V$, sowie $|E| = n - 1$ Kanten,
$E = \{\{v_i, v_{i+1}\} |\; i \in \{1, \ldots, n-1\}\}$, mit beliebigem Gewicht
$\beta_i := w_E(\{v_i, v_{i+1}\}) \in \mathbb{N} \; \forall i \in \{1, \ldots, n-1\}$

Ausgabe : $\Delta_\Lambda(P)$ für eine optimale Orientierung Λ

1 Lege Variablen $x_1, \ldots, x_{n+1}$, sowie $y_1, \ldots, y_{n+1}$ an
2 Setze $\beta_0 := 0$, $\beta_n := 0$, $x_{n+1} := 0$, $y_{n+1} := 0$
3 **für** $i := n$ **bis** 1 **tue**
4 $\quad x_i := \min(\max(\alpha_i + \beta_i, x_{i+1}), \max(\alpha_i, y_{i+1}))$
5 $\quad y_i := \min(\max(\alpha_i + \beta_i + \beta_{i-1}, x_{i+1}), \max(\alpha_i + \beta_{i-1}, y_{i+1}))$
6 Gib x_1 aus

Funktionsweise: Der Algorithmus läuft nach der Deklaration der nötigen Variablen in Zeile 1 und Initialisierung von Randwerten ($\beta_0 = \beta_n = x_{n+1} = y_{n+1} = 0$) in Zeile 2 in einer Schleife von rechts nach links über den pfadförmigen Graphen (Zeile 3). Hierbei werden pro Knoten die Werte

$$x_i = \min(\max(\alpha_i + \beta_i, x_{i+1}), \max(\alpha_i, y_{i+1}))$$

und

$$y_i = \min(\max(\alpha_i + \beta_i + \beta_{i-1}, x_{i+1}), \max(\alpha_i + \beta_{i-1}, y_{i+1}))$$

berechnet (Zeilen 4 und 5). Wie im Abschnitt über Korrektheit nachgewiesen wird, gilt folgende Interpretation dieser Werte:

- x_i ist der minimale maximale gewichtete Ausgangsgrad $\Delta_\Lambda(P)$ mit Λ optimal wenn man den pfadförmigen Graphen lediglich vom Knoten v_i bis Knoten v_n betrachtet.
- y_i ist der minimale maximale gewichtete Ausgangsgrad $\Delta_\Lambda(P)$ mit Λ optimal wenn man den pfadförmigen Graphen lediglich vom Knoten v_i bis Knoten v_n betrachtet und zusätzlich das Kantengewicht β_{i-1} der Kante $\{v_{i-1}, v_i\}$ zum Knoten v_i hinzu zählt.

Die Werte x_i und y_i sind also, rechts am pfadförmigen Graphen beginnend, das bisher gefundene Optimum und die beiden Werte unterscheiden sich lediglich darin, wie die nächste folgende Kante $\{v_{i-1}, v_i\}$ orientiert wird.

Am Ende (Zeile 6) wird x_1 ausgegeben, was dann genau der maximale gewichtete Ausgangsgrad unter einer optimalen Orientierung ist. Zudem ist $x_1 = y_1$ wegen $\beta_0 = 0$. In Abbildung 4.12 ist ein schematischer Überblick über die Variablen zu sehen.

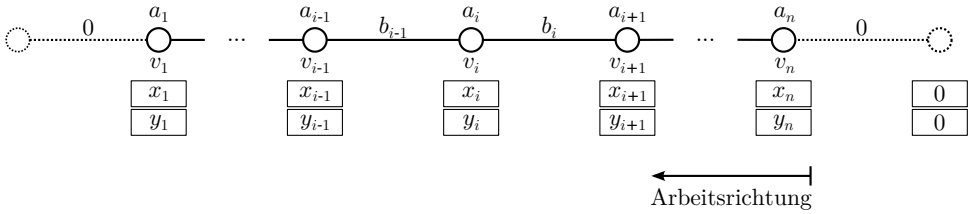


Abbildung 4.12: Überblick über die verwendeten Variablen in Algorithmus 2. Jeder Knoten v_i hat ein Knotengewicht α_i und jede Kante ein Kantengewicht β_i. Zudem werden, beim letzten Knoten beginnend und nach links voran arbeitend, für jeden Knoten die beiden Werte $x_i = \min(\max(\alpha_i + \beta_i, x_{i+1}), \max(\alpha_i, y_{i+1}))$ und $y_i = \min(\max(\alpha_i + \beta_i + \beta_{i-1}, x_{i+1}), \max(\alpha_i + \beta_{i-1}, y_{i+1}))$ berechnet. Für die Randwerte gilt $\beta_0 = \beta_n = x_{n+1} = y_{n+1} = 0$.

Laufzeit: Die Laufzeitbehauptung zeigt sich wie folgt: Es ist $|V| = n$. Zeile 1 geschieht in $O(n)$, Zeile 2 in $O(1)$. Die Schleife wird n-mal wiederholt und die Berechnung von x_i und y_i passiert jeweils in konstanter Zeit. Dies ergibt für

die gesamte Schleife aus Zeile 3 eine Laufzeit von $O(n)$. Die Ausgabe benötigt ebenfalls konstante Zeit. Somit gilt die Behauptung, dass der Algorithmus die Lösung in $O(|V|)$ Zeit ermittelt.

Korrektheit: Die Korrektheit des Algorithmus zeigen wir durch vollständige Induktion über die Knoten des Graphen. Hierbei betrachten wir ein immer größer werdendes rechtes Ende des pfadförmigen Graphen $P = (V, E, w_V, w_E)$ mit den Mengen $V = \{v_1, \ldots, v_n\}$ und $E = \{\{v_i, v_{i+1}\} |\ i \in \{1, \ldots, n-1\}\}$. Sei hierzu $P_k = (V_k, E_k, w_V, w_E)$ derjenige Teilgraph von P, den man erhält, wenn man nur die Knoten von v_k bis v_n und dazwischenliegende Kanten betrachtet. Das bedeutet, es gilt $V_k = \{v_k, \ldots, v_n\} \subseteq V$ und $E_k = \{\{v_i, v_{i+1}\} |\ i \in \{k, \ldots, n-1\}\} \subseteq E$. Offenbar ist $P = P_1$.

Zudem benötigen wir noch eine Variante von P_k, bei der noch eine weitere Kante beachtet wird: Sei hierzu $\widehat{P}_k = (V_k, E_k, \widehat{w}_V, w_E)$ derjenige Teilgraph von P, den man erhält, wenn man nur die Knoten von v_k bis v_n und dazwischenliegende Kanten betrachtet und zudem das Gewicht β_{k-1} der Kante $\{v_{k-1}, v_k\}$ zum Knotengewicht α_k des Knotens v_k hinzu addiert. Folglich setzen wir das Knotengewicht $\widehat{\alpha}_k = \widehat{w}_V(v_k) := w_V(v_k) + \beta_{k-1} = \alpha_k + \beta_{k-1}$ und behalten im Übrigen $\widehat{w}_V(v_i) = w_V(v_i)\ \forall i \in \{1, \ldots, n-1\} \setminus \{k\}$.

Der Teilgraph P_k ist also das rechte Ende von P beginnend ab Knoten v_k und $\widehat{P}_k$ ist dieses rechte Ende unter Berücksichtigung der nächsten Kante auf der linken Seite.

Behauptung: Sei OPT eine optimale Orientierung, so gilt für die im Algorithmus berechneten x- und y-Werte:

$$x_k = \min(\max(\alpha_k + \beta_k, x_{k+1}), \max(\alpha_k, y_{k+1})) = \Delta_{OPT}(P_k)$$

$$y_k = \min(\max(\alpha_k + \beta_k + \beta_{k-1}, x_{k+1}), \max(\alpha_k + \beta_{k-1}, y_{k+1})) = \Delta_{OPT}(\widehat{P}_k)$$

Mit anderen Worten: Die im Algorithmus berechneten x- und y-Werte sind jeweils der minimale maximale gewichtete Ausgangsgrad der bisher betrachteten Knoten, beginnend bei v_n bis v_k, wobei bei y_k noch die Kante $\{v_{k-1}, v_k\}$ berücksichtigt wird.

Diese Behauptung nun zeigen wir per vollständiger Induktion, beginnend beim Knoten v_n.

Induktionsanfang: Wir beginnen bei $k = n$. P_n ist lediglich der Graph bestehend aus dem Knoten v_n und keinen Kanten. Somit gilt offenbar $\Delta_{OPT}(P_n) = \alpha_n$. Zudem gilt durch die gesetzten Randwerte $\beta_n = x_{n+1} = y_{n+1} = 0$, dass

$$\begin{aligned} x_n &= \min(\max(\alpha_n + \beta_n, x_{n+1}), \max(\alpha_n, y_{n+1})) \\ &= \min(\max(\alpha_n + 0, 0), \max(\alpha_n, 0)) \\ &= \min(\alpha_n, \alpha_n) = \alpha_n = \Delta_{OPT}(P_n) \end{aligned}$$

Analog ist $\widehat{P}_k$ der Graph bestehend aus dem Knoten v_n und dem Knotengewicht $\widehat{\alpha}_n = \alpha_n + \beta_{n-1}$. Entsprechend gilt $\Delta_{OPT}(\widehat{P}_n) = \widehat{\alpha}_n$. Auch hier gilt durch die gesetzten Randwerte $\beta_n = x_{n+1} = y_{n+1} = 0$, dass

$$\begin{aligned} y_n &= \min(\max(\alpha_n + \beta_n + \beta_{n-1}, x_{n+1}), \max(\alpha_n + \beta_{n-1}, y_{n+1})) \\ &= \min(\max(\alpha_n + 0 + \beta_{n-1}, 0), \max(\alpha_n + \beta_{n-1}, 0)) \\ &= \min(\alpha_n + \beta_{n-1}, \alpha_n + \beta_{n-1}) = \widehat{\alpha}_n = \Delta_{OPT}(\widehat{P}_n) \end{aligned}$$

Somit ist die Korrektheit des Induktionsanfangs gezeigt.

Induktionshypothese: Die Behauptung gelte für x_{i+1} und y_{i+1}.

Induktionsschritt: Von $i + 1$ nach i ($i \geq 1$): Im Schritt $i + 1$ wurde der Graph vom End-Knoten v_n bis hin zum Knoten v_{i+1} betrachtet. Diese Betrachtung er-

weitert sich nun im Induktionsschritt auf den Knoten v_i. Um nun $\Delta_{OPT}(P_i)$ und $\Delta_{OPT}(\widehat{P}_i)$ zu bestimmen, untersuchen wir die beiden für den Knoten v_i relevanten Kanten: $\{v_{i-1}, v_i\}$ mit Gewicht β_{i-1} und $\{v_i, v_{i+1}\}$ mit Gewicht β_i. Beide Kanten können jeweils nach links oder nach rechts orientiert sein, dies ergibt insgesamt vier mögliche Orientierungen. Mindestens eine dieser vier Orientierungen führt zum minimalen maximalen gewichteten Ausgangsgrad, da die Kanten eine Orientierung haben müssen. Wir unterscheiden nun im Folgenden die vier möglichen Fälle, wie die Kanten orientiert sein können:

- **Fall 1a (→→):** Die Kante $\{v_{i-1}, v_i\}$ ist von links nach rechts orientiert und auch die Kante $\{v_i, v_{i+1}\}$ ist von links nach rechts orientiert.
- **Fall 1b (→←):** Die Kante $\{v_{i-1}, v_i\}$ ist von links nach rechts orientiert; die Kante $\{v_i, v_{i+1}\}$ ist von rechts nach links orientiert.
- **Fall 2a (←→):** Die Kante $\{v_{i-1}, v_i\}$ ist von rechts nach links orientiert; die Kante $\{v_i, v_{i+1}\}$ ist von links nach rechts orientiert.
- **Fall 2b (←←):** Die Kante $\{v_{i-1}, v_i\}$ ist von rechts nach links orientiert und auch die Kante $\{v_i, v_{i+1}\}$ ist von rechts nach links orientiert.

Diese vier Fälle sind auch nochmal in Abbildung 4.13 dargestellt und werden nun im Detail von uns betrachtet:

Fall 1a (→→): Da die Kante $\{v_{i-1}, v_i\}$ hin zum Knoten v_i orientiert ist, somit also nicht zum gewichteten Ausgangsgrad von v_i hinzu zählt, ist diese Kante für die Betrachtung des Teilgraphen vom rechten Ende bis zum Knoten v_i nicht relevant. Wir untersuchen in diesem Fall 1a folglich $\Delta_{OPT}(P_i)$ unter der Bedingung, dass im Optimalfall die Kante $\{v_i, v_{i+1}\}$ von links nach rechts zu orientieren wäre. Also zählt das Kantengewicht β_i zum gewichteten Ausgangsgrad von v_i hinzu. Der gewichtete Ausgangsgrad von v_i ist somit $\alpha_i + \beta_i$.

Nach Induktionshypothese gilt: $\Delta_{OPT}(P_{i+1}) = x_{i+1}$. Abhängig von all diesen Werten nun berechnet sich $\Delta_{OPT}(P_i)$. Entweder $\alpha_i + \beta_i$ ist größer als x_{i+1} und somit der neue minimale maximale gewichtete Ausgangsgrad im Fall 1a oder x_{i+1}

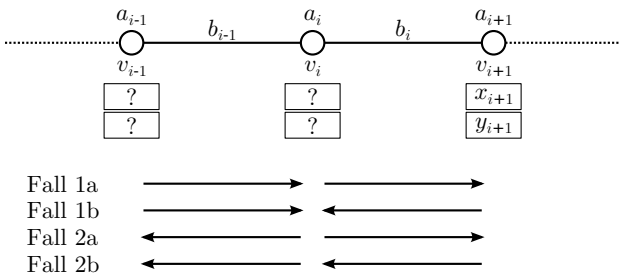


Abbildung 4.13: Fallunterscheidung der Orientierung der beiden für v_i relevanten Kanten.
Relevant sind die Kanten $\{v_{i-1}, v_i\}$ und $\{v_i, v_{i+1}\}$. Mindestens eine dieser vier Möglichkeiten ist optimal für den minimalen maximalen gewichteten Ausgangsgrad, da die Kanten schließlich eine Orientierung benötigen. Im Folgenden unterscheiden wir die Fälle 1a, 1b, 2a und 2b wie angezeigt.

bleibt der minimale maximale gewichtete Ausgangsgrad, auch wenn der Knoten v_i hinzu kommt. Es gilt somit:

$$\Delta_{OPT}(P_i) = \max(\alpha_i + \beta_i, x_{i+1})$$

unter der Bedingung, dass im Optimalfall die Kante $\{v_i, v_{i+1}\}$ von links nach rechts zu orientieren wäre.

Fall 1b ($\rightarrow\leftarrow$): Analog zum Fall 1a untersuchen wir im Fall 1b $\Delta_{OPT}(P_i)$ unter der Bedingung, dass im Optimalfall die Kante $\{v_i, v_{i+1}\}$ von rechts nach links zu orientieren ist und können die Kante $\{v_{i-1}, v_i\}$ ignorieren. Wenn die Kante $\{v_i, v_{i+1}\}$ von rechts nach links zu orientieren ist, zählt das Kantengewicht β_i zum gewichteten Ausgangsgrad von v_{i+1} hinzu. Dies können wir hierdurch berücksichtigen, in dem wir auf $\Delta_{OPT}(\widehat{P}_{i+1})$ zurückgreifen, denn hier wird das Kantengewicht β_i eben gerade per Definition zum gewichteten Ausgangsgrad von v_{i+1} hinzu addiert. Der gewichtete Ausgangsgrad des neu betrachteten Knotens v_i bleibt folglich bei α_i.

Nach Induktionshypothese gilt: $\Delta_{OPT}(\widehat{P}_{i+1}) = y_{i+1}$. Auch hier unterscheiden sich nun zwei Fälle zur Berechnung von $\Delta_{OPT}(P_i)$: Entweder α_i ist größer als y_{i+1} und somit der neue minimale maximale gewichtete Ausgangsgrad im Fall 1b oder y_{i+1} bleibt der minimale maximale gewichtete Ausgangsgrad, auch wenn der Knoten v_i hinzu kommt. Es gilt somit:

$$\Delta_{OPT}(P_i) = \max(\alpha_i, y_{i+1})$$

unter der Bedingung, dass im Optimalfall die Kante $\{v_i, v_{i+1}\}$ von rechts nach links zu orientieren ist.

Zusammenfassung der Fälle 1a und 1b: Wir haben mit den obigen Fällen 1a und 1b also gezeigt, dass gilt:

$$\Delta_{OPT}(P_i) = \begin{cases} \max(\alpha_i + \beta_i, x_{i+1}), & \text{im Fall 1a} \\ \max(\alpha_i, y_{i+1}), & \text{im Fall 1b} \end{cases}$$

Also abhängig von der korrekten Orientierung der Kante $\{v_i, v_{i+1}\}$ kennen wir den Wert von $\Delta_{OPT}(P_i)$. Die korrekte Orientierung ist diejenige, die den kleineren Wert für $\Delta_{OPT}(P_i)$ bildet, also gerade das Minimum zwischen den beiden Alternativen. Somit gilt:

$$\Delta_{OPT}(P_i) = \min(\max(\alpha_i + \beta_i, x_{i+1}), \max(\alpha_i, y_{i+1}))$$

Dies entspricht genau der Definition des Wertes x_i. Somit ist der erste Teil der Behauptung im Induktionsschritt gezeigt: $x_i = \Delta_{OPT}(P_i)$.

Fall 2a ($\leftarrow\rightarrow$) und Fall 2b ($\leftarrow\leftarrow$): Diese beiden Fälle funktionieren ganz analog zu den Fällen 1a und 1b mit dem Unterschied, dass die Kante $\{v_{i-1}, v_i\}$ weg vom Knoten v_i orientiert ist, somit also zu dessen gewichteten Ausgangsgrad hinzu addiert werden muss. Dies entspricht genau der Betrachtung von $\Delta_{OPT}(\widehat{P}_i)$, einmal unter der Bedingung, dass die Kante $\{v_i, v_{i+1}\}$ von links nach rechts

(Fall 2a) zu orientieren ist, und das andere mal unter der Bedingung, dass die Kante $\{v_i, v_{i+1}\}$ von rechts nach links (Fall 2b) orientiert werden muss.

Im Fall 2a ist der gewichtete Ausgangsgrad von v_i somit $\alpha_i + \beta_i + \beta_{i-1}$ und muss mit $\Delta_{OPT}(P_{i+1})$ verglichen werden, wobei nach Induktionshypothese gilt, dass $\Delta_{OPT}(P_{i+1}) = x_{i+1}$ ist. Im Fall 2b ist der gewichtete Ausgangsgrad von v_i lediglich $\alpha_i + \beta_{i-1}$, muss jedoch (wie im Fall 1b) mit $\Delta_{OPT}(\widehat{P}_{i+1})$ verglichen werden, da β_i nun zum Knoten v_{i+1} hinzu zählt. Nach Induktionshypothese gilt, dass $\Delta_{OPT}(\widehat{P}_{i+1}) = y_{i+1}$ ist.

Somit berechnet sich $\Delta_{OPT}(\widehat{P}_i)$, abhängig von der korrekten Orientierung der Kante $\{v_i, v_{i+1}\}$ wie folgt:

$$\Delta_{OPT}(\widehat{P}_i) = \begin{cases} \max(\alpha_i + \beta_i + \beta_{i-1}, x_{i+1}), & \text{im Fall 2a} \\ \max(\alpha_i + \beta_{i-1}, y_{i+1}), & \text{im Fall 2b} \end{cases}$$

Auch hier gilt wieder, dass die korrekte Variante diejenige ist, die den kleineren Wert für $\Delta_{OPT}(\widehat{P}_i)$ bildet, also gerade wieder das Minimum zwischen den beiden Alternativen. Somit gilt:

$$\Delta_{OPT}(\widehat{P}_i) = \min(\max(\alpha_i + \beta_i + \beta_{i-1}, x_{i+1}), \max(\alpha_i + \beta_{i-1}, y_{i+1}))$$

Dies ist genau die Definition von y_i. Hierdurch ist auch der zweite Teil der Behauptung gezeigt: $y_i = \Delta_{OPT}(\widehat{P}_i)$. Weiterhin ist somit die Korrektheit des gesamten Induktionsschritt gezeigt und die Behauptung gilt.

Mit $P = P_1$ und der Behauptung folgt direkt, dass für den vom Algorithmus ausgegebenen Wert gilt: $x_1 = \Delta_{OPT}(P_1) = \Delta_{OPT}(P)$. Der Algorithmus ist also korrekt. Gemeinsam mit der oben gezeigten Laufzeit ist damit der Beweis von Lemma 4.4 erbracht.

□

Implementierungshinweis zum Algorithmus 2: Durch die Verwendung eines x- und eines y-Wertes für jeden Knoten wird insgesamt $O(n)$ Platz benötigt. Dies ist allerdings unnötig und nur für die Verständlichkeit des Beweises hier so durchgeführt. Tatsächlich ist es auch ausreichend jeweils nur den Wert von x und y vom zuletzt betrachteten Knoten zu speichern, da auf die Werte weiter zurückliegender Knoten nicht mehr zugegriffen werden muss. Der Algorithmus lässt sich also in $O(1)$ Platz realisieren.

Für die weitere Verwendung des Ergebnisses aus Lemma 4.4 benötigen wir noch zwei weitere Algorithmen um mit Hilfe des oben beschriebenen Algorithmus 2 (MOO_Pfad(P)) auch eine optimale Orientierung erzeugen zu können.

Algorithmus 3: MOO_Pfadorientierung(P)

Eingabe : Ein Graph $P = (V, E, w_V, w_E)$ mit $|V| = n$ Knoten,
$V = \{v_1, \ldots, v_n\}$, mit beliebigem
Gewicht $\alpha_i := w_V(v_i) \in \mathbb{N}_0 \; \forall v_i \in V$, sowie $|E| = n - 1$ Kanten,
$E = \{\{v_i, v_{i+1}\} |\; i \in \{1, \ldots, n-1\}\}$, mit beliebigem Gewicht
$\beta_i := w_E(\{v_i, v_{i+1}\}) \in \mathbb{N} \; \forall i \in \{1, \ldots, n-1\}$

Ausgabe : Eine Orientierung Λ mit $\Delta_\Lambda(P)$ minimal

```
1 Δ := MOO_Pfad(P)
2 Λ := ∅
3 für i := 1 bis n − 1 tue
4 |   wenn α_i + β_i ≤ Δ dann
5 |   |   Λ := Λ ∪ {(v_i, v_{i+1})}
6 |   sonst
7 |   |   Λ := Λ ∪ {(v_{i+1}, v_i)}
8 |   |   α_{i+1} := α_{i+1} + β_i
9 Gib Λ aus
```

Algorithmus 3 (MOO_Pfadorientierung(P)): Dieser Algorithmus konstruiert in $O(|V|)$ Zeit eine optimale Orientierung für einen pfadförmigen Graphen. Hierzu ermittelt der Algorithmus zunächst (durch Verwendung von Algorithmus 2) den minimalen maximalen gewichteten Ausgangsgrad Δ des pfadförmigen Graphen (Zeile 1). In einer Schleife (Zeile 3) wird nun über den pfadförmigen Graphen

iteriert und sukzessive die Orientierung Λ aufgebaut. Hierzu wird die folgende Überlegung verwendet:

Wenn das Gewicht des äußersten Knotens und das Kantengewicht seiner einzigen inzidenten Kante kleiner oder gleich Δ ist (Zeile 4), so ist es erlaubt, die Kante von diesem äußersten Knoten weg zu richten (Zeile 5). Dieser Knoten und die Kante können dann im weiteren Verlauf ignoriert werden und der nächste Knoten ist entsprechend wieder der äußerste Knoten des Rest-Graphen. Trifft die Bedingung $\alpha_i + \beta_i \leq \Delta$ jedoch nicht zu (Zeile 6), so muss die betreffende Kante zum äußersten Knoten hin gerichtet sein (Zeile 7). Um diese Kante beim nächsten Knoten mit zu berücksichtigen, wird in Zeile 8 das Knotengewicht des nächsten Knotens um das Kantengewicht der betreffenden Kante erhöht. Auch in diesem Fall kann nun für den weiteren Verlauf der äußerste Knoten und dessen inzidente Kante ignoriert werden und der nächste Knoten kann wieder als äußerster Knoten des Rest-Graphen betrachtet werden.

Am Ende wird in Zeile 9 die so gefundene Orientierung ausgegeben. Die Laufzeitbehauptung folgt direkt daraus, dass jede Kante nur einmal betrachtet werden muss und diese Betrachtung in konstanter Zeit durchgeführt wird. Zudem ist mit $|V| = |E| + 1$ auch $O(|E|) = O(|V|)$.

Algorithmus 4 (MOO_Zyklusorientierung(Z)): Dieser Algorithmus konstruiert in $O(|V|)$ Zeit eine optimale Orientierung für einen zyklusförmigen Graphen. Ein Graph ist zyklusförmig, falls alle Knoten den Grad 2 haben, es sich also um einen pfadförmigen Graphen handelt, dessen beide Endknoten ebenfalls durch eine Kante verbunden sind. Eine formale Definition ist in der Eingabespezifikation von Algorithmus 4 zu finden.

Der Algorithmus führt das Problem auf Algorithmus 3 für pfadförmige Graphen zurück: Hierzu wird eine Kante aus dem zyklusförmigen Graphen gewählt und es wird für beide möglichen Orientierungen dieser Kante untersucht, wie sich der pfadförmige Rest-Graph orientieren lässt und die bessere der beiden Varianten

Algorithmus 4: MOO_Zyklusorientierung(Z)

Eingabe : Ein Graph $Z = (V, E, w_V, w_E)$ mit $|V| = n \geq 3$ Knoten,
$V = \{v_1, \ldots, v_n\}$, mit beliebigem
Gewicht $\alpha_i := w_V(v_i) \in \mathbb{N}_0 \; \forall v_i \in V$, sowie $|E| = n - 1$ Kanten,
$E = \{\{v_i, v_{i+1}\} |\; i \in \{1, \ldots, n-1\}\} \cup \{\{v_n, v_1\}\}$, mit beliebigem
Gewicht $\beta_i := w_E(\{v_i, v_{i+1}\}) \in \mathbb{N} \; \forall i \in \{1, \ldots, n-1\}$ sowie
$\beta_n := w_E(\{v_n, v_1\}) \in \mathbb{N}$

Ausgabe : Eine Orientierung Λ mit $\Delta_\Lambda(Z)$ minimal

1 $\Lambda := \emptyset$
2 Erzeuge Graph $P_1 := (V, E', w'_V, w_E)$ aus Z mit $E' := E \setminus \{\{v_n, v_1\}\}$, $w'_V(v_1) := w_V(v_1) + \beta_n$ und $w'_V(v_i) := w_V(v_i) \forall i \in \{2, \ldots, n\}$ und berechne $\Delta_1 :=$ MOO_Pfad(P_1)
3 Erzeuge Graph $P_n := (V, E', w'_V, w_E)$ aus Z mit $E' := E \setminus \{\{v_n, v_1\}\}$, $w'_V(v_n) := w_V(v_n) + \beta_n$ und $w'_V(v_i) := w_V(v_i) \forall i \in \{1, \ldots, n-1\}$ und berechne $\Delta_n :=$ MOO_Pfad(P_n)
4 **wenn** $\Delta_1 \leq \Delta_n$ **dann**
5 | Erzeuge Orientierung $\Lambda_1 :=$ MOO_Pfadorientierung(P_1)
6 | Setze $\Lambda := \Lambda_1 \cup \{(v_1, v_n)\}$
7 **sonst**
8 | Erzeuge Orientierung $\Lambda_n :=$ MOO_Pfadorientierung(P_n)
9 | Setze $\Lambda := \Lambda_n \cup \{(v_n, v_1)\}$
10 Gib Λ aus

als Lösung ausgegeben. Im Fall hier wird die (willkürlich nummerierte) letzte Kante $\{v_n, v_1\}$ gewählt und in den Zeilen 2 und 3 die jeweiligen pfadförmigen Graphen konstruiert, wenn die Kante entfernt wird und ihr Kantengewicht einmal zu α_1 und einmal zu α_n hinzuaddiert wird. Für beide Graphen wird mit Algorithmus 2 der minimale maximale gewichtete Ausgangsgrad berechnet. Abhängig davon welcher der beiden Werte kleiner ist (Zeile 4) wird Algorithmus 3 verwendet um eine optimale Orientierung des pfadförmigen Graphen zu bestimmen (Zeile 5 bzw. 8). Zuletzt wird noch die korrekte Orientierung der vormals gelöschten Kante der Ergebnisorientierung Λ hinzugefügt (Zeile 6 bzw. 9) und Λ wird in Zeile 10 ausgegeben.

Die Laufzeitbehauptung folgt daraus, dass alle Einzelschritte höchstens $O(|V|)$ Zeit beanspruchen und der Algorithmus keine Schleifen oder Wiederholungen hat.

Parametrisierung: Nachdem wir die Algorithmen 2, 3 und 4 untersucht haben, besitzen wir alle Hilfsmittel um ein weiteres von Ergebnis der Diplomarbeit zeigen zu können. Wir untersuchen hierbei folgende parametrisierte Variante von Minimum Outdegree Orientation:

> Minimum Outdegree Orientation mit k überzähligen Kanten (kurz $\text{MOO}_{d\leq 2}(k)$):
> Gegeben: Ein einfacher Graph $G = (V, E = E_2 \cup E_k, w)$ mit $|E_k| = k$. Die Kanten aus E haben alle ein beliebiges Gewicht aus $\mathbb{N}$ und für den Teilgraph $TG = (V, E_2, w)$ gilt, dass $\forall v \in V : grad(v) \leq 2$.
> Gesucht: Eine Orientierung Λ, sodass $\Delta_\Lambda(G)$ minimal ist.

$\text{MOO}_{d\leq 2}(k)$ betrachtet also Graphen, deren Knoten großteils nur einen Grad ≤ 2 haben, also Graphen die aus pfadförmigen und zyklusförmigen Strukturen bestehen. Lediglich k überzählige Kanten verhindern, dass alle Knoten einen Grad ≤ 2 haben. In Abbildung 4.14 ist ein Beispiel für einen solchen Graphen dargestellt. Zu beachten ist, dass $\text{MOO}_{d\leq 2}(k)$ kein Spezialfall von der im letzten Abschnitt

untersuchten Variante $\mathrm{MOO}(k)$ ist. $\mathrm{MOO}_{d\leq 2}(k)$ gestattet jeder Kante ein individuelles Gewicht aus $\mathbb{N}$.

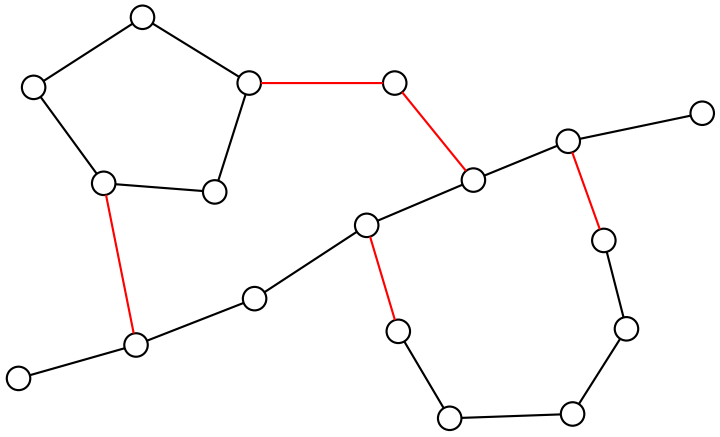


Abbildung 4.14: Beispiel einer Eingabeinstanz von $\mathrm{MOO}_{d\leq 2}(k)$.
Die schwarzen Kanten bilden die Kantenmenge E_2 und die roten Kanten bilden die Kantenmenge E_k, es gilt also $k = 5$. Die Aufteilung von Kanten eines Graphen in E_2 und E_k ist nicht eindeutig und in diesem Beispiel ist k auch nicht minimal. Betrachtet man nur die schwarzen Kanten E_2, so haben alle Knoten dieses Teilgraphs einen Grad ≤ 2. Der Teilgraph zerfällt in mehrere Zusammenhangskomponenten: Links oben eine zyklusförmige Zusammenhangskomponente, sowie drei pfadförmige Zusammenhangskomponenten. Zu beachten ist, dass der Knoten oben Mitte ebenfalls eine pfadförmige Zusammenhangskomponente ist, wenn auch nur bestehend aus einem Knoten.

$\mathrm{MOO}_{d\leq 2}(k)$ ist $\mathcal{NP}$-vollständig, weil MOO mit hinreichend großem k (eine natürliche obere Schranke ist die Anzahl der Kanten) ein Spezialfall des hier betrachteten Problems $\mathrm{MOO}_{d\leq 2}(k)$ darstellt.

Es gelingt nun folgenden Satz zu zeigen:

Satz 4.5. $\mathrm{MOO}_{d\leq 2}(k) \in \mathcal{FPT}$

Beweis zu Satz 4.5. Der Beweis erfolgt durch Konstruktion eines Algorithmus, welcher $\mathrm{MOO}_{d\leq 2}(k)$ in $f(k) \cdot n^{O(1)}$ Zeit löst, wobei n die Größe der Eingabeinstanz ist. Es handelt sich hierbei um Algorithmus 5 ($\mathrm{MOO}_{d\leq 2}((G, k))$).

Algorithmus 5: $\mathrm{MOO}_{d\leq 2}((G,k))$

Eingabe : Ein einfacher kantengewichteter Graph $G = (V, E_2 \cup E_k, w)$ mit $|E_k| = k$. Zudem $w(e) \in \mathbb{N}\ \forall e \in E = E_2 \cup E_k$ und für den Teilgraph $TG = (V, E_2, w)$ gilt: $\forall v \in V : grad(v) \leq 2$

Ausgabe : Eine Orientierung Λ mit $\Delta_\Lambda(G)$ minimal

```
1  Λ_min := ∅
2  O := Menge aller möglichen Orientierungen der Kanten E_k
3  solange O ≠ ∅ tue
4      Wähle Λ ∈ O und setze O := O \ {Λ}
5      Erzeuge Graphkopie G' := G
6      Setze ∀ Knoten v_i in G' ein Knotengewicht α_i entsprechend Λ
7      Entferne die Kanten E_k aus G'
8      für alle Zusammenhangskomponenten Z von G' tue
9          wenn Z ist zyklusförmig dann
10             Erzeuge Orientierung Λ_Z = MOO_Zyklusorientierung(Z)
11         sonst    /* Z muss pfadförmig sein */
12             Erzeuge Orientierung Λ_Z = MOO_Pfadorientierung(Z)
13         Λ := Λ ∪ Λ_Z
14     wenn Λ_min = ∅ oder Δ_{Λ_min}(G) > Δ_Λ(G) dann Λ_min := Λ
15 Gib Λ_min aus
```

Funktionsweise: Der Algorithmus testet sukzessive alle Belegungen der k Kanten aus E_k und ermittelt für jede Belegung mit den bekannten Algorithmen den minimalen maximalen gewichteten Ausgangsgrad für die verbleibenden Graphteile. Hierzu iteriert der Algorithmus zunächst (Zeile 3) über die Menge aller Orientierungen der Kanten E_k (Zeile 2). Dann wird eine Orientierung Λ gewählt und aus der Menge aller Orientierungen entfernt (Zeile 4). Entsprechend dieser Orientierung Λ wird auf G', einer Kopie des Graphen G (Zeile 5), für jeden Knoten ein Knotengewicht gesetzt (Zeile 6). Das bedeutet jeder Knoten bekommt das Gewicht 0 zugewiesen zuzüglich aller Kantengewichte von unter Λ ausgehenden Kanten aus E_k. Danach werden die Kanten E_k aus G' entfernt, da ihre Kantengewichte bereits in den Knotengewichten des Graphen berücksichtigt wurden (Zeile 7).

Durch das Löschen der Kanten E_k zerfällt der Graph G' in einzelne Zusammenhangskomponenten. Die Schleife in Zeile 8 iteriert über diese Komponenten, welche etwa per Tiefensuche identifiziert werden können. Für jede dieser Komponenten gilt nach Definition des Problems $\mathrm{MOO}_{d\leq 2}(k)$, dass jeder Knoten der jeweiligen Zusammenhangskomponente höchstens noch den Grad 2 hat. Folglich handelt es sich um zyklusförmige oder pfadförmige Teilgraphen. In den Zeilen 10 und 12 wird nun je nach Fall mit den bestehenden Algorithmen eine optimale Orientierung für die Komponente gefunden und der gewählten Orientierung Λ hinzugefügt (Zeile 13).

Ist dieser Schritt mit allen Zusammenhangskomponenten geschehen, so ist die Orientierung Λ eine vollständige Orientierung für den Graphen G. Sofern die gefundene Orientierung Λ einen geringeren maximalen gewichteten Ausgangsgrad hat, wird Λ_{min} auf dieses neue Λ gesetzt (Zeile 14). Der Algorithmus endet mit der Ausgabe von Λ_{min} (Zeile 15), wenn keine Orientierungen der Kanten E_k mehr zu testen sind.

Korrektheit: Wir arbeiten uns in dem Korrektheitsbeweis von innen nach außen: Es ist nach den Betrachtungen der Algorithmen dieses Abschnittes klar, dass der Aufruf der Algorithmen MOO_Zyklusorientierung(Z) beziehungsweise MOO_Pfadorientierung(Z) (Zeilen 10 und 12) für die Zusammenhangskomponenten jeweils optimale Orientierungen liefert. Ebenfalls ist klar, das sich Lösungen von verschiedenen Zusammenhangskomponenten nicht beeinflussen, die Einzellösungen also zu einer Gesamtlösung kombiniert werden können (Zeile 13).

Die Behauptung, dass durch das Löschen der Kanten E_k (Zeile 7) tatsächlich nur zyklusförmige und pfadförmige Zusammenhangskomponenten entstehen, folgt direkt aus Definition von $\text{MOO}_{d\leq 2}(k)$. Hier wird verlangt, dass für den Teilgraph $TG = (V, E_2, w)$ gilt, dass $grad(v) \leq 2 \quad \forall v \in V$. Durch das Löschen der Kanten E_k wird eben genau dieser Teilgraph erzeugt, der nun potentiell aus mehreren Zusammenhangskomponenten bestehen kann. Dass alle Knoten einer Zusammenhangskomponente höchstens den Grad 2 haben, gestattet nur zyklusförmige und pfadförmige Strukturen. Hierbei ist zu beachten, dass auch Knoten vom Grad 0 durch das Löschen der Kanten E_k entstehen können. Diese einzelnen Knoten sind jedoch nur Grenzfälle von pfadförmigen Graphen, werden also ebenfalls durch den Algorithmus MOO_Pfadorientierung(Z) abgedeckt.

Eine Orientierung einer Kante festzuhalten, ist äquivalent dazu ihr Kantengewicht demjenigen Knoten als Knotengewicht hinzuzufügen, aus dem die Kante ausgeht, und die Kante dann zu löschen. Daher kann die in Zeile 4 gewählte Orientierung der Kanten E_k, wie vom Algorithmus durchgeführt, durch das Erstellen von Knotengewichten und dem anschließenden Löschen der Kanten E_k realisiert werden (Zeilen 6 und 7). Die Erstellung der Knotengewichte muss für jede gewählte Orientierung der Kanten E_k erneut geschehen. Daher werden diese Operationen auf einer jeweiligen Kopie des Graphen durchgeführt, welche in Zeile 5 erstellt wurde. Aus der Untersuchung der Algorithmen 2, 3 und 4 wissen wir, dass diese Algorithmen korrekt mit Knotengewichten umgehen.

Die Argumentation bis hierhin hat gezeigt, dass für eine Orientierung der Kanten E_k eine dazu optimale Orientierung der restlichen Kanten gefunden wird. Die Korrektheit des gesamten Algorithmus folgt nun daraus, dass die äußere Schleife des Algorithmus (Zeile 3) alle möglichen Orientierungen der Kanten E_k systematisch durchprobiert. Es befindet sich also insbesondere auch eine optimale Orientierung OPT_k der Kanten E_k unter den von der äußerem Schleife durchprobierten Orientierungen. Im Inneren der Schleife wird nun, wie oben gezeigt, eine optimale vervollständigte Orientierung Λ für jede beliebige Orientierung der Kanten E_k konstruiert, also auch für OPT_k. Das Ergebnis Λ_{min} ist dann diese optimale vervollständigte Orientierung Λ zu OPT_k. Damit ist die Korrektheit gezeigt.

Laufzeit: Sei hierzu n die Größe der Eingabe in Bits.

Bei der Laufzeitbetrachtung des Algorithmus fällt ganz analog zu Algorithmus 1 zunächst auf, dass $\mathcal{O}$ wieder nicht explizit aufgestellt werden muss. Statt dessen ist es auch hier ausreichend systematisch alle $|\mathcal{O}| = 2^k$ Orientierungen zu testen, etwa über einen Suchbaum oder indem man für jede der k Kanten eine Richtung mit 0 und die andere mit 1 identifiziert und binär durchzählt. Die Laufzeit der äußeren Schleife (Zeile 3) ist wegen $|\mathcal{O}| = 2^k$ gerade $O(2^k) =: f(k)$. Betrachten wir im Folgenden die Laufzeit innerhalb dieser Schleife.

Die initialen Schritte in der Schleife, also das Wählen einer Orientierung, das Kopieren des Graphen, die Erstellung der Knotengewichte und auch das Entfernen der Kanten E_k (Zeilen 4 bis 7) lassen sich jeweils in $O(n)$ Zeit realisieren. Verbleibt noch der Aufwand der inneren Schleife aus Zeile 8 sowie die Überprüfung, ob eine bessere Orientierung gefunden wurde (Zeile 14). Letzteres ist ebenfalls in $O(n)$ Zeit realisierbar, indem für jeden Knoten der gewichtete Ausgangsgrad berechnet und der Maximalwert über die Knoten ermittelt wird. Verbleibt also die innere Schleife (Zeile 8).

Die innere Schleife (Zeile 8) muss zunächst die Zusammenhangskomponenten identifizieren. Dies ist in $O(n)$ Zeit realisierbar, indem aus dem noch zu untersuchenden Graphteil ein beliebiger Knoten gewählt wird und dann per Tiefensuche ermittelt wird, welche Knoten von ihm aus erreichbar sind. Da jeder Knoten nur in einer Zusammenhangskomponente liegen kann, wird durch diesen Schritt jeder Knoten nur einmal betrachtet, daher lässt sich das ganze Verfahren in $O(n)$ Zeit durchführen. Durch die Tiefensuche kann ebenfalls gleich herausgefunden werden, ob es sich um eine zyklusförmige Zusammenhangskomponente oder eine pfadförmige Zusammenhangskomponente handelt (Zeile 9). Der Aufruf der Algorithmen MOO_Zyklusorientierung(Z) bzw. MOO_Pfadorientierung(Z) benötigt ebenfalls nur lineare Zeit in der Größe der jeweiligen Zusammenhangskomponente. Da jeder Knoten aus G in nur genau einer Zusammenhangskomponente liegt, ergeben diese Aufrufe der jeweiligen Algorithmen eine kumulierte Gesamtzeit von $O(n)$. Letztlich lässt sich auch das Zusammenfügen der Orientierung Λ aus den Einzelorientierungen mit $O(n)$ abschätzen, da jede Kante nur einmal berücksichtigt wird.

Somit ergibt sich für die äußere Schleife (Zeile 3) ein Zeitbedarf von $O(n)$ für die initialen Schritte sowie nochmals $O(n)$ für die innere Schleife, in Summe als ein Gesamtzeitbedarf von $O(n)$. Gemeinsam mit den 2^k Wiederholungen der äußeren Schleife ergibt dies ein Gesamtzeitaufwand für den Algorithmus von $2^k \cdot O(n) = f(k) \cdot O(n)$ Zeit. Dies liegt in $\mathcal{FPT}$. Somit ist auch die Laufzeitbehauptung bewiesen.

Mit der gezeigten Laufzeit und der Korrektheit des Algorithmus haben wir gezeigt: $\mathrm{MOO}_{d\leq 2}(k) \in \mathcal{FPT}$.

□

Hinweis zum Algorithmus: Es ist möglich das Problem $\mathrm{MOO}_{d\leq 2}(k)$ auch derart zu erweitern, dass bereits im Ausgangsgraphen Knotengewichte zulässig sind. Die verwendeten Algorithmen 2, 3 und 4 sind bereits dafür ausgelegt mit

beliebigen Knotengewichten $\in \mathbb{N}_0$ umzugehen, lediglich der Algorithmus 5 sieht dies nicht vor. Tatsächlich würde es die Laufzeit jedoch nicht verändern, weitere Knotengewichte zu gestatten. Im Algorithmus 5 müsste hierzu lediglich Zeile 6 derart angepasst werden, dass die bestehenden Knotengewichte nicht überschrieben werden, sondern die aus Λ folgenden Knotengewichte den bestehenden Knotengewichten hinzu addiert werden.

Offene Fragestellung: $\mathrm{MOO}_{d\leq 2}(k)$ setzt voraus, dass die Kantenmenge E des Graphen aufgeteilt ist in $E_2 \cup E_k$ mit $|E_k| = k$. Weiter muss für den Teilgraphen $TG = (V, E_2, w)$ gelten, dass $grad(v) \leq 2 \ \ \forall v \in V$. Es stellt sich die Frage, wie diese Kantenmenge E_k in einem Graphen gefunden werden kann. Zudem ist die Laufzeit exponentiell in k, daher ist ein minimales k, also eine minimale Kantenmenge E_k von großem Interesse. Formal definiert sich das Optimierungsproblem wie folgt:

> Kantenmenge für Grad 2 (kurz KMG2):
> Gegeben: Ein einfacher Graph $G = (V, E)$.
> Gesucht: Eine Kantenmenge E_k minimaler Größe, sodass für den Teilgraph $TG = (V, E \setminus E_k)$ gilt, dass $\forall v \in V : grad(v) \leq 2$.

Dazu gehört folgendes Entscheidungsproblem:

> Kantenmenge für Grad 2 (k) (kurz KMG2(k)):
> Gegeben: Ein einfacher Graph $G = (V, E)$.
> Gefragt: Gibt es eine Kantenmenge E_k mit $|E_k| = k$, sodass für den Teilgraph $TG = (V, E \setminus E_k)$ gilt, dass $\forall v \in V : grad(v) \leq 2$?

Die Laufzeitkomplexität von KMG2(k) ist eine offene Frage. Leicht zu zeigen ist, dass KMG2(k) in $\mathcal{NP}$ liegt: Teste hierzu nichtdeterministisch jede mögliche Wahl von k Kanten aus E. Die Überprüfung ob der verbleibende Graph nach Entfernung der k Kanten die Bedingung $grad(v) \leq 2 \ \ \forall v \in V$ erfüllt, ist in polynomieller Zeit durchführbar.

Unbekannt ist, ob KMG2(k) und somit auch KMG2 in $\mathcal{P}$ liegt. Die Problemstellung wurde von mir eingehend untersucht, sowohl auf die Theorie hin, dass für KMG2 $\in \mathcal{P}$ gilt, als auch, dass KMG2 $\mathcal{NP}$-vollständig ist. Ergebnisse ließen sich jedoch nicht erzielen. Als mögliche Ansätze für KMG2 $\in \mathcal{P}$ wurden unter anderem Greedy-Algorithmen, die Anwendung des Ford-Fulkerson-Algorithmus für maximale Flüsse [14] und rekursive Lösungen mit dynamischer Programmierung untersucht. Siehe auch Schönings Buch „Algorithmik“ [27] für eine Beschreibung dieser Techniken. Eine $\mathcal{NP}$-Vollständigkeit wurde durch Reduktion der Probleme Hamiltonkreis, 3-SAT, Clique (Angewendet auf den Komplementgraphen) und weiterer Probleme versucht. Die genannten Probleme sind alle $\mathcal{NP}$-vollständig [18].

KMG2 lässt sich mit Faktor 2 approximieren: Es müssen für einen Knoten v mit $grad(v) > 2$ so viele Kanten entfernt werden, dass $grad(v) = 2$ gilt, dies sind $grad(v) - 2$ viele Kanten. Somit ist $s = \sum_{v \in V} \max\{0, grad(v) - 2\}$ eine obere Schranke für die Anzahl der zu entfernenden Kanten aus dem Graphen. Gleichzeitig reduziert das Entfernen einer Kante nur bei ihren zwei inzidenten Knoten den Grad um Eins. Im besten Fall haben diese beiden Knoten einen Grad größer als 2, sodass das Entfernen der Kante den maximalen Effekt hat. Somit ist $\lceil s/2 \rceil$ eine untere Schranke für die Anzahl der zu entfernenden Kanten. Die Approximierung mit Faktor 2 ist daher dadurch realisierbar, dass für jeden Knoten v mit $grad(v) > 2$ willkürlich $grad(v) - 2$ inzidente Kanten entfernt werden. Es gilt also KMG2 $\in \mathcal{APX}$.

Die obige Approximation lässt sich in der Praxis sogar verbessern, indem der Algorithmus nicht willkürlich die zu entfernenden Kanten wählt, sondern solche Kanten bevorzugt, bei denen beide inzidente Knoten einen Grad größer als 2 haben. Dieser Greedy-Ansatz führt jedoch nicht zwingend zu einer optimalen Lösung, wie Abbildung 4.15 zeigt.

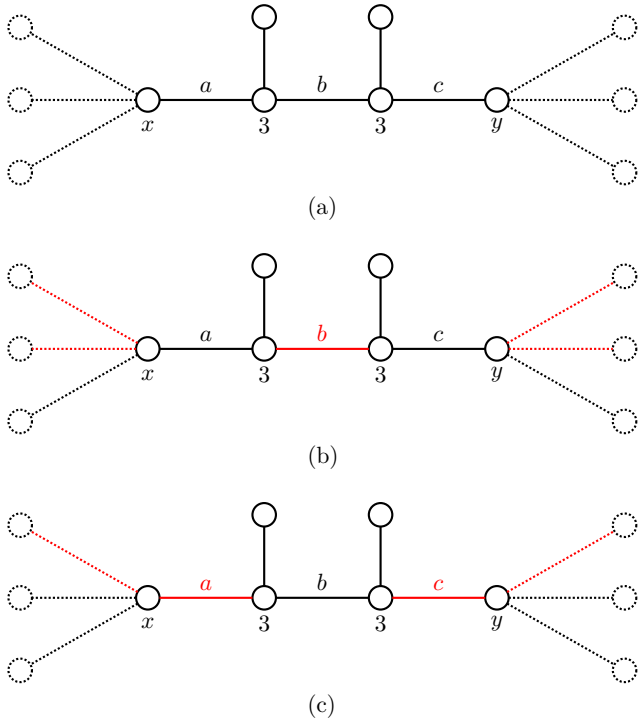


Abbildung 4.15: Gegenbeispiel zum Greedy-Ansatz für KMG2.
(a) Oben ist der Ausgangsgraph zu sehen. Die mittleren beiden Knoten haben den Grad 3 während die äußeren beiden Knoten den Grad x bzw. y haben mit $x, y > 2$. Die drei Kanten a, b und c erfüllen das Kriterium, dass jeweils beide inzidenten Knoten einen Grad größer als 2 haben. Somit kann der genannte Greedy-Algorithmus jede dieser drei Kanten wählen.
(b) In der Mitte wurde die Kante b gewählt. Dies sorgt dafür, dass an den äußeren Knoten noch $x-1$ und $y-2$ Kanten gewählt werden müssen. In Summe werden somit $x+y-3$ Kanten ausgewählt.
(c) Im unteren Bild wurde hingegen die Kante a gewählt. Die Kante b kann daraufhin nicht mehr bevorzugt gewählt werden, da der zweite Knoten von links nun nur noch den Grad 2 hat. Somit wird Kante c gewählt. Es werden an den äußeren Knoten $x-2$ und $y-2$ Kanten gewählt, inklusive a bzw. c. In Summe werden somit $x+y-4$ Kanten ausgewählt. Diese Lösung ist um eine Kante kleiner als die in der Mitte gezeigte Lösung. Der Greedy-Ansatz garantiert also keine optimalen Lösungen.

4.5 Diskussion der Ergebnisse

Das vorangegangene Kapitel hat sich mit dem Problem Minimum Outdegree Orientation beschäftigt, welches zum ersten Mal 2006 durch Asahiro et al. [2] definiert worden ist. Es gelang zu zeigen, dass Minimum Outdegree Orientation bereits mit sehr geringen Bedingungen (Knotengrad ≤ 3, nur zwei Kantengewichte) $\mathcal{NP}$-vollständig ist. Dies ist eine wesentliche Verbesserung des vormals bestehenden Ergebnisses von Asahiro et al. Dieser neue $\mathcal{NP}$-Vollständigkeitsbeweis zeigt insbesondere untere Grenzen für die Komplexität der betrachteten Graphen, sodass Minimum Outdegree Orientation noch $\mathcal{NP}$-vollständig ist:

- Verringert man die Anzahl der gestatteten verschiedenen Kantengewichte von zwei auf eines, so kann die Lösung mit Algorithmus 1 ($\mathrm{MOO}((G,k))$) in polynomieller Zeit gelöst werden. Alle Kanten haben dasselbe Gewicht c, folglich ist die Anzahl der andergewichteten Kanten $k = 0$ und der exponentielle Laufzeitanteil entfällt.
- Reduziert man den maximal erlaubten Knotengrad von ≤ 3 auf ≤ 2, so handelt es sich um ein in polynomieller Zeit lösbares Problem, welches mit Algorithmus 5 ($\mathrm{MOO}_{d\leq 2}((G,k))$) direkt gelöst werden kann. Hierzu wird $k = 0$ gesetzt, da nach Voraussetzung dann keine überzähligen Kanten vorliegen. Somit entfällt der exponentielle Laufzeitanteil des Algorithmus.

Darüber hinaus führt der $\mathcal{NP}$-Vollständigkeitsbeweis auch direkt zum Ergebnis $\mathrm{MOO} \notin \mathcal{PTAS}$.

Minimum Outdegree Orientation wurde in dieser Diplomarbeit erstmalig auf parametrisierte Komplexität untersucht. Die in den jeweiligen Abschnitten gezeigten Ergebnisse

- $\mathrm{MOO}(k) \in \mathcal{FPT}$ und
- $\mathrm{MOO}_{d\leq 2}(k) \in \mathcal{FPT}$

sind die konsequente Weiterführung der Erkenntnisse aus dem $\mathcal{NP}$-Vollständigkeitsbeweis. In beiden Fällen wurde der Parameter so gewählt, dass die betrachtete Variante von Minimum Outdegree Orientation durch das Entfernen der als Parameter genutzten Kanten ein in polynomieller Zeit lösbarer Spezialfall wurde. Diese Ergebnisse bilden somit eine Brücke zwischen den polynomiell lösbaren Spezialfällen und der generellen $\mathcal{NP}$-vollständigen Problemstellung und schließen die vormals vorhandene Lücke im Verständnis, woher das Problem Minimum Outdegree Orientation seine Schwierigkeit erhält.

Die Untersuchung von Minimum Outdegree Orientation ist mit diesen Ergebnissen nicht abgeschlossen. Weitere Parametrisierungen sind von Interesse. Ansatzpunkte hierfür kann insbesondere der in polynomieller Zeit lösbare Spezialfall von Minimum Outdegree Orientation auf Bäumen sein. Auf Bäumen ist Minimum Outdegree Orientation trivial lösbar [2]. Als zu untersuchende Parameter sollten daher die Anzahl oder die Struktur von Zyklen untersucht werden. Ebenfalls vielversprechend wären Ansätze über eine Baumzerlegung und die Wahl der Baumweite als Parameter. Durch den $\mathcal{NP}$-Vollständigkeitsbeweis ist allerdings naheliegend, dass der maximale Knotengrad kein geeigneter Parameter ist, da $\mathcal{NP}$-Vollständigkeit bereits ab Knotengrad ≤ 3 vorliegt.

Es sind außerdem weitere Untersuchungen zum Ergebnis $\mathrm{MOO}_{d\leq 2}(k) \in \mathcal{FPT}$ notwendig: Offen bleibt die Frage, in welcher Komplexitätsklasse das Problem Kantenmenge für Grad 2 liegt, welches am Ende der des letzten Abschnitts dargestellt wurde. Es fragt sich insbesondere, ob das Problem effizient lösbar ist, das heißt: Gilt KMG2 $\in \mathcal{P}$?

Sollte KMG2 $\notin \mathcal{P}$ gelten, so sollte sich die $\mathcal{NP}$-Vollständigkeit zeigen lassen. Es stellen sich in diesem Fall weitere Fragen: Wie schwierig ist KMG2 lösbar? Hierbei wären Untersuchungen auf parametrisierte Komplexität von Interesse. Auch sollte im Fall KMG2 $\notin \mathcal{P}$ in die Approximierbarkeit genauer betrachtet werden. Das bisherige Approximationsergebnis lässt offen, ob etwa KMG2 $\in \mathcal{PTAS}$ gilt.

Wegen des Satzes 2.8 hat Minimum Outdegree Orientation bezüglich beider gezeigten Parametrisierungen einen Problemkern. Allerdings ist dieser Problemkern nicht notwendigerweise optimal, wie im Beweis zu Satz 2.8 zu sehen ist. Eine weitere offene Fragestellung ist, ob es Parametrisierungen gibt, die noch bessere Problemkerne von Minimum Outdegree Orientation ermöglichen.

5 Zusammenfassung und Ausblick

In der vorliegenden Diplomarbeit wurde ein Einblick in das Themengebiet der Fixed Parameter Tractability ($\mathcal{FPT}$) gegeben und gezeigt, wie Techniken hieraus dazu verwendet werden können um $\mathcal{NP}$-vollständige Probleme für die Praxis handhabbar zu machen. Näher betrachtet haben wir insbesondere Techniken zur Problemkern-Reduktion. Im Kapitel 2 haben wir hierzu Grundlagen kennen gelernt. Das Konzept der Problemkern-Reduktion und ihr Zusammenhang zur Komplexitätsklasse $\mathcal{FPT}$ wurden genauer beleuchtet. Auf Basis dieser Grundlagen haben wir in Kapitel 3 verschiedene dieser Techniken zur Problemkern-Reduktion am $\mathcal{NP}$-vollständigen Graphenproblem Vertex Cover kennen gelernt und sie miteinander verglichen. Wir haben hierdurch an einem in der Praxis relevanten Problem erkennen können, wie die Techniken der Problemkern-Reduktion erhebliche Verbesserungen in der Handhabbarkeit dieses schwierigen Problems ermöglichen.

Darüber hinaus hat diese Diplomarbeit in Kapitel 4 das Problem Minimum Outdegree Orientation detailliert untersucht. Hierbei sind drei neue Ergebnisse entstanden:

- Ein verbesserter $\mathcal{NP}$-Vollständigkeitsbeweis welcher nachweist, dass Minimum Outdegree Orientation bereits für Knotengrad ≤ 3 und lediglich zwei Kantengewichte nicht mehr effizient lösbar ist.
- Es konnte gezeigt werden, dass Minimum Outdegree Orientation mit dem Parameter „k andersgewichtete Kanten“ in $\mathcal{FPT}$ liegt.

- Es wurde nachgewiesen, dass Minimum Outdegree Orientation auch mit dem Parameter „k überzählige Kanten“ in $\mathcal{FPT}$ liegt.

Diese Ergebnisse tragen zu einem verbesserten Verständnis des Problems bei. Durch die erstmalige Parametrisierung und den Beweis, dass diese parametrisierten Varianten von Minimum Outdegree Orientation in $\mathcal{FPT}$ liegen, sind für den praktischen Umgang mit diesem Graphenproblem erhebliche Fortschritte erzielt. Insbesondere ist durch den Satz 2.8 zudem gezeigt, dass Minimum Outdegree Orientation einen Problemkern besitzt.

Durch die intensive Betrachtung des Problems Minimum Outdegree Orientation sind zahlreiche weitere Fragestellungen entstanden, deren Untersuchung sehr lohnenswert erscheint. Die Diskussion im vorhergehenden Kapitel bietet hierfür eine gute Grundlage. Wichtige Ansatzpunkte sind hierbei die Untersuchung weiterer Parametrisierungen von Minimum Outdegree Orientation aber auch dessen Komplexität auf speziellen Graphenklassen. Die Komplexität des Problems Kantenmenge für Grad 2 ist eine weitere offene Frage, deren Beantwortung einen wichtigen Beitrag für die Parametrisierungen von Minimum Outdegree Orientation leisten kann.

Das vergleichsweise jungen Gebiet der parametrisierten Komplexitätstheorie wird in Zukunft sicherlich eine wichtige Rolle spielen. Wie schon in der Einleitung erwähnt sind viele Problemstellungen, gerade etwa $\mathcal{NP}$-vollständige Graphenprobleme, aus praktischen Bedürfnissen von Gebieten wie der Bioinformatik entstanden. Die parametrisierte Komplexitätstheorie schafft es an dieser Stelle das Bindeglied zwischen Theorie und algorithmischer Praxis zu sein und einen Beitrag dazu zu leisten, den hohen Bedarf an exakten und möglichst schnellen Algorithmen für schwierige Probleme zu befriedigen.

Abbildungsverzeichnis

Liste der Algorithmen

Literaturverzeichnis

[1] Abu-Khzam, Faisal N. ; Collins, Rebecca L. ; Fellows, Michael R. ; Langston, Michael A. ; Suters, Michael A. ; Symons, Chris T.: Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. In: *Proceedings ALENEX/ANALC 2004, Springer-Verlag, Lecture Notes in Computer Science (2004)* (2004), S. 62–69

[2] Asahiro, Yuichi ; Miyano, Eiji ; Ono, Hirotaka ; Zenmyo, Kouhei: Graph orientation algorithms to minimize the maximum outdegree. In: *Proceedings of the 12th Computing: The Australasian Theroy Symposium - Volume 51*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2006 (CATS '06). – ISBN 1–920682–33–3, 11–20

[3] Bader, Martin: *Basierend auf einer Idee aus einem persönlichen Gespräch*

[4] Bodlaender, Hans L.: Kernelization: New Upper and Lower Bound Techniques. (2009), S. 17–37. ISBN 978–3–642–11268–3

[5] Bodlaender, Hans L. ; Fomin, Fedor V. ; Lokshtanov, Daniel ; Penninkx, Eelko ; Saurabh, Saket ; Thilikos, Dimitrios M.: (Meta) Kernelization. In: *CoRR* abs/0904.0727 (2009)

[6] Buss, Jonathan ; Goldsmith, Judy: Nondeterminism within P. Version: 1991. `http://dx.doi.org/10.1007/BFb0020811`. In: Choffrut, Christian (Hrsg.) ; Jantzen, Matthias (Hrsg.): *STACS 91* Bd. 480. Springer Berlin / Heidelberg, 1991, 348-359. – 10.1007/BFb0020811

[7] Chen, Jianer ; Kanj, Iyad ; Jia, Weijia: Vertex Cover: Further Observations and Further Improvements. Version: 1999. `http://dx.doi.org/10.`

1007/3-540-46784-X_30. In: WIDMAYER, Peter (Hrsg.) ; NEYER, Gabriele (Hrsg.) ; EIDENBENZ, Stephan (Hrsg.): *Graph-Theoretic Concepts in Computer Science* Bd. 1665. Springer Berlin / Heidelberg, 1999, 313-324

[8] CHOR, Benny ; FELLOWS, Mike ; JUEDES, David: Linear Kernels in Linear Time, or How to Save k Colors in $O(n^2)$ Steps. Version: 2005. http://dx.doi.org/10.1007/978-3-540-30559-0_22. In: HROMKOVIC, Juraj (Hrsg.) ; NAGL, Manfred (Hrsg.) ; WESTFECHTEL, Bernhard (Hrsg.): *Graph-Theoretic Concepts in Computer Science* Bd. 3353. Springer Berlin / Heidelberg, 2005, 257-269

[9] DIESTEL, Reinhard: *Graph Theory (Graduate Texts in Mathematics)*. 3rd. Springer, 2006 http://www.worldcat.org/isbn/3540261834. – ISBN 3540261834

[10] DINIC, E. A.: Algorithm for solution of a problem of maximum flow in a network with power estimaton. In: *Soviet Math. Dokl. Vol. 11 No. 5.* 1970

[11] DOWNEY, Rod G. ; FELLOWS, M. R.: *Parameterized Complexity.* Springer, 1999 http://www.worldcat.org/isbn/038794883X. – ISBN 038794883X

[12] DOWNEY, Rod G. ; FELLOWS, Michael R.: Fixed-Parameter Tractability and Completeness I: Basic Results. In: *SIAM J. Comput.* 24 (1995), Nr. 4, S. 873–921. http://dx.doi.org/http://dx.doi.org/10.1137/S0097539792228228. – DOI http://dx.doi.org/10.1137/S0097539792228228. – ISSN 0097–5397

[13] FERNAU, Henning ; NIEDERMEIER, Rolf: An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover. 1672 (1999), 387-397. http://dx.doi.org/10.1007/3-540-48340-3_35

[14] FORD, L. R. ; FULKERSON, D. R.: *Flows in Networks.* Princeton University Press, 1962

[15] GAREY, Michael R. ; JOHNSON, David S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathe-*

matical Sciences). W. H. Freeman & Co Ltd, 1979 http://www.worldcat.org/isbn/0716710455. – ISBN 0716710455

[16] Guo, Jiong ; Niedermeier, Rolf: Invitation to data reduction and problem kernelization. In: *SIGACT News* 38 (2007), Nr. 1, S. 31–45. http://dx.doi.org/http://doi.acm.org/10.1145/1233481.1233493. – DOI http://doi.acm.org/10.1145/1233481.1233493. – ISSN 0163–5700

[17] Jörg Flum, Martin G.: *Parameterized Complexity Theory*. Springer Berlin Heidelberg, 2006

[18] Karp, R. M.: Reducibility Among Combinatorial Problems. In: Miller, R. E. (Hrsg.) ; Thatcher, J. W. (Hrsg.): *Complexity of Computer Computations*. Plenum Press, 1972, S. 85–103

[19] Khot, Subhash ; Regev, Oded: Vertex cover might be hard to approximate to within 2-[epsilon]. In: *Journal of Computer and System Sciences* 74 (2008), Nr. 3, 335 - 349. http://dx.doi.org/DOI:10.1016/j.jcss.2007.06.019. – DOI DOI: 10.1016/j.jcss.2007.06.019. – ISSN 0022–0000. – Computational Complexity 2003

[20] Krüger, Dominikus: *Multivariate Algorithmik zur Orientierung von Protein-Protein-Interaktionsnetzwerken (Masterarbeit an der Universität Tübingen)*. 2010

[21] Lovász, M.D. László; P. László; Plummer: *Matching Theory*. North-Holland, 1986

[22] Nemhauser, G. L. ; Trotter, L. E.: Vertex packings: Structural properties and algorithms. In: *Mathematical Programming* 8 (1975), 232-248. http://dx.doi.org/10.1007/BF01580444. – ISSN 0025–5610. – 10.1007/BF01580444

[23] Niedermeier, Rolf: *Invitation to Fixed-Parameter Algorithms (Habilitation Thesis)*. Oktober 2002

[24] Niedermeier, Rolf: *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, 2006 http://www.worldcat.org/isbn/0198566077. – ISBN 0198566077

[25] O'Rourke, Joseph: *Art gallery theorems and algorithms*. New York, NY, USA : Oxford University Press, Inc., 1987. – ISBN 0–19–503965–3

[26] Robertson, Neil ; Sanders, Daniel P. ; Seymour, Paul ; Thomas, Robin: Efficiently four-coloring planar graphs. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 1996 (STOC '96). – ISBN 0–89791–785–5, 571–575

[27] Schöning, Uwe: *Algorithmik*. Spektrum Akademischer Verlag, 2001

[28] Wagner, K.: Über eine Eigenschaft der ebenen Komplexe. In: *Mathematische Annalen* 114 (1937), 570-590. http://dx.doi.org/10.1007/BF01594196. – ISSN 0025–5831. – 10.1007/BF01594196

Index

Name: Marcus Bombe Matrikelnummer: 617885

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 28. Februar 2011 .....

Marcus Bombe