Institut für theoretische Informatik

Leibniz Universität Hannover

Bachelorarbeit

Graph Modification Problems

von Daniel Wiebking
Matrikel-Nr.: 2941990

9. Februar 2015

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

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe und keine anderen Hilfsmittel und Quellen als angegeben verwendet habe.

Daniel Wiebking

Inhaltsverzeichnis

1 Einleitung

In der theoretischen Informatik geht es vor allem darum Probleme algorithmisch zu lösen. Wir stellen uns beispielsweise das Problem vor, dass wir eine Landkarte vorliegen haben, die wir gerne farbig darstellen würden. Dazu möchten wir jedes Land mit einer Farbe einfärben. Außerdem möchten wir je zwei angrenzende Länder mit unterschiedlicher Farbe einfärben. Leider werden uns nur drei verschiedene Farben zur Verfügung gestellt, die wir benutzen dürfen. Dieses Problem wird auch 3-Färbbarkeitsproblem genannt. Es kann für ein Problem verschiedene Lösungsansätze und daher verschiedene Algorithmen für die jeweilige Berechnung geben. Einige von ihnen sind jedoch schneller als Andere. Mit der Effizienz von Algorithmen beschäftigt man sich in der theoretischen Informatik. Doch nicht jedes Problem lässt sich effizient lösen. Es gibt Probleme, die selbst auf modernsten Computern Jahre bräuchten, um gelöst zu werden. Bei einigen dieser Probleme liegt es nicht an der Tatsache, dass bisher kein effizienter Algorithmus gefunden wurde, sondern vielmehr konnte bewiesen werden, dass es keinen effizienten Algorithmus geben kann. Für andere dieser Probleme dagegen ist noch nicht bekannt, ob es einen effizienten Algorithmus gibt. Doch was bedeutet überhaupt Effizienz? Kehren wir zu unserem 3-Färbbarkeitsproblem zurück. Es ist ersichtlich, dass ein Algorithmus, der die Deutschlandkarte in 3 Farben färben soll, weniger Zeit beansprucht, als für die 3-Färbung von Europa. Aus diesem Grund hat der selbe Algorithmus verschiedene Laufzeiten, sodass es Sinn macht die Laufzeit eines Algorithmus als Funktion anzusehen. Die Funktion liefert, abhängig von der Landkarte, die wir dem Algorithmus zur Verfügung stellen, einen anderen Wert. Es spielt jedoch nicht die Landkarte selbst, sondern die Komplexität der Landkarte eine Rolle. Ein geeignetes Maß für die Komplexität stellt hierbei die Größe der Landkarte dar. Hier wird klar, warum die Komplexität der Europakarte größer als die der Deutschlandkarte ist.

Für die Laufzeit eines Algorithmus betrachten wir die Anzahl der elementaren Schritte, da die tatsächliche Zeit in Minuten von Hardware zu Hardware variieren kann und daher keine Eigenschaft des Algorithmus selbst ist. Eine Laufzeitfunktion f ist also eine Funktion von $\mathbb{N}$ nach $\mathbb{N}$. Um zwei Laufzeitfunktionen f und g miteinander zu vergleichen macht es wenig Sinn einzelne Werte zu vergleichen, vielmehr ist das Wachstumsverhalten der Funktion von Interesse. Eine logarithmisch wachsende Funktion wird als schwächer, als eine quadratisch wachsende Funktion angesehen, da erst genannte auf Dauer dominieren wird. Mit diesem Hintergedanken wurde 1894 die O-Notation von Paul Bachmann eingeführt. $O(g(n))$ ist die Menge aller Funktionen, die höchstens so stark wächst wie $g(n)$. Es hat sich gezeigt, dass Algorithmen mit polynomiell wachsender Laufzeitfunktion in der Praxis meist effizient genug sind. Des Weiteren hat es sich als ausreichend erwiesen, sich nur mit einer bestimmten Art von Problemen zu beschäftigen – den Entscheidungsproblemen. Ein Entscheidungsproblem ist ein solches Problem, bei dem die Ausgabe bzw. die Antwort nur mit „Ja“ oder „Nein“ beantwortet werden kann. Einen Algorithmus, der ein Entscheidungsproblem löst, nennen wir Entscheidungsalgorithmus. Das zum 3-Färbbarkeitsproblem analoge Entscheidungsproblem wäre das 3-Färbbarkeitsentscheidungsproblem. Bei diesem Entscheidungsproblem ist nicht nach einer gültigen 3-Färbung gefragt, es soll lediglich beantwortet werden, ob eine gegebene Landkarte eine 3-Färbung besitzt oder nicht.

Wir können Algorithmen hinsichtlich ihrer Laufzeiten klassifizieren. Außerdem können wir Entscheidungsprobleme nach den Laufzeiten der dazu gehörigen Entscheidungsalgorithmen klassifizieren. Die Klasse der Entscheidungsprobleme, die wir effizient lösen können, wird mit P bezeichnet. Falls wir effizient ermitteln können, ob die Deutschlandkarte 3-färbbar ist, so ist das 3-Färbbarkeitsentscheidungsproblem effizient lösbar und gehört somit zur Klasse P. Außerdem gibt es noch die Klasse der Entscheidungsprobleme, die wir effizient überprüfen können, welche mit NP bezeichnet wird. Was eine effiziente Überprüfung bedeutet, kann am Beispiel des 3-Färbbarkeitsentscheidungsproblems veranschaulicht werden. Nehmen wir an, wir kennen eine Färbung der Deutschlandkarte. Falls effizient überprüft werden kann, ob unsere vorgeschlagene Färbung tatsächlich eine gültige 3-Färbung ist, so ist das 3-Färbbarkeitsentscheidungsproblem effizient überprüfbar. Das Überprüfen einer Lösung ist nie schwieriger als das Lösen des Problems und somit ist P eine Teilklasse von NP. Also gilt $P \subseteq NP$. Die Frage ist, ob auch die Rückrichtung gilt. Also, ob jedes effizient überprüfbare Problem auch noch hinreichend effizient lösbar ist und somit die Gleichheit der Klassen P und NP gilt. Diese Frage konnte bis heute nicht beantwortet werden. Um die Gleichheit zu beweisen, müsste für jedes Problem aus NP gezeigt werden, dass es sich effizient lösen lässt. Dies erweist sich als sehr schwierig, da die Klasse NP unendlich viele Probleme beinhaltet. Es wurde jedoch ein Problem gefunden, welches sich in gewisser Weise als eines der schwierigsten in NP herausgestellt hat. Dabei handelt es sich um das Erfüllbarkeitsproblem aussagenlogischer Formeln, welches mit SAT bezeichnet wird. SAT ist gerade aus dem Grund eines der schwierigsten Probleme, weil 1971 von Stephen A. Cook gezeigt wurde, dass ein beliebiges Problem aus NP gelöst werden kann, indem wir einen leicht modifizierten Algorithmus für SAT auf das Problem anwenden. SAT wird daher auch als NP-vollständig bezeichnet. Falls also ein effizienter Algorithmus für SAT gefunden wird, wäre auch jedes beliebige Problem aus NP effizient lösbar. Dies hätte natürlich auch $P = NP$ zur Folge. Doch es konnte weder bewiesen, noch widerlegt werden, dass SAT effizient lösbar ist. Analog zu SAT konnten weitere schwierigste – also NP-vollständige – Probleme gefunden werden, mit deren Hilfe man beliebe Probleme aus NP lösen kann. Doch auch für diese wurde bislang weder ein effizienter Algorithmus gefunden, noch konnte widerlegt werden, dass es keinen effizienten Algorithmus geben kann. Um diese NP-vollständigen Probleme besser zu verstehen, wurde das Gebiet der Parametrisierten Komplexität ergründet. Hier wird untersucht an welchen Parametern die Schwierigkeit der NP-vollständigen Probleme begründet ist. Am Beispiel des 3-Färbbarkeitsentscheidungsproblems, welches auch NP-vollständig ist, kann gefragt werden welche Faktoren das 3-Färben einer Landkarte so schwer machen. Hierbei kann die Auswirkung der Anzahl der Länder oder die der Anzahl der Ländergrenzen auf die Laufzeit genauer untersucht werden. Ein möglicher Faktor kann auch die maximale Anzahl an angrenzenden Ländern sein.

Um außerdem das 3-Färbbarkeitsproblem formal zu beschreiben, wird eine möglichst einfache Darstellung von Landkarten verwendet. Landkarten besitzen meist viele redundante Informationen, die zur Entscheidung der 3-Färbbarkeit nicht weiter helfen. Es ist beispielsweise unerheblich wie groß die jeweiligen Länder sind oder welche geometrische Form die Länder aufweisen. Die einzig relevante Information ist, welche Länder jeweils eine Grenze zueinander haben und welche nicht. Diese Information kann in einer Relation gespeichert werden, und somit lässt sich das 3-Färbbarkeitsproblem als so genanntes Graphenproblem auffassen. Auch andere NP-vollständige Probleme können mithilfe von Graphen beschrieben werden, weshalb die Graphentheorie eine gewisse Rolle in der theoretischen Informatik spielt.

Aus diesem Grund werden wir uns in meiner Arbeit mit der NP-Vollständigkeit des Graphen-Modifikations-Problems beschäftigen und untersuchen, welche Faktoren dieses Problem so schwer machen. Das Graph-Modifikations-Problem besteht darin, einen Eingabegraphen so zu modifizieren, sodass er einer vorgegebenen Gruppe von Zielgraphen angehört. Wir werden insbesondere das Cluster-Graphen-Modifikations-Problem untersuchen, bei dem die Gruppe der Zielgraphen verschiedene Klassen von Cluster-Graphen seien werden. Ein Cluster-Graph ist ein Graph, bei dem jede Zusammenhangskomponente eine Clique bildet. Seine Kantenrelation ist daher transitiv. Da wir uns ausschließlich mit einfachen Graphen beschäftigen werden, sind Kantenrelationen stets symmetrisch und obwohl stets von irreflexiven Relationen ausgegangen wird, können die selben Betrachtungen analog mit reflexiven Kantenrelationen gemacht werden. Aus diesem Grund kann das Cluster-Graph-Modifikations-Problem auch so aufgefasst werden, als würde es darin bestehen aus einer gegebenen reflexiven und symmetrischen Eingaberelation durch möglichst wenig Veränderungen eine Äquivalenzrelation zu erhalten. Äquivalenzrelationen spielen auch in der Praxis eine Rolle. Zum Beispiel kann man sich einen Algorithmus vorstellen, der entscheidet, ob sich zwei Objekte auf auf gewisser Weise ähnlich sind. Die dadurch entstandene Relation sollte einer Äquivalenzrelation entsprechen. Doch meist wird ein solcher Algorithmus nicht zuverlässig arbeiten, da die Ähnlichkeit von Objekten häufig nicht formal definiert werden kann. Die berechnete Relation wird jedoch sicherlich zumindest reflexiv und symmetrisch sein. An dieser Stelle kann die Cluster-Graph-Modifizierung zum Einsatz kommen und aus der vorher berechneten Relation eine Äquivalenzrelation berechnen, die sich möglichst wenig von der Ausgangsrelation unterscheidet.

2 Grundlagen

2.1 Parametrisierte Komplexität

Die Definitionen aus diesem Abschnitt stammen aus [FG06].

Definition 2.1. *Sei Σ ein endliches Alphabet. Eine Parametrisierung über Σ^* ist eine in Polynomialzeit berechenbare Funktion $\kappa : \Sigma^* \to \mathbb{N}$.*

Definition 2.2. *Sei Σ ein endliches Alphabet. Ein parametrisiertes Problem über Σ ist ein Paar (Q, κ), wobei Q eine Sprache über Σ und κ eine Parametrisierung über Σ ist.*

Wir notieren das Entscheidungsproblem einer Sprache Q wie folgt.

Problem Q	
Eingabe	$x \in \Sigma^*$
Frage	liegt x in Q

Das parametrisierte Problem notieren wir wie folgt.

Problem κ-Q	
Eingabe	$x \in \Sigma^*$
Parameter	$\kappa(x)$
Frage	liegt x in Q

Wir betrachten nun ein Beispiel.

Beispiel 2.3 Wir wählen $Q =$ Clique. Wir werden unsere Eingaben x im folgenden derart einschränken, dass sie einer gültigen Kodierung des jeweiligen Problems entsprechen. Für Clique betrachten wir daher nur diejenigen Eingaben über dem Alphabet Σ, welche einer gültigen Kodierung eines Graphen G und einem $k \in \mathbb{N}$ entsprechen. Wir notieren das Cliquenproblem daher wie folgt.

Problem Clique	
Eingabe	Ein Graph G und $k \in \mathbb{N}$
Frage	Existiert in G eine Clique der Größe k

Wir können auch eine mögliche parametrisierte Variante des Problems formulieren. Hierzu ist $\kappa(x) := \begin{cases} k, & \text{falls } x \text{ eine gültige Kodierung der Form } \langle G, k \rangle \text{ ist} \\ 1, & \text{sonst} \end{cases}$.
Da unsere Eingaben x stets einer gültigen Kodierung entsprechen, wird für $\kappa(x)$ nie der sonst-Fall eintreten. Um zwischen parametrisierten und klassischen Problemen zu unterscheiden, verwenden wir im Fall von parametrisierten Problemen den Parameter $\kappa(x)$ als Präfix. Somit notieren wir unsere parametrisierte Variante wie folgt.

Problem	k-Clique
Eingabe	Ein Graph G und $k \in \mathbb{N}$
Parameter	k
Frage	Existiert in G eine Clique der Größe k

Bezüglich der Ein- und Ausgabe unterscheiden sich parametrisierte Probleme nicht von klassischen und folglich lassen sich die selben Algorithmen für die jeweiligen Probleme verwenden. Der Unterschied liegt jedoch in der Laufzeitbetrachtung. Klassisch können wir die Laufzeit von Algorithmen nur mit der Länge der Eingabe $|x|$ abschätzen. In der parametrisierten Welt ist es zusätzlich möglich die Laufzeit durch den Parameter abzuschätzen. Die Laufzeitfunktion ist daher eine Funktion in Abhängigkeit der Größen $|x|$ und $\kappa(x)$. Nun können wir einige Komplexitätsklassen definieren.

Definition 2.4. FPT *ist die Menge der parametrisierten Probleme* (Q, κ)*, für die es einen Entscheidungsalgorithmus gibt, der für eine berechenbare Funktion* $f : \mathbb{N} \to \mathbb{N}$ *in* $O(f(\kappa(x)) \cdot |x|^{O(1)})$ *arbeitet.*

Die Funktion f kann hierbei beliebig stark wachsen. Mit der o-Notation erhalten wir die folgende Verschärfung.

Definition 2.5. SUBEPT *ist die Menge der parametrisierten Probleme* (Q, κ)*, für die es einen Entscheidungsalgorithmus gibt, der in* $O(2^{o(\kappa(x))} \cdot |x|^{O(1)})$ *arbeitet.*

Für parametrisierte Probleme, für die Q bereits in P liegt erhalten wir eine weitere Klasse.

Definition 2.6. PT *ist die Menge der parametrisierten Probleme* (Q, κ)*, für die es einen Entscheidungsalgorithmus gibt, der in* $O(|x|^{O(1)})$ *arbeitet.*

Außerdem definieren wir eine Oberklasse von FPT.

Definition 2.7. XP *ist die Menge der parametrisierten Probleme* (Q, κ)*, für die es einen Entscheidungsalgorithmus gibt, der für eine berechenbare Funktion* $f : \mathbb{N} \to \mathbb{N}$ *in* $O(f(\kappa(x)) \cdot |x|^{f(\kappa(x))})$ *arbeitet.*

Es gilt die Hierarchie PT $\subseteq$ SUBEPT $\subseteq$ FPT $\subseteq$ XP.

2.2 Graph-Modifizierung

Ein einfacher Graph G ist ein 2-Tupel $(V[G], E[G])$, wobei $V[G]$ eine nicht leere Menge von Knoten und $E[G]$ eine symmetrische, irreflexive Relation auf $V[G]$ ist. $E[G]$ nennen wir die Menge der Kanten. Wenn klar ist auf welchen Graphen wir uns beziehen, notieren wir im folgenden die Knoten mit V und die Kanten mit E. Für eine irreflexive und symmetrische Relation gilt für alle $u, v \in V$, dass $(u, u) \notin E$ und dass $(u, v) \in E$ genau dann, wenn $(v, u) \in E$. Aus diesem Grund macht es Sinn die Relation E nicht durch eine Menge von Tupeln, sondern durch eine Menge von 2-elementigen Mengen zu beschreiben. Wir definieren $A \circ B := \{\{u, v\} \mid u \in A, v \in B, u \neq v\}$. Anstelle von $A \circ A$ schreiben wir im folgenden A^2. Das kartesische Produkt hingegen schreiben wir stets vollständig mit $A \times A$ aus. Es ist $E \subseteq V^2$. Für den Fall, dass wir die Kanten betrachten wollen, welche nicht in E enthalten sind, definieren wir $\bar{E} := V^2 \setminus E$. Um diese Definition auf Graphen zu erweitern, definieren wir $\bar{G} := (V, \bar{E})$. Mit $E(A, B)$

bezeichnen wir die Menge sämtlicher vorhandener Kanten zwischen Knoten aus A und B. Somit gilt $E(A, B) = E \cap (A \circ B)$.

Beispiel 2.8 $G =$  $A =$  $B =$ $\Rightarrow E[G](A, B) =$ 

Auch hier können wir analog die nicht vorhandenen Kanten zwischen A und B betrachten und definieren hierfür $\bar{E}(A, B) := \bar{E} \cap (A \circ B)$. Wir können eine Knotenmenge $V' \subseteq V$ aus einem Graphen $G = (V, E)$ entfernen und definieren hierzu $G - V' := (V \setminus V', E(V \setminus V', V \setminus V'))$. Wir nennen einen Graphen H Teilgraph vom Graphen G, falls eine Knotenmenge V' existiert, für die die Gleichung $G - V' = H$ erfüllt ist, und schreiben in einem solchen Fall auch $H \subseteq G$.

Beispiel 2.9 $G =$  $V' =$  $\Rightarrow G - V' =$  $\subseteq G$

Für einzelne Knoten v definieren wir analog $G - v := G - \{v\}$. Kanten hingegen können in einem Graphen G mithilfe einer Modifikationsmenge $F \subseteq V^2$ entfernt oder hinzugefügt werden. Der modifizierte Graph ist $G_F := (V, (E \setminus F) \cup (F \setminus E))$.

Beispiel 2.10 $G =$  $F =$  $\Rightarrow G_F =$ 

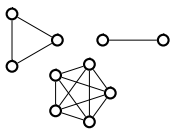
Ein Graph-Modifikations-Problem ist wie folgt definiert.

Problem Π-Graph Modification (Π-GM)

Eingabe	Ein Graph $G = (V, E)$ und $i, j, k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq V^2$ mit $\lvert F \cap E \rvert \leq i$, $\lvert F \cap \bar{E} \rvert \leq j$ und $\lvert F \rvert \leq k$, sodass $G_F \in \Pi$

Es geht also darum einen Eingabegraphen G derart zu modifizieren, dass er einer Menge von Graphen Π angehört. Dabei dürfen maximal i Kanten entfernt, maximal j Kanten hinzugefügt und maximal k Kanten modifiziert werden. Es gibt eine Vielzahl von Graphen-Modifikations-Problemen, da die Menge der Zielgraphen Π beliebig gewählt sein kann. Solange wir $G_F \in \Pi$ in Polynomialzeit überprüfen können, liegt das Graph-Modifikations-Problem in NP. Dies wird bei uns jedoch stets der Fall sein, weshalb wir die NP-Mitgliedschaft unserer NP-harten Probleme stillschweigend voraussetzen werden. Wir konzentrieren uns speziell auf einen bestimmten Zielgraphen.

Definition 2.11. *Ein Graph heißt $^{p\leq}/^{p=}/^{p\geq}$Cluster-Graph, falls dieser aus mindestens/genau/maximal p Zusammenhangskomponenten besteht, die jeweils eine Clique bilden.*

Beispiel 2.12 Der Graph $G =$  ist sowohl ein $^{2\leq}$Cluster-Graph als auch ein $^{3=}$Cluster-Graph, sowie ein $^{6\geq}$Cluster-Graph.

Falls es sich bei Π um die Menge der $^{p\leq}/^{p=}/^{p\geq}$Cluster-Graphen handelt, so nennen wir das dazugehörige Graphen-Modifikations-Problem auch $^{p\leq}/^{p=}/^{p\geq}$Cluster-Graph-Modifikations-Problem. Des Weiteren können wir Scheiben des Problems betrachten, indem wir zusätzlich $i = j = k$ fordern. Nun bleibt lediglich die Bedingung $\lvert F \rvert \leq k$ und wir erhalten die Sprachen $^{p\leq}/^{p=}/^{p\geq}$Cluster Editing. Falls wir jedoch $j = 0$ und

$i = k$ fordern, so erhalten wir die Bedingungen $F \subseteq E$ und $|F| \leq k$. Somit können aus dem Eingabegraphen nur Kanten gelöscht werden. Auf diese Weise erhalten wir weitere Sprachen, welche wir mit ${}^{p\leq}/{}^{p=}/{}^{p\geq}$Cluster Deletion bezeichnen wollen. Außerdem können wir $i = 0$ und $j = k$ fordern und erhalten die Bedingungen $F \subseteq \bar{E}$ und $|F| \leq k$. In diesem Fall ist nur das Hinzufügen von Kanten erlaubt und wir erhalten weitere Sprachen, welche wir mit ${}^{p\leq}/{}^{p=}/{}^{p\geq}$Cluster Completion bezeichnen wollen. Sollte p ein Teil der Eingabe sein, so werden wir p im Sprachennamen weglassen. Eine für diesen Fall exemplarische Sprache wäre ${}^{\geq}$Cluster Editing, bei der wir abhängig von der Eingabe $\langle G, k, p\rangle$ eine andere Menge von Zielgraphen erhalten. Da Π somit von der Eingabe abhängt, ist ${}^{\geq}$Cluster Editing keine Scheibe des Graphen-Modifikations-Problems. Es wird jedoch trotzdem in dieser Arbeit untersucht.

3 Cluster Editing

3.1 $^{p\leq}$Cluster Editing

Als erstes beschäftigen wir uns mit $^{p\leq}$Cluster Editing, welches wie folgt definiert ist.

Problem $^{p\leq}$Cluster Editing ($^{p\leq}$CE)

Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq V^2$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p\leq}$Cluster-Graph ist

Wir wählen zunächst ein festes $p = 0$, sodass keine Einschränkungen hinsichtlich der Anzahl der Cliquen des Zielgraphen vorliegen.

Satz 3.1. $^{0\leq}$CE *ist* NP*-vollständig.*

Beweis. Für die Härte reduzieren wir von 3X3C mit der Reduktion aus [SST04]. Wir verwenden „Tripel“ als Synonym für eine 3-elementige Menge.

Problem 3-Exact 3-Cover (3X3C)

Eingabe	Eine Menge C von Tripeln mit Elementen aus $U = \{u_1, \ldots, u_{3n}\}$ derart, dass jedes Element aus U in maximal 3 Tripeln vorkommt
Frage	Gibt es eine Menge $I \subseteq C$ der Größe n, die U überdeckt

Wir reduzieren mit der Reduktionsfunktion 3.1.

Algorithmus 3.1 Reduktion von 3X3C auf $^{0\leq}$CE

```
1:  function REDUCE(⟨C, U, n⟩)
2:      for all X ∈ C do
3:          V_X := {v_{X,1}, ..., v_{X,30n}}
4:          V := V ∪ V_X
5:          E_1 := E_1 ∪ (V_X ∘ V_X)          ▷ Cliquenbildung von V_X
6:          E_2 := E_2 ∪ (X ∘ V_X)            ▷ Verbinden von V_X und X
7:          E_3 := E_3 ∪ (X ∘ X)              ▷ Cliquenbildung von X
8:      for all u ∈ U do
9:          V := V ∪ {u}
10:     G := (V, E_1 ∪ E_2 ∪ E_3)
11:     N := 30n · 3 · (|C| − n)
12:     M := |E_3| − 3n
13:     return ⟨G, N + M⟩
```

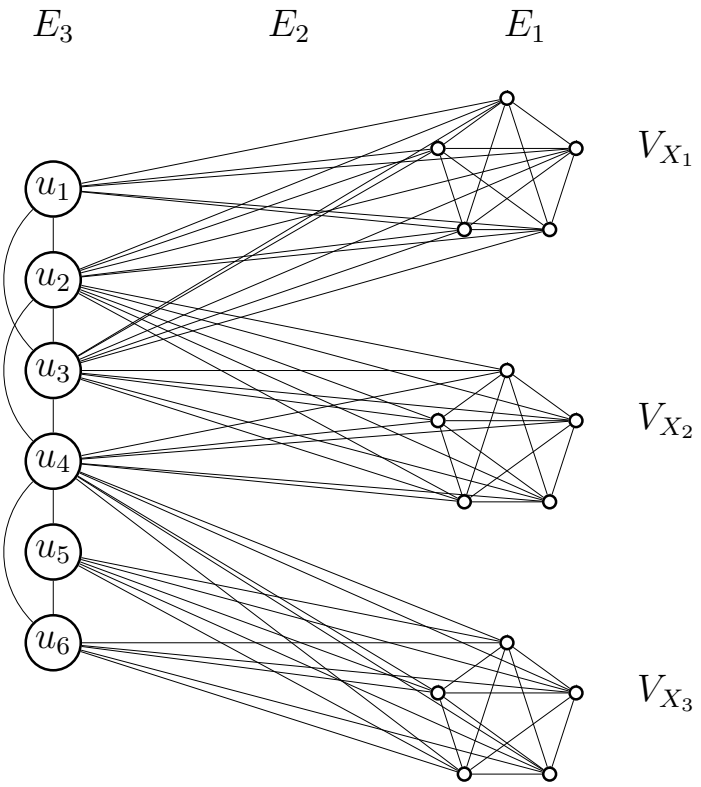


Abbildung 3.1: Reduktion von 3X3C auf ${}^{0}\!\leq$CE

Beispiel 3.2 Wir wählen als Eingabe

$$\begin{aligned} n &= 2, \\ U &= \{u_1, \ldots, u_6\}, \\ C &= \{X_1, X_2, X_3\}, \\ X_1 &= \{u_1, u_2, u_3\}, \\ X_2 &= \{u_2, u_3, u_4\}, \\ X_3 &= \{u_4, u_5, u_6\}. \end{aligned}$$

Der Übersicht halber haben unsere Cliquen nur eine Größe von 5 anstelle der tatsächlichen Größe von $2 \cdot 30$. Die Ausgabe des Algorithmus ist in Abbildung 3.1 zu sehen. Um eine Überdeckung zu erhalten würden wir in diesem Beispiel die $M = 2$ Kanten $\{u_3, u_4\}$ und $\{u_2, u_4\}$ und die $N = 180$ Kanten $E(X_2, V_{X_2})$ entfernen, womit wir gleichzeitig einen ${}^{0}\!\leq$Cluster-Graphen erhalten würden.

Da $|C|$ im allgemeinen größer als n ist, bezeichnet $|C| - n$ die Anzahl der Tripel, die wir aus C entfernen müssen um eine Überdeckung zu erhalten. Da jedes Tripel aus C aus genau 3 Elementen besteht und jedes Element wegen Zeile 6 mit genau $30n$ Knoten verbunden ist, sehen wir, dass N genau die Anzahl der Kanten aus E_2 ist, welche bei einer Überdeckung entfernt werden müssen. Da außerdem jedes Tripel wegen Zeile 7 genau aus 3 Kanten besteht, ist M genau die Anzahl der Kanten aus E_3, welche bei einer Überdeckung entfernt werden müssen. Falls also eine Überdeckung existiert, so gibt es demnach ein F mit $|F| \leq N + M$, sodass G_F ein ${}^{0}\!\leq$Cluster-Graph ist. Die Implikation in die andere Richtung ist dagegen aufwändig zu zeigen und kann in [SST04] nachgelesen werden. □

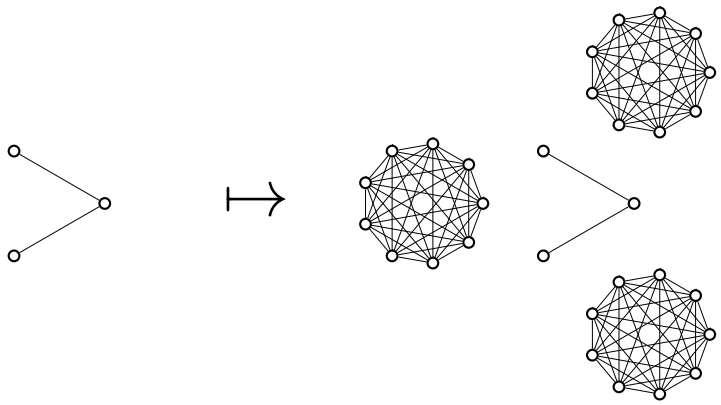


Abbildung 3.2: Reduktion von ${}^{0\leq}$CE auf ${}^{3\leq}$CE

Die Frage, die sich stellt ist ob ${}^{p\leq}$CE auch für ein beliebiges festes p NP-vollständig ist. Die Antwort erhalten wir aus folgendem Korollar.

Korollar 3.3. ${}^{p\leq}$CE *ist* NP*-vollständig für alle* $p \in \mathbb{N}$.

Beweis. ${}^{0\leq}$CE reduziert sich für ein beliebiges p auf ${}^{p\leq}$CE. Wir benutzen dabei die Reduktionsfunktion 3.2 aus [SST04] mit $i = 0$. Eine mögliche Reduktion ist in Abbildung 3.2 zu sehen.

Algorithmus 3.2 Reduktion von ${}^{i\leq}$CE auf ${}^{p\leq}$CE für alle $i \leq p$

```
1: function REDUCE(⟨G, k⟩,i)
2:     if k ≥ (|V[G]| choose 2) then return ⟨G⁺, k⟩        ▷ G⁺ positive Instanz
3:     erzeuge (p − i) Cliquen der Größe |V[G]|²
4:     erzeuge einen Graphen G′, indem G und die Cliquen vereinigt werden
5:     return ⟨G′, k⟩
```

Die Reduktion läuft in Polynomialzeit. Für die Korrektheit nehmen wir an, dass $\langle G, k\rangle \in {}^{i\leq}$CE gilt. Nun liegt sicherlich auch $\langle G', k\rangle$ in ${}^{p\leq}$CE, da G ein Teilgraph von G' ist und wir mögliche Modifikationen von G auf G' übertragen können. Für die Implikation in die andere Richtung wird benutzt, dass für das Hinzufügen eines Knotens zu einer $|V[G]|^2$ großen Clique $|V[G]|^2$ Kanten gesetzt werden müssen. Für das Entfernen eines Knotens einer $|V[G]|^2$ großen Clique müssen $(|V[G]|^2 - 1)$ Kanten entfernt werden. In Zeile 2 darf eine positive Instanz zurückgegeben werden, da $k \geq |E[G]|$ gilt und daher beliebig viele Modifikationen am Eingabegraphen G gemacht werden können. Daher ist es immer möglich G zu einem ${}^{i\leq}$Cluster-Graphen zu modifizieren. Zeile 2 garantiert uns außerdem $k < \binom{|V[G]|}{2} \leq |V[G]|^2 - 1$. Deshalb können den $|V[G]|^2$ großen Cliquen in G' weder Kanten hinzugefügt noch Kanten entfernt werden und sämtliche Modifikationen müssen am Teilgraphen G geschehen. □

3.2 ${}^{p=}$Cluster Editing

Als nächstes beschäftigen wir uns mit ${}^{p=}$Cluster Editing. Es ist wie folgt definiert.

Problem ${}^{p=}$Cluster Editing (${}^{p=}$CE)

Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq V^2$ mit $\lvert F\rvert \leq k$, sodass G_F ein ${}^{p=}$Cluster-Graph ist

Lemma 3.4. $^{p=}\text{CE} \in \text{P}$ *für* $p = 0, 1$.

Beweis. Für ein festes $p = 0$ ist das Problem trivial und liegt damit in P. Für $p = 1$ liegt es ebenfalls in P, da nur überprüft werden muss, ob G zum vollständigen Graphen modifiziert werden kann. Dies ist genau dann der Fall, wenn $k \geq |V^2| - |E|$ gilt. □

Für ein festes $p = 2$ wird das Problem schwierig.

Satz 3.5. $^{2=}\text{CE}$ *ist* NP*-vollständig.*

Beweis. Für die Härte wird in [SST04] von Balanced 2-Coloring of a 3-Uniform Hypergraph reduziert, welches in [Lov73] als NP-vollständig klassifiziert worden ist.

Problem	Balanced 2-Coloring of a 3-Uniform Hypergraph (B2C3UH)
Eingabe	Ein Hypergraph $H = (V, E)$ mit $\|e\| = 3$ für alle $e \in E$
Frage	Besitzt G eine Färbung mit genau 2 Farben, wobei jede Farbe genau $\frac{\|V\|}{2}$ Knoten einfärbt, sodass die Knoten jeder Kante nicht alle gleichfarbig sind

Es lässt sich die Reduktionsfunktion 3.3 verwenden. Eine mögliche Reduktion ist in

Algorithmus 3.3 Reduktion von B2C3UH auf $^{2=}\text{CE}$

```
1:  function REDUCE(⟨H⟩)
2:      for all u ∈ V[H] do
3:          V_u := {v_{u,i} | i ∈ {1, ..., 2|V[H]|^3}}
4:          V_1 := V_1 ∪ V_u
5:          E_1 := E_1 ∪ (V_u ∘ V_u)                    ▷ Cliquenbildung
6:      for all 1 ≤ i < j < l ≤ |V[H]| do
7:          if {u_i, u_j, u_l} ∉ E[H] then
8:              r := 2(i|V[H]|^2 + j|V[H]| + l) − 1
9:              E_{i,j,l} := {{v_{i,r}, v_{j,r}}, {v_{j,r+1}, v_{l,r}}, {v_{l,r+1}, v_{i,r+1}}}
10:             E_2 := E_2 ∪ E_{i,j,l}
11:     G := (V_1, E_1 ∪ E_2)
12:     N := 2·binom(½|V[H]|, 2)·(4|V[H]|^6 − (|V[H]| − 2))
13:     M := (|V[H]|/2)^2 · (|V[H]| − 2) − |E[H]|
14:     return ⟨G, N + M⟩
```

Abbildung 3.3 zu sehen, wobei unsere Cliquen der Übersicht halber nur eine Größe von 5 anstelle der tatsächlichen Größe von $2 \cdot 4^3$ haben.

Der Algorithmus arbeitet in Polynomialzeit. Für die Korrektheit nehmen wir an, der Eingabegraph H läge in B2C3UH und besäße damit eine balancierte 2-Färbung. Wir fassen die 2-Färbung als Partitionierung $(A[H], B[H])$ auf, wobei $A[H]$ und $B[H]$ Knotenmengen der jeweiligen Farben darstellen. Wir übertragen die Färbung auf unseren Ausgabegraphen G, indem wir die jeweiligen Cliquen einfärben. Formal definieren wir hierzu $A[G] := \{V_u \mid u \in A[H]\}$ und $B[G] := \{V_u \mid u \in B[H]\}$. Nun zeigen wir, dass der $^{2=}$Cluster-Graph $C := (V[G], A[G]^2 \cup B[G]^2)$ durch genau $N + M$ Modifikationen in G entsteht. Hierzu betrachten wir mit selbiger Definition von $E_{i,j,l}$ wie im Algorithmus 3.3 die folgende Kantenmenge.

$$E' := E[G] \cup \underbrace{\{E_{i,j,l} \mid 1 \leq i < j < l \leq |V[H]| \text{ und } \{u_i, u_j, u_l\} \in E[H]\}}_{E_3}$$

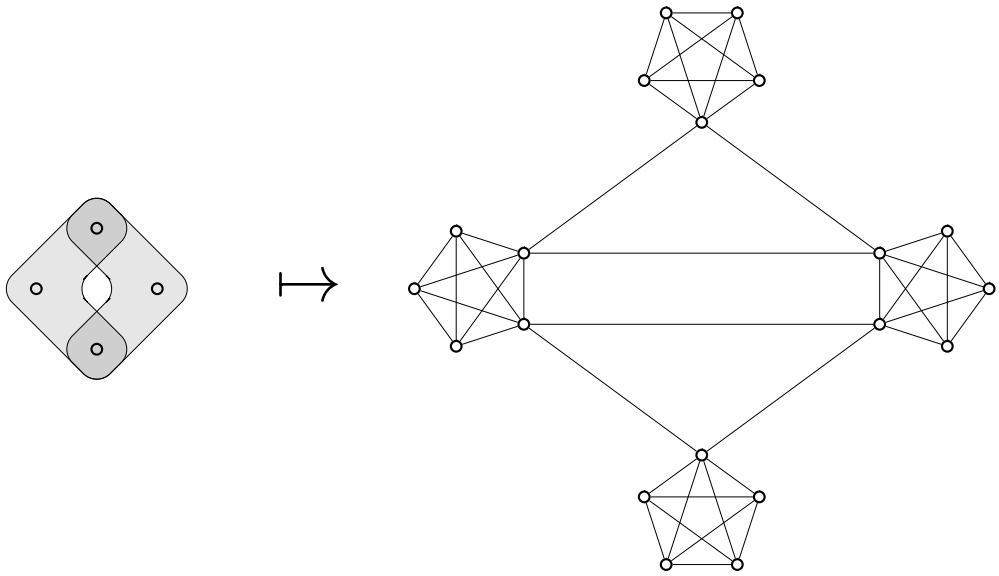


Abbildung 3.3: Reduktion von B2C3UH auf $^{2=}$CE

Zuerst betrachten wir die Modifikationsmenge F', welche benötigt wird um aus der Kantenmenge E' den $^{2=}$Cluster-Graph C zu erhalten. Formal gilt $F' := (E' \setminus E[C]) \cup (E[C] \setminus E')$. Um die Kardinalität von F' abzuschätzen betrachten wir $|E' \setminus E[C]|$. Hierbei handelt es sich um die Kanten zwischen den Clustern. Da die Färbung balanciert ist, gilt nach Definition $|A[H]| = |B[H]| = \frac{|V[H]|}{2}$. Somit haben wir $\left(\frac{|V[H]|}{2}\right)^2$ Möglichkeiten zwei Cliquen V_i und V_j aus den verschiedenen Clustern zu wählen. Die beiden Cliquen verbinden nun genau $(|V[H]| - 2)$ viele Kanten, da für jedes l mit $i \neq l \neq j$ eine Kante in $E_{i,j,l}$ besteht. Somit erhalten wir $|E' \setminus E[C]| = \left(\frac{|V[H]|}{2}\right)^2 (|V[H]| - 2) = M + |E[H]|$.

Nun berechnen wir $|E[C] \setminus E'|$. Dies sind Kanten innerhalb der Cluster. Wir betrachten als erstes die Menge $A[H]$. Wir haben $\binom{\frac{1}{2}|V[H]|}{2}$ viele Möglichkeiten zwei verschieden Cliquen V_i und V_j aus $A[G]$ zu wählen. Auch hier verbinden die Cliquen genau $(|V[H]| - 2)$ viele Kanten. Wir müssen diesmal jedoch die Kanten zählen, welche nicht in E' vorhanden sind. Somit erhalten wir diesmal $|V_i \circ V_j| - (|V[H]| - 2))$ fehlende Kanten zwischen den Cliquen V_i und V_j. Analog zählen wir die Kanten aus $B[G]$, sodass wir die doppelte Anzahl, also $2\binom{\frac{1}{2}|V[H]|}{2}(|V_i \circ V_j| - (|V[H]| - 2)) = N$, erhalten. Somit ergibt sich die nächste Gleichung.

$$|F'| = N + M + |E[H]| \tag{1}$$

Nun können wir F betrachten, welches die tatsächliche Modifikationsmenge darstellt. Diese ist durch $F := (E[G] \setminus E[C]) \cup (E[C] \setminus E[G])$ definiert. Es gilt $E[G] = (E' \setminus E_3) = (E' \setminus E_3) \cup (E_3 \setminus E')$ und daher unterscheiden sich die Kantenmengen $E[G]$ und E' um genau die Menge E_3. Folglich müssen sich auch die Modifikationsmengen um genau die Kantenmenge E_3 unterscheiden und auch durch Umformung lässt sich die analoge Gleichung $F = (F' \setminus E_3) \cup (E_3 \setminus F')$ für die Modifikationsmengen ableiten. Somit gilt die folgende Gleichung.

$$|F| = |F' \setminus E_3| + |E_3 \setminus F'| = |F'| - |F' \cap E_3| + |\bar{F}' \cap E_3| \tag{2}$$

Für jede Kante $\{u_i, u_j, u_l\}$ in $E[H]$ existieren die drei Kanten $E_{i,j,l}$ in E_3. Somit gilt die folgende Gleichung.

$$E_3 = 3E[H] \tag{3}$$

Da die Kanten aus E_3 die Kanten aus $E[H]$ repräsentieren, müssen aufgrund der Färbungsbedingung genau zwei der Knoten u_i, u_j und u_l gleichfarbig und der dritte andersfarbig sein. Somit liegt genau eine Kante aus $E_{i,j,l}$ in $A[G]^2$ oder $B[G]^2$ und genau zwei Kanten aus $E_{i,j,l}$ liegen in $A[G] \circ B[G]$. Da die beiden Kanten aus $A[G] \circ B[G]$ in C nicht vorhanden sind, wurde die Kante modifiziert und liegt folglich in F'. Die Kante aus $A[G]^2$ oder $B[G]^2$ liegt jedoch in C und liegt folglich in $\bar{F}'$. Somit erhalten wir die folgende Aufteilung der Kanten in E_3.

$$|F' \cap E_3| = \frac{2}{3}E_3 \overset{3}{=} 2E[H] \tag{4}$$

$$|\bar{F}' \cap E_3| = \frac{1}{3}E_3 \overset{3}{=} E[H] \tag{5}$$

Schließlich erhalten wir die folgende Gleichung.

$$\begin{aligned} |F| &\overset{2}{=} |F'| - |F' \cap E_3| + |\bar{F}' \cap E_3| \\ &\overset{4,5}{=} |F'| - |E[H]| \\ &\overset{1}{=} N + M \end{aligned}$$

Wir haben nun gezeigt, dass F tatsächlich eine gültige Modifikationsmenge darstellt. Die Korrektheit der anderen Richtung kann in [SST04] nachgelesen werden. □

Auch für alle $p \geq 2$ ist das Problem schwer.

Korollar 3.6. *${}^{p=}$CE ist* NP*-vollständig für alle $p \in \mathbb{N} \setminus \{0, 1\}$.*

Beweis. Analog zu Korollar 3.3 reduzieren wir von ${}^{2=}$CE auf ein beliebiges ${}^{p=}$CE Problem mit $p \geq 2$. Es kann mit selbiger Argumentation die Reduktionsfunktion 3.2 verwendet werden. □

3.3 ${}^{p\geq}$Cluster Editing

Nun beschäftigen wir uns mit ${}^{p\geq}$Cluster Editing. Es ist wie folgt definiert.

Problem ${}^{p\geq}$Cluster Editing (${}^{p\geq}$CE)

Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq V^2$ mit $\lvert F\rvert \leq k$, sodass G_F ein ${}^{p\geq}$Cluster-Graph ist

Als erstes machen wir uns über den Zusammenhang von ${}^{\geq}$CE und ${}^{=}$CE Gedanken.

Lemma 3.7. *Sei $f : \Sigma^* \to \mathbb{N}$ eine Funktion und sei ${}^{=}$CE bei Eingabe $x \in \Sigma^*$ in $O(f(x))$ entscheidbar. Dann lässt sich ${}^{\geq}$CE in $O(|x| \cdot f(x))$ entscheiden.*

Beweis. Es sei DECIDE[${}^{=}$CE] ein Algorithmus, der ${}^{=}$CE in $O(f(x))$ entscheidet. Wir verwenden den Algorithmus 3.4 um ${}^{\geq}$CE zu entscheiden.

Um uns die Korrektheit klar zu machen, sehen wir ein, dass $\langle G, k, p\rangle$ genau dann in ${}^{\geq}$CE liegt, wenn es ein $i \in \{1, \ldots, p\}$ gibt, für das $\langle G, k, i\rangle$ in ${}^{=}$CE liegt. Wir können

Algorithmus 3.4 Entscheidungsalgorithmus von $^{\geq}$CE

```
1: function DECIDE(⟨G, k, p⟩)
2:     for all i ∈ {1, ..., min(p, |V[G]|)} do
3:         if DECIDE[=CE](⟨G, k, i⟩) = accept then
4:             accept
5:     reject
```

jedoch behaupten, dass $\langle G, k, j\rangle$ für alle $j > |V[G]|$ nicht in $^{=}$CE liegt, da für eine beliebige Modifikationsmenge F der Graph G_F genau $|V[G_F]| = |V[G]| < j$ viele Knoten besitzt und daher kein $^{j=}$Cluster-Graph sein kann. Aus diesem Grund reicht es in Zeile 2 aus i von 1 bis $\min(p, |V[G]|)$ laufen zu lassen und wegen $\min(p, |V[G]|) \leq |V[G]| \leq |x|$ arbeitet der Algorithmus daher in $O(|x| \cdot f(x))$ □

Wir können dieselben Betrachtungen auch analog für $^{\leq}$CE machen.

Lemma 3.8. *Sei $f : \Sigma^* \to \mathbb{N}$ eine Funktion und sei $^{=}$CE bei Eingabe $x \in \Sigma^*$ in $O(f(x))$ entscheidbar. Dann lässt sich $^{\leq}$CE in $O(|x| \cdot f(x))$ entscheiden.*

Beweis. Der Beweis ist analog zum Vorigen. Diesmal lassen wir jedoch i von p bis $|V[G]|$ laufen oder überspringen die Schleife, falls $p > |V[G]|$ gilt. □

Für den Fall, dass wir $^{p\geq}$CE betrachten, bei dem p fest ist, lässt sich eine ähnliche Aussage formulieren.

Lemma 3.9. *Sei $f : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ eine Funktion und sei $^{p=}$CE bei Eingabe $x = \langle G, k\rangle$ in $O(f(|x|, k))$ entscheidbar. Dann lässt sich $^{p\geq}$CE in $O(|x|^{O(1)} \cdot f(|x|^{O(1)}, k))$ entscheiden.*

Beweis. Es sei DECIDE[$^{p=}$CE] ein Algorithmus, der $^{p=}$CE in $O(f(|x|, k))$ entscheidet. REDUCE ist hier der Algorithmus 3.2. Nun lässt sich mit Algorithmus 3.5 die Sprache $^{p\geq}$CE entscheiden.

Algorithmus 3.5 Entscheidungsalgorithmus von $^{p\geq}$CE

```
1: function DECIDE(⟨G, k⟩)
2:     for all i ∈ {1, ..., min(p, |V[G]|)} do
3:         ⟨G', k⟩ := REDUCE(⟨G, k⟩, i)        ▷ Reduktion von i=CE auf p=CE
4:         if DECIDE[p=CE](⟨G', k⟩) = accept then
5:             accept
6:     reject
```

Da uns nur ein $^{p=}$CE-Algorithmus für ein festes p zur Verfügung steht, müssen wir unsere Instanzen aus $^{i=}$CE zunächst auf eine Instanz aus $^{p=}$CE reduzieren. Die Reduktionsfunktion in Zeile 3 geschieht in $|x|^{O(1)}$. Hierbei kann auch der Graph G polynomiell anwachsen, k bleibt jedoch gleich in der Größe, wodurch wir Zeile 4 in $f(|x|^{O(1)}, k))$ erledigen können. Die Schleife durchlaufen wir $O(|x|)$-mal, wodurch wir insgesamt eine Laufzeit von $O(|x|^{O(1)} \cdot f(|x|^{O(1)}, k))$ erhalten. □

Lemma 3.10. $^{p\geq}$CE $\in$ P *für $p = 0, 1$.*

Beweis. Aus Lemma 3.9 folgt, dass für ein festes $p = 0$ oder $p = 1$ die Sprache $^{p\geq}$CE auch in P liegen muss. □

Als nächstes möchten wir den Fall für $p = 2$ betrachten. Es lässt sich hier die gleiche Aussage wie bei ${}^{2=}$CE machen.

Satz 3.11. ${}^{2\geq}$CE *ist* NP*-vollständig.*

Beweis. Hierzu betrachten wir erneut die Reduktionsfunktion 3.3. In dem Beweis von Satz 3.5 haben wir

$$\langle H \rangle \in \text{B2C3UH} \Leftrightarrow \langle G, N+M \rangle \in {}^{2=}\text{CE}$$

gezeigt. Wir werden nun

$$\langle G, N+M \rangle \in {}^{2=}\text{CE} \Leftrightarrow \langle G, N+M \rangle \in {}^{2\geq}\text{CE} \tag{1}$$

beweisen und zeigen damit, dass B2C3UH auf ${}^{2\geq}$CE mit der Reduktionsfunktion 3.3 reduzierbar ist.
(1)„⇒“. Jeder ${}^{2=}$Cluster-Graph ist auch ein ${}^{2\geq}$Cluster-Graph und somit gilt die Inklusion ${}^{2=}\text{CE} \subseteq {}^{2\geq}\text{CE}$.
(1)„⇐“. Es gilt ${}^{0=}\text{CE} = \emptyset$, da unsere Graphen per Definition mindestens einen Knoten besitzen. Aus diesem Grund reicht es $\langle G, N+M \rangle \notin {}^{1=}\text{CE}$ zu zeigen. Dies ist genau dann der Fall, wenn $|V[G]^2| - |E[G]| > N + M$ gilt. Um dies zu überprüfen definieren wir $v := |V[H]|$.

$$\begin{aligned}
|V[G]^2| - |E[G]| &= |V[G]^2| - |E_1 \cup E_2| \\
&= |V[G]^2| - (|E_1| + 3|\bar{E}[H]|) \\
&= \binom{v \cdot 2v^3}{2} - (v\binom{2v^3}{2} + 3\binom{v}{3} - 3|E[H]|) \\
&= v^4(2v^4 - 1) - v \cdot v^3(2v^3 - 1) - \frac{1}{2}v(v-1)(v-2) + 3|E[H]| \\
&= 2v^8 - 2v^7 - \frac{1}{2}v^3 + \frac{3}{2}v^2 - v + 3|E[H]|
\end{aligned} \tag{2}$$

Für $N + M$ erhalten wir die folgende Rechnung.

$$\begin{aligned}
N + M &= 2\binom{\frac{1}{2}v}{2}(4v^6 - (v-2)) + \left(\frac{v}{2}\right)^2 (v-2) - |E[H]| \\
&= \frac{1}{2}v(\frac{1}{2}v - 1)(4v^6 - v + 2) + \frac{1}{4}v^2(v-2) - |E[H]| \\
&= v^8 - 2v^7 + \frac{1}{2}v^2 - v - |E[H]|
\end{aligned} \tag{3}$$

Insgesamt gilt für alle v

$$|V[G]^2| - |E[G]| - (N+M) \overset{2,3}{=} v^8 - \frac{1}{2}v^3 + v^2 + 4|E[H]| > 0.$$

Somit muss $\langle G, N+M \rangle$ in ${}^{2=}$CE liegen. □

Auch für alle $p \geq 2$ ist das Problem schwer.

Korollar 3.12. ${}^{p\geq}$CE *ist* NP*-vollständig für alle* $p \in \mathbb{N} \setminus \{0, 1\}$.

Tabelle 3.1: Übersicht von Cluster Editing

CE	**Eingabe**	**Para**	**Fest**	**Resultat**	**Begründung**
$^{p\leq}$	p, k			NP-vollständig	folgt aus Korollar 3.3
	p	k		$\in$ FPT	Korollar 6.9
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.9
	k	p		P $\neq$ NP $\Rightarrow \notin$ XP	folgt aus Korollar 3.3
	k		$p \in \{0, \ldots\}$	NP-vollständig	Korollar 3.3
		p, k		$\in$ FPT	folgt aus Korollar 6.9
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Korollar 6.9
		k	$p \in \{0, \ldots\}$	$\in$ FPT	folgt aus Korollar 6.9
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.9
$^{p=}$	p, k			NP-vollständig	folgt aus Korollar 3.6
	p	k		$\in$ FPT	Satz 6.7
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 6.7
	k	p		P $\neq$ NP $\Rightarrow \notin$ XP	folgt aus Korollar 3.6
	k		$p \in \{0, 1\}$	$\in$ P	Lemma 3.4
	k		$p \in \{2, \ldots\}$	NP-vollständig	Korollar 3.6
		p, k		$\in$ FPT	folgt aus Satz 6.7
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Satz 6.7
		k	$p \in \{0, \ldots\}$	$\in$ SUBEPT	Satz 6.10
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 6.7
$^{p\geq}$	p, k			NP-vollständig	folgt aus Korollar 3.12
	p	k		$\in$ FPT	Korollar 6.8
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.8
	k	p		P $\neq$ NP $\Rightarrow \notin$ XP	folgt aus Korollar 3.12
	k		$p \in \{0, 1\}$	$\in$ P	Lemma 3.10
	k		$p \in \{2, \ldots\}$	NP-vollständig	Korollar 3.12
		p, k		$\in$ FPT	folgt aus Korollar 6.8
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Korollar 6.8
		k	$p \in \{0, \ldots\}$	$\in$ SUBEPT	Korollar 6.11
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.8

Beweis. Wir reduzieren wir von $^{2\geq}$CE auf ein beliebiges $^{p\geq}$CE Problem mit $p \geq 2$, indem wir mit selbiger Argumentation die Reduktionsfunktion 3.2 mit $i = 2$ verwenden. □

4 Cluster Deletion

Nun beschäftigen wir uns mit Cluster Deletion. Der Unterschied zu Cluster Editing liegt darin, dass beim Cluster Deletion nur Kanten aus dem Eingabegraphen gelöscht werden dürfen.

4.1 $^{p\leq}$Cluster Deletion

Unser erster Fokus liegt bei $^{p\leq}$Cluster Deletion, welches wie folgt definiert ist.

Problem	$^{p\leq}$Cluster Deletion ($^{p\leq}$CD)
Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq E$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p\leq}$Cluster-Graph ist

Wir wählen zunächst ein festes $p = 0$, sodass keine Einschränkungen hinsichtlich der Anzahl der Cliquen des Zielgraphen vorliegen.

Satz 4.1. $^{0\leq}$CD *ist* NP*-vollständig.*

Beweis. Hier lässt sich der Beweis von Satz 3.1 analog übertragen, da nur Kanten gelöscht werden um eine Überdeckung zu erhalten. □

Korollar 4.2. $^{p\leq}$CD *ist* NP*-vollständig für alle* $p \in \mathbb{N}$*.*

Beweis. Auch hier lässt sich von $^{0\leq}$CD aus reduzieren. Es kann die selbe Reduktionsfunktion 3.2 wie bei Cluster Editing verwendet werden. □

4.2 $^{p=}$Cluster Deletion

Als nächstes beschäftigen wir uns mit $^{p=}$Cluster Deletion. Es ist wie folgt definiert.

Problem	$^{p=}$Cluster Deletion ($^{p=}$CD)
Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq E$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p=}$Cluster-Graph ist

Für ein festes $p = 2$ erscheint das Problem leichter als $^{2=}$CE.

Satz 4.3. $^{p=}$CD $\in$ P *für alle* $p \in \{0, 1, 2\}$*.*

Beweis. Analog zu Cluster Editing ist das Problem für ein festes $p = 0$ trivial und liegt damit in P. Für $p = 1$ liegt es ebenfalls in P, da nur überprüft werden muss, ob G der vollständige Graph ist. Dies ist genau dann der Fall, wenn $|E| = |V^2|$ gilt. Um die Mitgliedschaft in P für $p = 2$ zu zeigen, betrachten wir bipartite Graphen.

Definition 4.4. *Eine disjunkte Partition* (A, B) *einer Knotenmenge heißt Bipartition, falls* $E(A, A) = E(B, B) = \emptyset$ *gilt. Ein Graph ist bipartit, falls seine Knotenmenge eine Bipartition besitzt.*

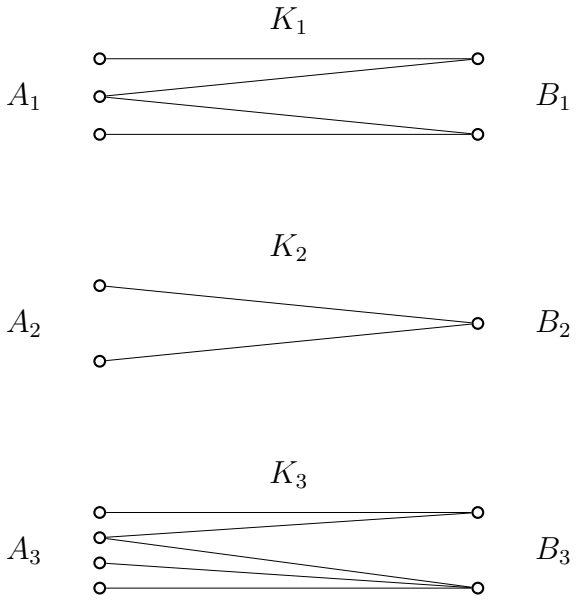


Abbildung 4.1: Der Graph $\bar{G}$ mit dessen Bipartitionen

Wir benutzen nun den Entscheidungsalgorithmus 4.6 aus [SST04]. Die Mitgliedschaft

Algorithmus 4.6 P-Entscheidungsalgorithmus für $^{2=}$CD

```
1: function DECIDE(⟨G, k⟩)
2:     berechne die Zusammenhangskomponenten von Ḡ und nenne sie K_1, ..., K_t
3:     for i = 1, ..., t do
4:         if (V[K_i], E[Ḡ]) ist kein bipartiter Graph then
5:             reject
6:         berechne Bipartition (A_i, B_i) des Graphen (V[K_i], E[Ḡ]) mit |A_i| ≥ |B_i|
7:     if |E[G](A_1 ∪ ... ∪ A_t, B_1 ∪ ... ∪ B_t)| ≤ k then
8:         accept
9:     else
10:        reject
```

in P ist darin begründet, dass sich Bipartition in Polynomialzeit berechnen lassen. Weniger offensichtlich ist die Korrektheit. Der Graph $\bar{G}$ mit dessen Bipartitionen ist in Abbildung 4.1 zu sehen. Die Beweisidee ist die selbe wie in [SST04]. Wir argumentieren zunächst für die Fälle mit $B_i \neq \emptyset$ und weil $|A_i| \geq |B_i|$ folglich auch $A_i \neq \emptyset$. Da K_i zusammenhängend ist, existiert ein Pfad $x_1, y_1, \ldots, x_s, y_s$ in $\bar{G}$, der jeden Knoten aus K_i mindestens einmal besucht. Ohne Einschränkungen wählen wir den Startknoten x_1 aus A_i und somit müssen induktiv alle $y_j \in B_i$ und $x_j \in A_i$ sein, da keine Kanten innerhalb von A_i bzw. B_i existieren. Da außerdem aufeinanderfolgende Knoten x_j, y_j in $\bar{G}$ verbunden sind, existiert in G keine Kante zwischen x_j und y_j. Da $F \subseteq E$ gilt und damit nur Kanten entfernt werden können, müssen x_j und y_j zwangsläufig in verschiedenen Cliquen sein. Analog müssen auch y_j und x_{j+1} in verschiedenen Cliquen landen und da es nur zwei Cliquen in G_F gibt, gehören x_j und x_{j+1} wieder der selben Clique an. Die Beobachtungen gelten für alle j woraus sich ergibt, dass die komplette Menge A_i der jeweils anderen Clique als der von B_i angehören muss. Auch für $B_i = \emptyset$ und somit $|A_i| = 1$, da K_i zusammenhängend ist, kann ohne Einschränkung gefordert werden, dass A_i und B_i unterschiedlichen Cliquen in G_F angehören werden.

Ansonsten existieren keine weiteren Kanten in $\bar{G}$ und damit sind insbesondere alle A_i und B_j, sowie alle A_i und A_j und auch alle B_i und B_j für $i \neq j$ vollständig in G verbunden. Somit gibt es prinzipiell 2^t mögliche Zielgraphen G_F, da jeweils für jedes A_i entschieden werden muss welche der beiden Cliquen es in G_F angehören wird. Die B_i gehören somit der jeweils anderen Clique an. Um herauszufinden welche der Möglichkeiten $|F|$ minimiert, möchten wir nun die Kardinalitäten von A_i und B_i mit a_i und b_i bezeichnen. Die Anzahl der zu entfernenden Kanten setzt sich wie folgt zusammen. $|F| = \sum_{i \neq j} a_i b_j + c_1$, wobei die Summe sämtliche Kanten zwischen A_i und B_j und die Zahl c_1 die Anzahl der Kanten zwischen A_i und B_i in G beschreibt. Umgeformt ergibt sich

$$|F| = \sum_{i,j} a_i b_j \underbrace{- \sum_i a_i b_i}_{-c_2} + c_1,$$

wobei c_2 die Anzahl der Kanten zwischen A_i und B_i in $\bar{G}$ beschreibt. c_2 ist daher für einen festen Eingabegraphen G konstant und kann daher nicht minimiert werden. Betrachten wir also

$$\begin{aligned} \sum_{i,j} a_i b_j &= (\sum_i a_i)(\sum_i b_i) \\ &= \underbrace{(\sum_i a_i)}_{x} \underbrace{(\sum_i a_i + \sum_i b_i}_{c_3} \underbrace{- \sum_i a_i)}_{x}. \end{aligned}$$

Die Zahl c_3 beschreibt die Anzahl der Knoten in G und ist für einen festen Graphen G konstant und es gilt daher das Minimum von $x(c_3 - x)$ mit $x \in \{0, \ldots, c_3\}$ zu ermitteln, welches bekanntlich bei minimalem oder maximalem x angenommen wird. Aus Symmetriegründen führen beide Optimierungen zum selben Ergebnis und es ist daher nicht relevant, ob wir $\sum_i b_i$ oder $\sum_i a_i$ maximieren. Der Algorithmus ist insbesondere korrekt, da er $\sum_i a_i$ maximiert. □

Für ein festes $p \geq 3$ wird das Problem schwer.

Satz 4.5. $^{p=}$CD *ist* NP*-vollständig für alle* $p \in \mathbb{N} \setminus \{0, 1, 2\}$.

Beweis. Wir reduzieren von $^{p=}$Coloring mittels der Reduktion aus [SST04].

Problem	$^{p=}$Coloring
Eingabe	Ein Graph G
Frage	Besitzt G eine Färbung von Knoten mit genau p Farben, sodass je zwei durch eine Kante verbundene Knoten in unterschiedlicher Farbe eingefärbt sind

$^{p\geq}$Coloring, bei dem auch Instanzen mit einer Färbung mit weniger als p Farben zur Sprache gehören, ist bekannterweise NP-vollständig. $^{p\geq}$Coloring lässt sich jedoch auf $^{p=}$Coloring reduzieren. Wir verwenden die Reduktionsfunktion 4.7.

Die Reduktion ist korrekt, weil jeder Graph mit weniger als p Knoten in $^{p\geq}$Coloring liegt, da jeder Knoten in einer anderen Farbe eingefärbt werden kann. Für jeden Graphen G mit mindestens p Knoten gilt jedoch $\langle G \rangle \in {}^{p\geq}$Coloring genau dann, wenn $\langle G \rangle \in {}^{p=}$Coloring. Dies liegt daran, dass die Anzahl der benutzten Farben stets vergrößert werden kann, indem schrittweise ein Knoten mit einer mehrfach vorkommenden Farbe mit einer unbenutzten Farbe eingefärbt wird. Kommen wir nun zur Reduktion

Algorithmus 4.7 Reduktion von $^{p\geq}$Coloring auf $^{p=}$Coloring

```
1: function REDUCE(⟨G⟩)
2:     if |V[G]| < p then
3:         return ⟨G⁺⟩
4:     else
5:         return ⟨G⟩
```

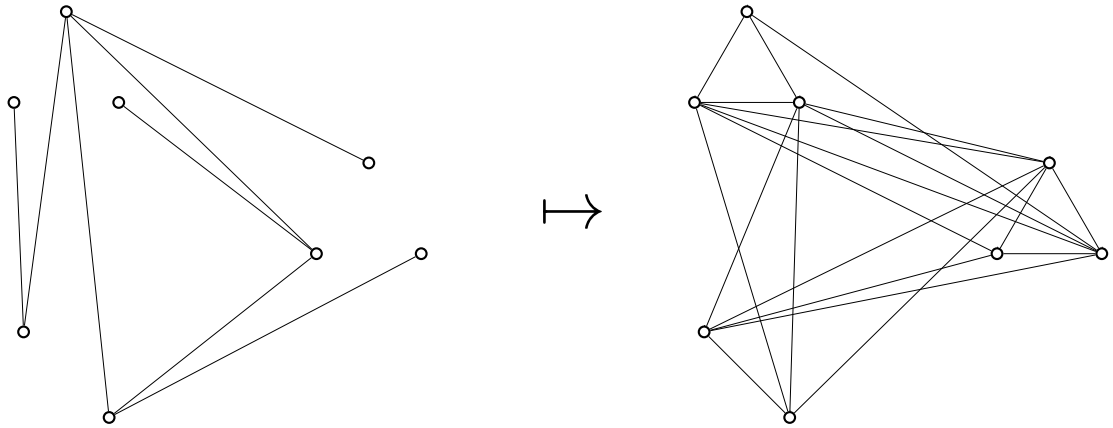


Abbildung 4.2: Reduktion von $^{p=}$Coloring auf $^{p=}$CD

4.8 von $^{p=}$Coloring auf $^{p=}$CD. Eine mögliche Reduktion ist in Abbildung 4.2 zu

Algorithmus 4.8 Reduktion von $^{p=}$Coloring auf $^{p=}$CD

```
1: function REDUCE(⟨G⟩)
2:     return ⟨Ḡ, |Ē|⟩
```

sehen.

Für den Beweis der Korrektheit bemerken wir, dass $k = |\bar{E}|$ gilt und somit beliebig viele Kanten in $\bar{G}$ entfernt werden dürfen. Falls $\langle G\rangle \in {}^{p=}$Coloring, so gibt es Knotenmengen $K_1, \ldots, K_p$, wobei K_i jeweils die Menge der Knoten mit Farbe i bezeichnet. Daher sind die Knoten aus K_i in G untereinander nicht verbunden und bilden daher eine Clique in $\bar{G}$. Durch Entfernen aller Kanten $E[\bar{G}](K_i, K_j)$ für alle $i \neq j$ erhalten wir daher einen $^{p=}$Cluster-Graphen und $\langle \bar{G}, |\bar{E}|\rangle \in {}^{p=}$CD.

Umgekehrt falls $\langle \bar{G}, |\bar{E}|\rangle \in {}^{p=}$CD, so existiert ein $F \subseteq \bar{E}$, sodass $(\bar{G})_F$ ein $^{p=}$Cluster-Graph ist. Außerdem gilt $(\bar{G})_F = \overline{(G_F)}$, da es unerheblich ist, ob zuerst die Kantenmenge F oder der Graph G invertiert wird. Folglich ist G_F ein invertierter $^{p=}$Cluster-Graph und besitzt Knotenmengen $K_1, \ldots, K_p$ mit $E[G_F](K_i, K_i) = \emptyset$ für alle i. Da $F \subseteq \bar{E}$, handelt es sich bei F um eine Hinzunahme von Kanten und daher muss für alle i insbesondere $E[G](K_i, K_i) = \emptyset$ gelten. Aus diesem Grund bildet die Partitionierung $K_1, \ldots, K_p$ eine gültige p-Färbung und $\langle G\rangle \in {}^{p=}$Coloring. □

4.3 $^{p\geq}$Cluster Deletion

Nun beschäftigen wir uns mit $^{p\geq}$Cluster Deletion. Es ist wie folgt definiert.

Problem $^{p\geq}$Cluster Deletion ($^{p\geq}$CD)

Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq E$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p\geq}$Cluster-Graph ist

Tabelle 4.1: Übersicht von Cluster Deletion

CD	Eingabe	Para	Fest	Resultat	Begründung
$^{p\leq}$	p, k			NP-vollständig	folgt aus Korollar 4.2
	p	k		$\in$ FPT	Korollar 6.9
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.9
	k	p		P $\neq$ NP $\Rightarrow \notin$ XP	folgt aus Korollar 4.2
	k		$p \in \{0, \ldots\}$	NP-vollständig	Korollar 4.2
		p, k		$\in$ FPT	folgt aus Korollar 6.9
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Korollar 6.9
		k	$p \in \{0, \ldots\}$	$\in$ FPT	folgt aus Korollar 6.9
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.9
$^{p=}$	p, k			NP-vollständig	folgt aus Satz 4.5
	p	k		$\in$ FPT	Satz 6.7
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 6.7
	k	p		P $\neq$ NP $\Rightarrow \notin$ XP	folgt aus Satz 4.5
	k		$p \in \{0, 1, 2\}$	$\in$ P	Satz 4.3
	k		$p \in \{3, \ldots\}$	NP-vollständig	Satz 4.5
		p, k		$\in$ FPT	folgt aus Satz 6.7
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Satz 6.7
		k	$p \in \{0, \ldots\}$	$\in$ SUBEPT	Satz 6.10
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 6.7
$^{p\geq}$	p, k			NP-vollständig	folgt aus Korollar 4.7
	p	k		$\in$ FPT	Korollar 6.8
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.8
	k	p		P $\neq$ NP $\Rightarrow \notin$ XP	folgt aus Korollar 4.7
	k		$p \in \{0, 1, 2\}$	$\in$ P	Lemma 4.6
	k		$p \in \{3, \ldots\}$	NP-vollständig	Korollar 4.7
		p, k		$\in$ FPT	folgt aus Korollar 6.8
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Korollar 6.8
		k	$p \in \{0, \ldots\}$	$\in$ SUBEPT	Korollar 6.11
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.8

Lemma 4.6. $^{p\geq}$CD $\in$ P *für alle* $p \in \{0, 1, 2\}$.

Beweis. Das Lemma 3.9 gilt auch analog für Cluster Deletion und mit Satz 4.3 erhalten wir eine polynomielle Laufzeit. □

Für $p \geq 3$ wird $^{p\geq}$CD schwer.

Korollar 4.7. $^{p\geq}$CD *ist* NP*-vollständig für alle* $p \in \mathbb{N} \setminus \{0, 1, 2\}$.

Beweis. Um für $p \geq 3$ die NP-Vollständigkeit von $^{p=}$CD zu zeigen, haben wir von $^{p\geq}$Coloring auf $^{p=}$Coloring und von $^{p=}$Coloring auf $^{p=}$CD reduziert. Mit der selben Reduktionsfunktion 4.8 können wir auch direkt von $^{p\geq}$Coloring auf $^{p\geq}$CD reduzieren. □

5 Cluster Completion

Nun beschäftigen wir uns mit Cluster Completion. Der Unterschied zu Cluster Editing liegt darin, dass dem Eingabegraphen bei Cluster Completion nur Kanten hinzugefügt werden dürfen.

5.1 $^{p\leq}$Cluster Completion

Nun möchten wir $^{p\leq}$Cluster Completion untersuchen. Es ist wie folgt definiert.

Problem	$^{p\leq}$Cluster Completion ($^{p\leq}$CC)
Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq \bar{E}$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p\leq}$Cluster-Graph ist

Bei $^{p\leq}$CC liegt die Aufgabe also darin einem Eingabegraphen G derart Kanten hinzuzufügen, sodass sich der modifizierte Graph G_F aus mindestens p disjunkten Cliquen zusammensetzt. Dieses Problem erscheint einfach.

Satz 5.1. $^{\leq}$CC $\in$ P *für alle* $p \in \mathbb{N}$.

Beweis. Der Algorithmus 5.9 entscheidet das Problem in Polynomialzeit.

Algorithmus 5.9 P-Entscheidungsalgorithmus für $^{\leq}$CC

```
1: function DECIDE(⟨G, k, p⟩)
2:     bestimme die Zusammenhangskomponenten Z_1, ..., Z_t von G
3:     if k ≥ Σ_{i=1}^{t} (|V[Z_i]^2| − |E[Z_i]|) and p ≤ t then
4:         accept
5:     else
6:         reject
```

Falls der Algorithmus akzeptiert, so ist $\langle G, k\rangle \in {}^{p\leq}$CC, da wir jede Zusammenhangskomponente zu einer Clique vervollständigen können. Falls der Algorithmus ablehnt, so gibt es zwei Fälle.
Als erstes argumentieren wir für den Fall $p > t$. Da beim Cluster Completion nur Kanten hinzufügt werden dürfen, muss jede Zusammenhangskomponente im Eingabegraph G zu einer Clique vervollständigt werden. Die hierzu benötigte Modifikationsmenge nennen wir F. Auf diese Weise erhalten wir t Cliquen. Nachdem die Zusammenhangskomponenten vervollständigt wurden, können wir die Anzahl der Cliquen nicht mehr erhöhen, da das Entfernen von Kanten unzulässig ist. Somit ist t maximal und es kann keine andere Modifikationsmenge H geben, bei der G_H aus mehr als t Cliquen besteht. Daher darf in diesem Fall abgelehnt werden.
Im zweiten Fall gilt $k < \sum_{i=1}^{t}(\lvert V[Z_i]^2\rvert - \lvert E[Z_i]\rvert)$, wobei die Summe angibt wie viele Kanten für das Vervollständigen der Zusammenhangskomponenten benötigt werden. Jede Hinzunahme von Kanten ist nötig damit G_F ein Cluster-Graph ist. Daher ist $\lvert F\rvert$ minimal, da für alle H mit $\lvert H\rvert < \lvert F\rvert$ der modifizierte Graph G_H kein Cluster-Graph

mehr sein kann. Daher darf auch in diesem Fall abgelehnt werden.
Die Laufzeit bleibt polynomiell, obwohl p ein Teil der Eingabe ist. □

5.2 $^{p=}$Cluster Completion

Als nächstes beschäftigen wir uns mit $^{p=}$Cluster Completion. Es ist wie folgt definiert.

Problem $^{p=}$Cluster Completion ($^{p=}$CC)

Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq \bar{E}$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p=}$Cluster-Graph ist

Wir werden sehen, dass es möglich ist $^{p=}$CC für beliebige feste p in Polynomialzeit zu lösen. Nun beweisen wir folgenden Satz aus [SST04].

Satz 5.2. $^{p=}$CC *ist bei Eingabe* x *in* $O(|x|^p)$ *entscheidbar und liegt damit in* P *für alle* $p \in \mathbb{N}$.

Beweis. Wir benutzen den gleichen Beweis wie in [SST04]. Wir beschreiben den Entscheidungsalgorithmus jedoch in Worten. Die erste Überprüfung, die der Algorithmus macht, ist ob $\langle G, k\rangle \in {}^{p\leq}$CC gilt. Falls dies nicht erfüllt ist, so kann auch nicht $\langle G, k\rangle \in {}^{p=}$CC gelten und der Algorithmus lehnt ab. Andernfalls ist es immer möglich die Anzahl Zusammenhangskomponenten in G zu verringern, indem wir je zwei Zusammenhangskomponenten vereinigen. Jedoch kann nun trotzdem abgelehnt werden, falls $|F| > k$ gilt. Daher wird versucht $|F|$ zu minimieren. Dazu bezeichnen wir die Cliquen in G_F mit $C_1, \ldots, C_p$. $|F|$ setzt sich wie folgt zusammen.

$$\begin{aligned}
|F| &= |E[G_F]| - |E| \\
&= \sum_{i=1}^{p} |V[C_i]^2| - |E| \\
&= \frac{1}{2}\sum_{i=1}^{p} (|V[C_i]|^2 - |V[C_i]|) - |E| \\
&= \frac{1}{2}\sum_{i=1}^{p} |V[C_i]|^2 - \frac{1}{2}|V| - |E|
\end{aligned}$$

Der Term $(-\frac{1}{2}|V| - |E|)$ ist für einen festen Eingabegraphen G konstant und daher gilt es $\sum_{i=1}^{p} |V[C_i]|^2$ zu minimieren. Somit sind nur die Kardinalitäten $c_i := |V[C_i]|$ relevant. Der Algorithmus probiert alle tatsächlich möglichen Belegungen für c_i aus und ermittelt diejenige Belegung, die $\sum_{i=1}^{p} c_i^2$ und folglich $|F|$ minimiert. Repräsentiert werden können solche Belegungen durch einen Vektor $(c_1, \ldots, c_{p-1})$. Es werden nur die Kardinalitäten der ersten $(p-1)$ Cliquen benötigt, da c_p durch $|V[G_F]| - \sum_{i=1}^{p-1} c_i$ rekonstruiert werden kann. Da c_i durch $|V[G]| \leq |x|$ beschränkt ist, gibt es maximal $|x|^{p-1}$ solcher Belegungsvektoren, für die der Wert $\sum_{i=1}^{p} c_i^2$ ausgewertet werden muss. Dies geschieht demnach in $O(|x|^p)$. Es gilt noch zu zeigen, dass die möglichen Belegungen auch in $O(|x|^p)$ aufgezählt werden können, da wir dadurch eine Gesamtlaufzeit von $O(|x|^p)$ garantiert hätten. Hierzu vervollständigen wir zuerst jede Zusammenhangskomponente in G zu einer Clique und bezeichnen diese mit $K_1, \ldots, K_t$. Nun können mögliche Belegungsvektoren rekursiv berechnet werden. Mit B_j bezeichnen wird die Menge der

Belegungsvektoren, für die die Cliquen $K_{j+1}, \ldots, K_t$ nicht in den Cliquen $C_1, \ldots, C_{p-1}$ enthalten sein dürfen. Für den Rekursionsanfang definieren wir

$$B_0 := \{(\underbrace{0, \ldots, 0}_{p-1})\}.$$

B_0 beinhaltet allein den Nullvektor, da die Cliquen $K_1, \ldots, K_t$ nicht in den Cliquen $C_1, \ldots, C_{p-1}$ enthalten sein dürfen und diese daher leer sein müssen. Nun können wir rekursiv B_j berechnen.

$$B_j := B_{j-1} \cup \{x + |V[K_j]| \cdot e_i \mid x \in B_{j-1}, i \in \{1, \ldots, p-1\}\}$$

Der Ausdruck e_i ist dabei der i-te Einheitsvektor. Gehen wir von B_{j-1} nach B_j über, so müssen wir alle Möglichkeiten in Betracht ziehen die Clique K_j zu einer der Cliquen $C_1, \ldots, C_{p-1}$ hinzuzufügen. Somit addieren wir für jedem Belegungsvektor die Kardinalität von $V[K_j]$ zu einer der möglichen $(p-1)$ Kardinalitäten der Cliquen $C_1, \ldots, C_{p-1}$. Schließlich erhalten wir nach t Schritten die Menge B_t, welche nach Eliminierung von Belegungsvektoren mit $c_i = 0$ alle möglichen Belegungsvektoren enthält. Außerdem ist der Zeitbedarf jedes Rekursionsschrittes durch die Kardinalität von B_j und daher durch $O(|x|^{p-1})$ beschränkt. B_t lässt sich daher in $O(t \cdot |x|^{p-1}) \subseteq O(|x|^p)$ ermitteln, was noch zu zeigen war. □

5.3 $^{p\geq}$Cluster Completion

Nun möchten wir $^{p\geq}$Cluster Completion untersuchen. Es ist wie folgt definiert.

Problem	$^{p\geq}$Cluster Completion ($^{p\geq}$CC)
Eingabe	Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq \bar{E}$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p\geq}$Cluster-Graph ist

Die Sprache $^{p\geq}$CC lässt sich durch Mehrfachanwendung unseres $^{p=}$CC-Algorithmus entscheiden.

Korollar 5.3. $^{p\geq}$CC *ist bei Eingabe* x *in* $O(|x|^{O(p)})$ *entscheidbar und liegt damit in* P *für alle* $p \in \mathbb{N}$.

Beweis. Lemma 3.9 lässt sich analog auf $^{p\geq}$CC übertragen und mit Satz 5.2 erhalten wir die gewünschte Laufzeit von $O((|x|^{O(1)})^p) = O(|x|^{O(p)})$. □

Tabelle 5.1: Übersicht von Cluster Completion

CC	Eingabe	Para	Fest	Resultat	Begründung
$^{p\leq}$	p, k			$\in$ P	Satz 5.1
	p	k		$\in$ PT	folgt aus Satz 5.1
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 5.1
	k	p		$\in$ PT	folgt aus Satz 5.1
	k		$p \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 5.1
		p, k		$\in$ PT	folgt aus Satz 5.1
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Satz 5.1
		k	$p \in \{0, \ldots\}$	$\in$ PT	folgt aus Satz 5.1
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 5.1
$^{p=}$	p, k			Unbekannt	
	p	k		$\in$ FPT	Satz 6.7
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 6.7
	k	p		$\in$ XP	Satz 5.2
	k		$p \in \{0, \ldots\}$	$\in$ P	Satz 5.2
		p, k		$\in$ FPT	folgt aus Satz 6.7
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Satz 6.7
		k	$p \in \{0, \ldots\}$	$\in$ PT	folgt aus Satz 5.2
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Satz 6.7
$^{p\geq}$	p, k			Unbekannt	
	p	k		$\in$ FPT	Korollar 6.8
	p		$k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.8
	k	p		$\in$ XP	Korollar 5.3
	k		$p \in \{0, \ldots\}$	$\in$ P	Korollar 5.3
		p, k		$\in$ FPT	folgt aus Korollar 6.8
		p	$k \in \{0, \ldots\}$	$\in$ PT	folgt aus Korollar 6.8
		k	$p \in \{0, \ldots\}$	$\in$ PT	folgt aus Korollar 5.3
			$p, k \in \{0, \ldots\}$	$\in$ P	folgt aus Korollar 6.8

6 Parametrisierte Cluster-Graph-Modifizierung

Als erstes betrachten wir eine parametrisierte Variante von Cluster Editing. Danach werden wir bemerken, dass wir die selben Betrachtungen auch analog für Cluster Deletion und Cluster Completion machen können. Wegen den Lemmata 3.7 bis 3.9 erscheint es sinnvoll die Mitgliedschaft in FPT für $^{=}$CE zu zeigen, um die FPT-Mitgliedschaft für andere Probleme zu folgern. p scheint kein sinnvoller Parameter zu sein, da für einige feste p die Sprache $^{=}$CE bereits NP-vollständig ist und es für den Fall P $\neq$ NP kein Polynomialzeitalgorithmus für alle Scheiben geben kann. Somit läge das parametrisierte Problem nicht in XP und folglich auch nicht in FPT. Aus diesem Grund wählen wir k als Parameter und erhalten folgendes parametrisierte Problem.

Problem k-$^{=}$Cluster Editing (k-$^{=}$CE)

Eingabe	Ein Graph $G = (V, E)$ und $p, k \in \mathbb{N}$
Parameter	k
Frage	Gibt es ein $F \subseteq V^2$ mit $\lvert F\rvert \leq k$, sodass G_F ein $^{p=}$Cluster-Graph ist

Um später p mit dem Parameter k abschätzen zu können, benötigen wir ein Lemma aus [FKP$^+$11].

Lemma 6.1. *Es gibt einen Polynomialzeitalgorithmus, der bei Eingabe $\langle G, k, p\rangle$ eine Ausgabe $\langle G', k', p'\rangle$ mit $p' \leq 6k'$ liefert, die $\langle G, k, p\rangle \in {}^{=}\text{CE} \Leftrightarrow \langle G', k', p'\rangle \in {}^{=}\text{CE}$ erfüllt.*

Beweis. Der Algorithmus 6.10 liefert das gewünschte Ergebnis. Eine isolierte Cliquen ist hierbei eine Clique, die nur mit Knoten der selben Clique verbunden ist. Nun werden

Algorithmus 6.10 Selbstreduktion von $^{=}$CE

```
1:  function KERNEL(⟨G, k, p⟩)
2:      if p ≤ 6k then
3:          return ⟨G, k, p⟩
4:      bestimme die isolierten Cliquen C_1, ..., C_t
5:      if t ≤ p − 2k then
6:          return ⟨G^−, 0, 0⟩                          ▷ negative Instanz
7:      bestimme die isolierten Knoten v_1, ..., v_s von G
8:      if s ≥ 2k + 1 then
9:          return KERNEL(⟨G − v_1, k, p − 1⟩)
10:     bestimme die nach absteigender Größe sortierten isolierten Cliquen C'_1, ..., C'_r,
        welche keine isolierten Knoten sind
11:     return KERNEL(⟨G − V[C'_1], k, p − 1⟩)          ▷ C'_1 ist die größte Clique
```

wir die Beweisidee der Korrektheit von Algorithmus 6.10 erläutern. In Zeile 3 darf die Eingabe zurückgegeben werden, da die geforderte Bedingung erfüllt ist. Der Zielgraph G_F soll bekanntlich genau p isolierte Cliquen enthalten. Da die Modifikationsmenge F

aus maximal k Kanten besteht, sind maximal $2k$ Knoten von Modifikationen betroffen bzw. in einer Kante aus F enthalten. Hieraus folgt, dass G bereits $p - 2k$ isolierte Cliquen enthalten muss und daher darf in Zeile 5 eine negative Instanz zurückgegeben werden. Falls die Anzahl der isolierten Knoten mehr als $2k$ beträgt, so können wir sicher sein, dass mindestens ein isolierter Knoten von Modifikationen unberührt sein wird und dürfen ohne Einschränkung der Allgemeinheit den isolierten Knoten v_1 in Zeile 9 entfernen. Aus der Abfrage in Zeile 5 und der Abfrage in Zeile 2 folgt $t > p - 2k > 6k - 2k = 4k$. Per Definition gilt außerdem $r = t - s$ und da Zeile 8 die Ungleichung $s < 2k + 1$ garantiert, gilt zudem

$$r = t - s \geq 4k + 1 - s \geq 4k + 1 - (2k + 1) + 1 = 2k + 1.$$

Auch hier können wir sicher sein, dass mindestens eine der Cliquen $C'_1, \ldots, C'_r$ von Modifikationen unberührt sein wird. Die restlichen $2k$ Cliquen könnten geteilt oder vereinigt werden um die Anzahl der Cliquen zu erhöhen bzw. zu verringern. Diese Operationen sind mit kleinen Cliquen am günstigsten. Dies bedeutet, dass ein minimales F die größte Clique unberührt lassen wird. Diese Argumentation gilt nur solange die restlichen $2k$ isolierten Cliquen zum Teilen geeignet sind, weshalb wir gefordert haben, dass diese isolierten Cliquen keine isolierten Knoten sind. □

Nun möchten wir Partitionierungen von Knotenmengen untersuchen. Für Cluster Editing sind Partitionierungen von Interesse, zwischen denen nur maximal k Kanten existieren.

Definition 6.2. *Eine disjunkte Partitionierung (V_1, V_2) von V heißt k-Schnitt, falls $|E(V_1, V_2)| \leq k$ gilt.*

Nun ist es von Nutzen k-Schnitte effizient aufzählen zu können.

Lemma 6.3. *k-Schnitte können mit Polynomialzeitverzögerung aufgezählt werden.*

Beweis. Wir verwenden den Beweis aus [FKP+11]. Der rekursive Algorithmus 6.11 wird anfangs mit $\langle \emptyset, \emptyset, V, k \rangle$ aufgerufen. Die Rekursionstiefe ist offensichtlich $|V|$. Jeder

Algorithmus 6.11 Aufzählung von k-Schnitten

```
1:  function ENUMERATE(⟨V1, V2, V, k⟩)
2:      if V = ∅ then
3:          if |E(V1, V2)| ≤ k then
4:              print ⟨V1, V2⟩
5:          else
6:              return
7:      if V1 = ∅ or V2 = ∅ or MINCUT(V1, V2) ≤ k then
8:          wähle ein beliebiges v ∈ V                    ▷ deterministisch
9:          ENUMERATE(⟨V1 ∪ {v}, V2, V \ {v}, k⟩)
10:         ENUMERATE(⟨V1, V2 ∪ {v}, V \ {v}, k⟩)
11:     return
```

Rekursionsschritt ist in Polynomialzeit $|V|^{O(1)}$ durchführbar. Da in Zeile 7 mit einem MinCut-Algorithmus überprüft wird, ob es überhaupt noch einen k-Schnitt geben kann, verfolgen wir nur Pfade, die mindestens eine gültige Lösung garantieren. Die Polynomialzeitverzögerung beträgt daher $|V| \cdot |V|^{O(1)}$. □

Das nächste Resultat aus [FKP+11] benötigen wir für eine spätere Laufzeitbetrachtung.

Lemma 6.4. $\binom{a+b}{a} \leq 2^{2\sqrt{ab}}$ *für alle* $a, b \in \mathbb{N}$

Beweis. Auch der Beweis stammt aus [FKP+11]. Als erstes zeigen wir die folgende Ungleichung.

$$\binom{a+b}{a} \leq \frac{(a+b)^{a+b}}{a^a b^b} \tag{1}$$

Wir verwenden ohne Beweis, dass $f(n) := (1+\frac{1}{n})^n$ eine monoton wachsende Funktion ist. Dies impliziert

$$\begin{aligned}
& f(b) \leq f(a+b) \\
\Leftrightarrow \quad & \left(1+\frac{1}{b}\right)^b \leq \left(1+\frac{1}{a+b}\right)^{a+b} \\
\Leftrightarrow \quad & \left(\frac{b+1}{b}\right)^b \leq \left(\frac{a+b+1}{a+b}\right)^{a+b} \\
\Leftrightarrow \quad & \frac{(b+1)^b}{b^b} \leq \frac{(a+b+1)^{a+b}}{(a+b)^{a+b}} \\
\Leftrightarrow \quad & \frac{(a+b)^{a+b}}{b^b} \leq \frac{(a+b+1)^{a+b}}{(b+1)^b}.
\end{aligned} \tag{2}$$

Nun zeigen wir Ungleichung 1 durch vollständige Induktion über b und für ein allgemeines a. Für den Induktionsanfang setzen wir $b = 1$.

$$\begin{aligned}
\binom{a+1}{a} &\leq \frac{(a+1)^{a+1}}{a^a 1^1} \\
(a+1) &\leq \frac{(a+1)(a+1)^a}{a^a} \\
a^a &\leq (a+1)^a
\end{aligned}$$

Kommen wir nun zum Induktionsschritt.

$$\begin{aligned}
\binom{a+b+1}{a} &= \frac{a+b+1}{b+1}\binom{a+b}{a} \\
&\overset{1(I.V.)}{\leq} \frac{a+b+1}{b+1} \cdot \frac{(a+b)^{a+b}}{a^a b^b} \\
&\overset{2}{\leq} \frac{a+b+1}{b+1} \cdot \frac{(a+b+1)^{a+b}}{a^a (b+1)^b} \\
&= \frac{(a+b+1)^{a+b+1}}{a^a (b+1)^{b+1}}
\end{aligned}$$

Somit wäre Ungleichung 1 gezeigt. Nun können wir Lemma 6.4 beweisen. Da Lemma 6.4 für $a = 0$ oder $b = 0$ trivial ist, nehmen wir ohne Einschränkungen der Allgemeinheit an, dass $b \geq a > 0$ gilt und definieren $t := \sqrt{\frac{a}{b}}$.

$$\begin{aligned}
\binom{a+b}{a} &\overset{1}{\leq} \frac{(a+b)^{a+b}}{a^a b^b} \\
&= \frac{(a+b)^a}{a^a} \cdot \frac{(a+b)^b}{b^b} \\
&= \left(1+\frac{b}{a}\right)^a \left(1+\frac{a}{b}\right)^b \\
&= \left(\left(1+\frac{1}{t^2}\right)^{\frac{t}{1}} \left(1+\frac{t^2}{1}\right)^{\frac{1}{t}}\right)^{\sqrt{ab}} \\
&= \exp\left(\sqrt{ab}\ln\left(\left(1+\frac{1}{t^2}\right)^{\frac{t}{1}} \left(1+\frac{t^2}{1}\right)^{\frac{1}{t}}\right)\right) \\
&=: \exp\left(\sqrt{ab}\cdot g(t)\right) \\
&\leq \exp\left(\sqrt{ab}\cdot 2\ln 2\right) \\
&= 2^{2\sqrt{ab}}
\end{aligned}$$

Der Schritt, der nun fehlt, ist die Gültigkeit von $g(t) \leq 2\ln 2$ für $t \in]0, 1]$ zu zeigen. In [FKP+11] wurde dies bewiesen, indem gezeigt wurde, dass $g(1) = 2\ln(2)$ gilt und die Funktion auf dem Intervall monoton wachsend ist. □

Das nächste Lemma aus [FKP+11] benötigen wir um die Anzahl der k-Schnitte abzuschätzen.

Lemma 6.5. *Die Anzahl der k-Schnitte eines $\overset{p=}{}$Cluster-Graphen G ist durch $2^{8\sqrt{pk}}$ beschränkt, falls $p \leq 6k$ gilt.*

Beweis. Die Voraussetzung $p \leq 6k$ kann äquivalent zur Ungleichung

$$p = \sqrt{p \cdot p} \leq \sqrt{6pk} \tag{1}$$

umgeformt werden. Wir bezeichnen die Cliquen in G mit $C_1, \ldots, C_p$. Jede Partitionierung (V_1, V_2) und der damit verbundene k-Schnitt teilt eine Clique C_i in genau 2 Mengen $X_i \subseteq V_1$ und $Y_i \subseteq V_2$. Wir schließen insbesondere $X_i \neq \emptyset$ bzw. $Y_i \neq \emptyset$ nicht aus. Als erstes möchten wir die Kardinalitäten von X_i und Y_i festlegen. Diese bezeichnen wir mit x_i und y_i. Wir möchten alle möglichen Belegungen von x_i und y_i zählen. Für jedes Paar x_i und y_i wird jeweils festgelegt, ob x_i oder y_i größer gleich ist. Somit ergeben sich schon einmal $2^p \overset{1}{\leq} 2^{\sqrt{6pk}}$ Möglichkeiten. Da durch Festlegung von x_i oder y_i die jeweils andere Variable auch festgelegt wird, reicht es nun jeweils die kleinere von beiden zu belegen. Wir möchten zunächst untersuchen welche Belegungen in Frage kommen. Die Kosten eines Schnittes einer Clique C_i betragen $x_i \cdot y_i$, und da wir nur k-Schnitte betrachten, muss demnach die Ungleichung

$$\sum_{i=1}^{p} x_i y_i \leq k \tag{2}$$

gelten. Es gilt zudem die Cauchy-Schwarz-Ungleichung

$$\sum_{i=1}^{p} a_i b_i \leq \sqrt{\sum_{i=1}^{p} a_i^2} \cdot \sqrt{\sum_{i=1}^{p} b_i^2}, \tag{3}$$

die wir mit $a_i = 1$ und $b_i = \sqrt{x_i y_i}$ benutzen werden. Somit erhalten wir die Ungleichung

$$\sum_{i=1}^{p} \sqrt{x_i y_i} \overset{3}{\leq} \sqrt{p} \cdot \sqrt{\sum_{i=1}^{p} x_i y_i} \overset{2}{\leq} \sqrt{pk} \tag{4}$$

Aus $\min(x_i, y_i) \leq \sqrt{x_i y_i}$ und Ungleichung 4 folgt $\sum_{i=1}^{p} \min(x_i, y_i) \leq \sqrt{pk}$. Diese Abschätzung reicht uns um alle möglichen Belegungen der jeweils kleineren Variablen durchzuprobieren. Dies bedeutet, dass wir $\sum_{i=1}^{p} \min(x_i, y_i)$ in p Zahlen partitionieren müssen. Da wir den Wert von $\sum_{i=1}^{p} \min(x_i, y_i)$ nicht kennen, zerlegen wir die Zahl $\sqrt{pk}$ in $(p+1)$ Zahlen, wobei die $(p+1)$te Zahl die Differenz $\sqrt{pk} - \sum_{i=1}^{p} \min(x_i, y_i)$ darstellt, welche wir letztendlich verwerfen können. Die Anzahl der möglichen Partitionierungen von $\sqrt{pk}$ in $(p+1)$ Zahlen beträgt

$$\binom{\sqrt{pk}+p}{p} \leq \sum_{i=1}^{p} \binom{\sqrt{pk}+p}{i} = 2^{\sqrt{pk}+p} \overset{1}{\leq} 2^{\sqrt{pk}+\sqrt{6pk}}.$$

Bis hierher haben wir $2^{\sqrt{pk}+2\sqrt{6pk}} \leq 2^{6\sqrt{pk}}$ Möglichkeiten für die Festlegung der Kardinalitäten x_i und y_i. Als letztes werden die Möglichkeiten betrachtet eine Clique C_i in x_i Knoten aus V_1 und y_i Knoten aus V_2 zu zerlegen. Es gibt hierfür $\binom{x_i+y_i}{x_i}$ viele Möglichkeiten und diese können wegen Lemma 6.4 mit $2^{2\sqrt{x_i y_i}}$ abgeschätzt werden. Für die Festlegung aller Cliquen gibt es also $\prod_{i=1}^{p} 2^{2\sqrt{x_i y_i}} = 2^{2\sum_{i=1}^{p} \sqrt{x_i y_i}} \overset{4}{\leq} 2^{2\sqrt{pk}}$ viele Möglichkeiten. Letztendlich erhalten wir maximal $2^{6\sqrt{pk}} \cdot 2^{2\sqrt{pk}} = 2^{8\sqrt{pk}}$ mögliche k-Schnitte. □

Lemma 6.6. *Falls $\langle G, p, k\rangle \in {}^{=}\text{CE}$ und $p \leq 6k$, so ist die Anzahl der k-Schnitte in G maximal $2^{8\sqrt{2pk}}$.*

Beweis. Jeder k-Schnitt in G ist ein $2k$-Schnitt in G_F, weil maximal k Kanten hinzugefügt werden können. Somit lassen sich die k-Schnitte in G mit den $2k$-Schnitten in G_F abschätzen. Da G_F ein ${}^{p=}$Cluster-Graph ist und $p \leq 6k \leq 6 \cdot 2k$ gilt, gibt es nach Lemma 6.5 maximal $2^{8\sqrt{2pk}}$ viele $2k$-Schnitte in G_F und damit auch nicht mehr k-Schnitte in G. □

Nun sind wir in der Lage einen FPT-Algorithmus für unsere parametrisierten Probleme anzugeben.

Satz 6.7. k-${}^{=}$CE, k-${}^{=}$CD, k-${}^{=}$CC $\in$ FPT.

Beweis. Wir benutzen den FPT-Algorithmus 6.12, welcher zunächst k-${}^{=}$CE entscheidet.

Wegen Lemma 6.6 reicht es in Zeile 3 nur $2^{8\sqrt{2pk}}$ k-Schnitte von G zu berechnen. Jeder Knoten in H symbolisiert einen Schnitt in G. Bei jedem Kantenübergang in H wird ein weiterer Schnitt durchgeführt. Die Anzahl der bisherigen Schnitte wird in j und die Anzahl der bisherigen Kantenmodifikationen werden in l gespeichert. Deshalb liegt $\langle G, k, p\rangle$ genau dann in k-${}^{=}$CE, wenn in H ein entsprechender Pfad existiert.

Algorithmus 6.12 FPT-Entscheidungsalgorithmus für k-$^{=}$CE

```
1: function DECIDE(⟨G, k, p⟩)
2:     ⟨G, k, p⟩ := KERNAL(⟨G, k, p⟩)
3:     berechne die ersten 2^{8√(2pk)} k-Schnitte von G und bezeichne sie mit S
4:     V[H] := S × {1, ..., p} × {0, ..., k}
5:     E[H] := {{((A, B), j, l), ((A', B'), j + 1, l')} | A ⊊ A'
       und l' = l + |E(A, A' \ A)| + |Ē(A' \ A, A' \ A)|}
6:     H := (V[H], E[H])
7:     if es gibt einen Pfad von ((∅, V), 0, 0) nach ((V, ∅), p, l) für ein l ≤ k in H then
8:         accept
9:     else
10:        reject
```

Ein möglicher Pfad von $((\emptyset, V), 0, 0)$ nach $((V, \emptyset), p, l)$ ist in Abbildung 6.1 zu sehen. Wir können den Algorithmus auch derart modifizieren, dass er auch k-$^{=}$CD und k-$^{=}$CC löst. Da wir bei k-$^{=}$CD keine Kanten hinzufügen können, fordern wir für jeden Kantenübergang $\bar{E}(A' \setminus A, A' \setminus A) = \emptyset$. Bei k-$^{=}$CC hingegen dürfen wir keine Kanten entfernen und daher fordern wir für jeden Kantenübergang $E(A, A' \setminus A) = \emptyset$.

Kommen wir nun zur Laufzeitabschätzung. Zeile 3 kann bei Eingabe x wegen Lemma 6.3 in $O(|x|^{O(1)} \cdot 2^{8\sqrt{2pk}})$ berechnet werden. Es gilt $|V[H]| = 2^{8\sqrt{2pk}}(k+1)p \in 2^{O(\sqrt{pk})}$. Das Quadrat der Knotenanzahl verschwindet in der O-Notation und es gilt außerdem $|E[H]| \in 2^{O(\sqrt{pk})}$. Die Abfrage in Zeile 7 kann durch Breitensuche ebenfalls in $O(|H|) \subseteq 2^{O(\sqrt{pk})}$ gemacht werden. Der Algorithmus 6.12 hat somit eine Laufzeit von $O(|x|^{O(1)} \cdot 2^{O(\sqrt{pk})})$. Durch die Selbstreduktion aus Lemma 6.1 in Zeile 2 können wir p mit $6k$ abschätzen und somit lässt sich die Laufzeit auch mit $O(|x|^{O(1)} \cdot 2^{O(k)})$ abschätzen. □

Korollar 6.8. k-$^{\geq}$CE, k-$^{\geq}$CD, k-$^{\geq}$CC $\in$ FPT.

Beweis. Das Lemma 3.7 gilt analog für Cluster Deletion und Cluster Completion. Außerdem lässt sich k-$^{=}$CE, k-$^{=}$CD und k-$^{=}$CC in $O(|x|^{O(1)} \cdot 2^{O(k)})$ entscheiden und folglich lassen sich k-$^{\geq}$CE, k-$^{\geq}$CD und k-$^{\geq}$CC in $O(|x| \cdot |x|^{O(1)} \cdot 2^{O(k)}) = O(|x|^{O(1)} \cdot 2^{O(k)})$ entscheiden. □

Korollar 6.9. k-$^{\leq}$CE, k-$^{\leq}$CD, k-$^{\leq}$CC $\in$ FPT.

Beweis. Hier lässt sich analog wie im Korollar 6.8 mit dem Lemma 3.8 argumentieren. □

Für feste p hingegen lässt sich eine bessere Laufzeitabschätzung machen.

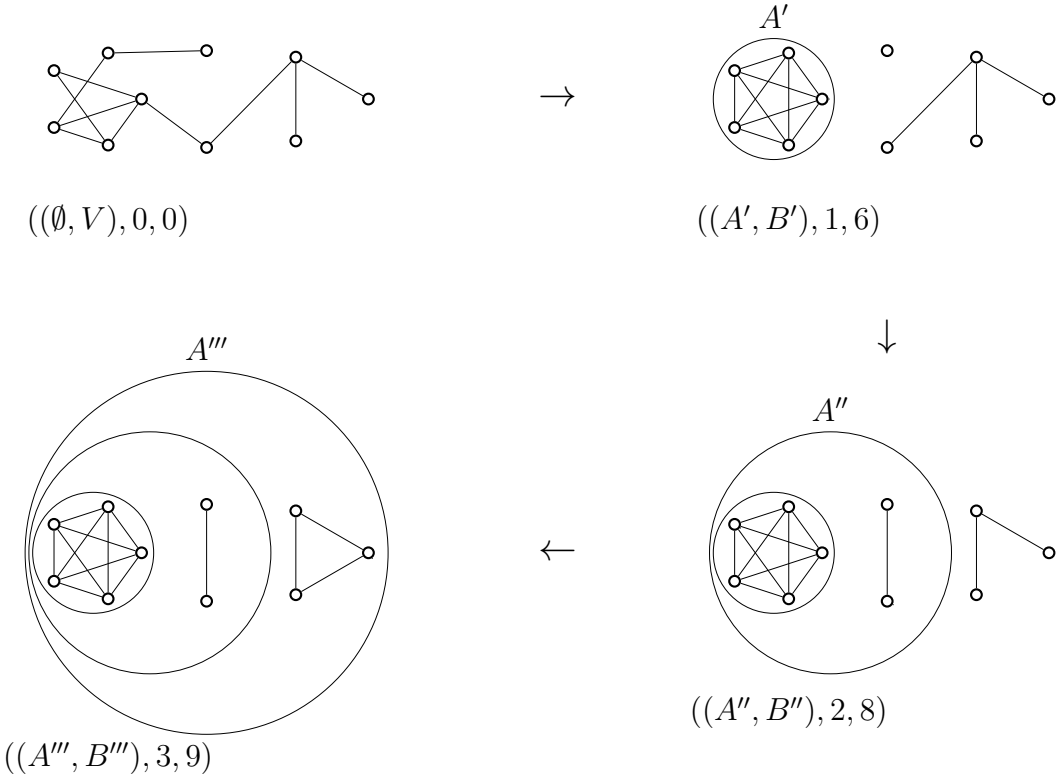
Satz 6.10. k-^{p}=CE, k-^{p}=CD, k-^{p}=CC $\in$ SUBEPT.

Beweis. Für feste p hat der Algorithmus 6.12 eine Laufzeit von $O(|x|^{O(1)} \cdot 2^{O(\sqrt{k})})$. □

Korollar 6.11. k-^{p}≥CE, k-^{p}≥CD, k-^{p}≥CC $\in$ SUBEPT.

Beweis. Mit Lemma 3.9 erhalten wir eine Laufzeit von $O(|x|^{O(1)} \cdot (|x|^{O(1)})^{O(1)} \cdot 2^{O(\sqrt{k})}) = O(|x|^{O(1)} \cdot 2^{O(\sqrt{k})})$. □

Abbildung 6.1: Ein möglicher Pfad in H



7 Graph-Modifizierung

7.1 Verbotene-Mengen-Charakterisierung

Die folgenden Resultate und Beweise sind aus [Cai96]. In Kapitel 2 wurden die Graphen-Modifikations-Probleme wie folgt definiert.

Problem Π-Graph Modification (Π-GM)

Eingabe	Ein Graph $G = (V, E)$ und $i, j, k \in \mathbb{N}$
Frage	Gibt es ein $F \subseteq V^2$ mit $\lvert F \cap E \rvert \leq i$, $\lvert F \cap \bar{E} \rvert \leq j$ und $\lvert F \rvert \leq k$, sodass $G_F \in \Pi$

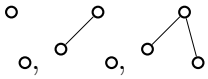
Die Menge Π ist dabei eine Menge von Zielgraphen. In Kapitel 3 bis 5 haben wir Scheiben des Problems betrachtet, indem wir für i, j und k nur bestimmte Kombinationen zugelassen haben und für Π verschiedene Mengen von Cluster-Graphen gewählt haben. In diesem Kapitel wollen wir einen Entscheidungsalgorithmus für alle Graphen-Modifikations-Probleme angeben, bei denen die Menge Π eine gewisse Eigenschaft vorweist.

Definition 7.1. *Eine Menge von Graphen* Π *heißt vererbbar, falls für jeden Graphen* $G \in \Pi$ *und für jeden Teilgraphen* $H \subseteq G$ *auch* $H \in \Pi$ *gilt.*

Beispiel 7.2 Die Menge der vollständigen Graphen ist vererbbar, da jeder Teilgraph eines vollständigen Graphen wieder ein vollständiger Graph ist.

Wir wollen nun Graphenmengen Π betrachten, die sich wie folgt charakterisieren lassen.

Definition 7.3. *Eine Graphenmenge* M *heißt Verbotene-Mengen-Charakterisierung der Menge* Π*, falls für alle Graphen* G *gilt:* $G \in \Pi$ *genau dann, wenn es keinen Teilgraphen von* G *in* M *gibt.*

Beispiel 7.4 Die Menge $M := \{$  $\}$ ist eine mögliche Verbotene-Mengen-Charakterisierung für die Menge der vollständigen Graphen.

Falls es sich bei M um eine endliche Menge handelt, so nennen wir M auch endliche Verbotene-Mengen-Charakterisierung.

Lemma 7.5. *Folgende Aussagen sind äquivalent:*

- *Es gibt eine Verbotene-Mengen-Charakterisierung von* Π.
- Π *ist vererbbar.*

Beweis. Zu zeigen sind zwei Richtungen.

„⇒“. Sei G ein Graph aus Π. Dann besitzt G nach Voraussetzung keinen Teilgraphen aus M. Daher besitzt auch jeder Teilgraph $H \subseteq G$ keinen Teilgraphen aus M und somit gilt $H \in \Pi$.

„⇐“. Wir definieren M als die Menge aller Graphen, die nicht in Π liegen. Wir zeigen nun, dass M eine Verbotene-Mengen-Charakterisierung von Π ist. Falls $G \in \Pi$, so liegt wegen der Vererbbarkeit von Π auch jeder Teilgraph $H \subseteq G$ in Π und somit liegt nach Definition H nicht in M. Umgekehrt falls $G \notin \Pi$, so liegt G nach Definition in M. Da G ein Teilgraph von sich selbst ist, gibt es also einen Teilgraphen von G, der in M liegt. □

Mit m möchten wir im folgenden die maximale Anzahl von Knoten in Graphen aus M bezeichnen. Damit ein Maximum existiert, ist M künftig endlich.

Lemma 7.6. *Falls Π eine endliche Verbotene-Mengen-Charakterisierung M besitzt, so kann in $O(|V[G]|^m)$ entschieden werden, ob ein gegebener Graph G in Π liegt.*

Beweis. Der Algorithmus 7.13 entscheidet ob G in Π liegt.

Algorithmus 7.13 P-Entscheidungsalgorithmus für Π

```
1: function DECIDE(⟨G⟩)
2:     for all Teilgraphen H ⊆ G mit |V[H]| ≤ m do
3:         if H ∈ M then
4:             reject
5:     accept
```

Der Graph G liegt genau dann in Π, wenn es keinen Teilgraphen von G in M gibt. Es muss jedoch nur für alle Teilgraphen mit maximal m Knoten geprüft werden, ob sie in M liegen, da M keine Graphen mit mehr als m Knoten enthält. Es gibt demnach maximal $|V[G]|^m$ Teilgraphen von G, die es zu prüfen gilt. Da M endlich und kein Teil der Eingabe ist, erledigt sich Zeile 3, also das Prüfen der Zugehörigkeit zu M, in konstanter Zeit. □

Lemma 7.7. *Für eine Menge Π mit endlicher Verbotener-Mengen-Charakterisierung M und für einen Graphen $G \notin \Pi$, lässt sich in $O(|V[G]|^{m+1})$ ein bezüglich Inklusion minimaler Teilgraph $H \subseteq G$ ermitteln, der ebenfalls nicht in Π liegt.*

Mit der Minimalität von H ist gemeint, dass jeder echte Teilgraph von H in Π liegt.

Beweis. Hierzu lässt sich der Algorithmus 7.14 verwenden. Der Algorithmus hat eine

Algorithmus 7.14 Finden von minimalen nicht-Π-Graphen

```
1: function FIND(⟨G⟩)                          ▷ G = (V, E)
2:     U := V
3:     while U ≠ ∅ do
4:         wähle ein v ∈ U                     ▷ deterministisch
5:         U := U \ {v}
6:         if G − v ∉ Π then
7:             G := G − v
8:     G* := G
9:     return G*
```

Laufzeit von $O(|V| \cdot |V|^m)$, da Zeile 6 nach Lemma 7.6 in $O(|V[G]|^m)$ läuft.

Durch Zeile 6 wird garantiert, dass G nicht in Π liegt. Der Eingabegraph liegt nach Voraussetzung ebenfalls nicht in Π. Somit gilt auch $G^* \notin \Pi$. Zu zeigen bleibt die Minimalität von G^*. Angenommen $G^* - u \notin \Pi$ für ein $u \in V[G^*]$, so muss es einen

Graphen G' geben für den in Zeile 6 geprüft wurde, ob $G' - u \notin \Pi$ gilt. Da $G^* \subseteq G'$ gelten muss, gilt auch $G^* - u \subseteq G' - u$. Da nach Annahme $G^* - u \notin \Pi$ gilt und Π nach Voraussetzung eine vererbbare Menge ist, muss auch $G' - u \notin \Pi$ gelten und somit landet der Algorithmus in Zeile 7, in der u aus G' entfernt wird. Somit ist $u \notin V[G^*]$ im Widerspruch zur Annahme. □

Als nächstes werden wir für diejenigen Graphen-Modifikations-Probleme einen Entscheidungsalgorithmus angeben, bei denen die Menge der Zielgraphen Π eine endliche Verbotene-Mengen-Charakterisierung besitzt.

Satz 7.8. *Für eine Menge von Graphen* Π *mit einer endlichen Verbotenen-Mengen-Charakterisierung* M*, lässt sich* Π-GM *bei Eingabe* $x = \langle G, i, j, k\rangle$ *in* $O(|x|^{m+1} \cdot m^{i+j+k+4})$ *entscheiden.*

Beweis. Es lässt sich der Algorithmus 7.15 verwenden, wobei wir ablehnen, falls der Algorithmus nicht akzeptiert.

Algorithmus 7.15 Entscheidungsalgorithmus für Π-GM

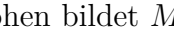
```
1:  function DECIDE(⟨G, i, j, k⟩)
2:      if i < 0 or j < 0 or k < 0 then
3:          return
4:      if G ∈ Π then
5:          accept
6:      finde einen minimaler Teilgraph H ⊆ G mit H ∉ Π
7:      for all e ∈ V[H]² do
8:          if e ∈ E[G] then
9:              DECIDE(⟨G_{e}, i − 1, j, k − 1⟩)
10:         else
11:             DECIDE(⟨G_{e}, i, j − 1, k − 1⟩)
12:     return
```

Der Algorithmus modifiziert schrittweise Kanten in G und verringert in jedem Schritt die entsprechenden Werte i, j und k abhängig davon, ob eine Kante entfernt oder hinzugefügt wurde. Der Algorithmus probiert nur einen Teil aller möglichen Modifikationen aus. Falls $\langle G, i, j, k\rangle \notin \Pi$-GM gilt, so akzeptiert der Algorithmus nicht. Falls jedoch $\langle G, i, j, k\rangle \in \Pi$-GM gilt, so gilt es zu zeigen, dass der Algorithmus akzeptiert. Da Π vererbbar ist, muss für jeden Teilgraphen $H \subseteq G$ auch $H \in \Pi$ gelten. Daher wird in Zeile 6 ein $H \subseteq G$ mit $H \notin \Pi$ ermittelt. Unabhängig von anderen Modifikationen in G muss dem Teilgraphen H demnach mindestens eine Kante entfernt oder hinzugefügt werden, da H als Teilgraph unzulässig ist. Aus diesem Grund verfolgt der Algorithmus mindestens einen akzeptierenden Pfad und arbeitet daher korrekt.

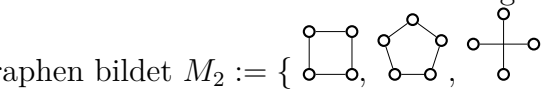
Kommen wir nun zur Laufzeitbetrachtung. Die Abfrage in Zeile 4 hat nach Lemma 7.6 eine Laufzeit von $O(|V[G]|^m)$. Zeile 6 benötigt nach Lemma 7.7 eine Zeit von $O(|V[G]|^{m+1})$. Es gilt $H \notin \Pi$ und somit gibt einen Teilgraphen $L \subseteq H$ mit $L \in M$. Außerdem ist H minimal und daher gilt für alle $H' \subsetneq H$, dass $H' \in \Pi$ und folglich auch $H' \notin M$. Somit muss $L = H$ gelten und folglich $H \in M$. Aus diesem Grund erhalten wir die Abschätzung $|V[H]| \leq m$ und es erfolgen daher maximal m^2 Schleifendurchläufe. Diese entsprechen dem Verzweigungsgrad. Die maximale Rekursionstiefe beträgt $\frac{i+j+k+4}{2}$. Die Gesamtlaufzeit beläuft sich damit auf $O((|V[G]|^m + |V[G]|^{m+1}) \cdot (m^2)^{\frac{i+j+k+4}{2}})$. □

Falls Π sich durch eine Verbotene-Mengen-Charakterisierung beschreiben lässt, so liegt das parametrisierte Problem ijk-Π-GM, bei dem $i + j + k$ der Parameter ist, in FPT, da sich i, j und k durch den Parameter abschätzen lassen und m nicht von der Eingabe abhängt. Nun werden wir uns einige Graphenmengen ansehen, die sich durch endliche verbotene Mengen charakterisieren lassen.

Definition 7.9. *Ein Graph G heißt Co-Graph, falls der Komplementgraph jedes zusammenhängenden Teilgraphen von G mit mindestens zwei Knoten nicht zusammenhängend ist.*

Beispiel 7.10 Eine endliche Verbotene-Mengen-Charakterisierung für die Menge aller Co-Graphen bildet $M_1 :=$ {  }.

Definition 7.11. *Ein Graph G heißt Split-Graph, falls dieser disjunkt in eine Clique und eine unabhängige Knotenmenge partitioniert werden kann.*

Beispiel 7.12 Eine endliche Verbotene-Mengen-Charakterisierung für die Menge aller Split-Graphen bildet $M_2 :=$ {  }.

Auch die $^{0\leq}$Cluster-Graphen aus Kapitel 3.1 können charakterisiert werden.

Beispiel 7.13 Eine endliche Verbotene-Mengen-Charakterisierung für die Menge aller $^{0\leq}$Cluster-Graphen bildet $M_3 :=$ {  }.

Aus Beispiel 7.13 und Satz 7.8 folgt, dass k-$^{0\leq}$CE, k-$^{0\leq}$CD und k-$^{0\leq}$CE in FPT liegen, was jedoch auch aus Korollar 6.9 ableitbar ist.

7.2 Bag-Charakterisierung

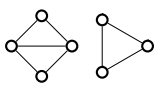
Die folgenden Resultate und Beweise sind aus [DM14]. In diesem Kapitel werden wir eine weitere Eigenschaft von Graphenmengen Π definieren. Auch hier werden wir einen Algorithmus angeben um solche Graphen-Modifikations-Probleme zu entscheiden, bei denen Π diese Eigenschaft erfüllt.

Für einen Knoten v möchten wir mit $N[v]$ alle Nachbarn von v einschließlich v selbst bezeichnen. Also $N[v] := \{u \in V \mid \{v, u\} \in E\} \cup \{v\}$.

Wir definieren eine Äquivalenzrelation auf Knoten.

Definition 7.14. *$u \sim v$ genau dann, wenn $N[u] = N[v]$. Wir sagen auch u und v sind Zwillinge.*

Nun möchten wir zu einem Graphen G seinen Bag-Graphen $H_{[G]}$ definieren. Die Äquivalenzklassen der Relation $\sim$ auf $V[G]$ bilden die Knotenmenge von $H_{[G]}$. Also $V[H_{[G]}] := \{[v]_\sim \mid v \in V\}$. Zwischen zwei Knoten $[u]_\sim$ und $[v]_\sim$ aus $H_{[G]}$ besteht genau dann eine Kante, falls $\{u, v\} \in E[G]$ liegt. Da alle Knoten der selben Äquivalenzklasse die selben Nachbarn haben, hängt diese Definition nicht von der Wahl des Repräsentanten ab. Formal gilt $E[H_{[G]}] := \{\{[u]_\sim, [v]_\sim\} \mid \{u, v\} \in E[G]\}$.

Beispiel 7.15 $G =$  $\Rightarrow H_{[G]} =$ 

Wir werden uns nun mit Graphenmengen Π beschäftigen, die sich wie folgt Charakterisieren lassen.

Definition 7.16. *Eine Graph B heißt Bag-Charakterisierung der Menge Π, falls für alle Graphen G gilt: $G \in \Pi$ genau dann, wenn $H_{[G]} \subseteq B$.*

Beispiel 7.17 Der Graph $B =$  ist eine Bag-Charakterisierung für die Menge aller $^{5\geq}$Cluster-Graphen.

Unter einer optimalen Lösung von Π-GM bei Eingabe $\langle G, i, j, k\rangle$ verstehen wir ein $G_F \in \Pi$, für das $|F|$ minimal ist. Wir definieren den Modifikationsgrad $D[v]$ eines Knotens v als die Anzahl der Kanten in F, die v enthalten. Also $D[v] = |\{e \in F \mid v \in e\}|$.

Beim Entscheiden der Sprache Π-GM werden wir um Laufzeit zu sparen unmittelbar nach der Eingabe von $\langle G, i, j, k\rangle$ zum Bag-Graphen $H_{[G]}$ übergehen. Hierfür benötigen wir das nächste Lemma.

Lemma 7.18. *Sei Π eine Menge mit einer Bag-Charakterisierung. Sei $\langle G, i, j, k\rangle \in \Pi$-GM. So gibt es eine optimale Lösung G_F, in der je zwei Zwillinge in G auch Zwillinge in G_F sind.*

Beweis. Sei B eine Bag-Charakterisierung von Π, sei G_F eine beliebige optimale Lösung und seien die Knoten u und v zwei Zwillinge in G. Wir wollen zeigen, dass es eine weitere optimale Lösung gibt, in der u und v auch Zwillinge in dieser Lösung sind. Ohne Einschränkungen der Allgemeinheit nehmen wir an, dass $D[v] \geq D[u]$ gilt. Wir betrachten nun den Bag-Graphen $H_{[G_F]}$. Da G_F eine Lösung ist, gilt $H_{[G_F]} \subseteq B$. Wir können eine weitere optimale Lösung konstruieren, indem wir den Knoten v aus seiner Äquivalenzklasse nehmen und diesen in der Äquivalenzklasse $[u]$ unterbringen. Auf diese Weise entsteht eine neue Modifikationsmenge $\tilde{F}$. Falls v der einzige Knoten in $[v]$ war, kann durch diesen Prozess die Äquivalenzklasse $[v]$ verloren gehen. Jedoch kann keine neue Äquivalenzklasse entstehen. Somit gilt $H_{[G_{\tilde{F}}]} \subseteq H_{[G_F]} \subseteq B$. Deshalb ist $G_{\tilde{F}}$ tatsächlich eine Lösung. Außerdem sind nun u und v Zwillinge in $G_{\tilde{F}}$. Wir müssen nun zeigen, dass $|\tilde{F}| \leq |F|$ gilt. Um von F nach $\tilde{F}$ zu gelangen, nehmen wir v aus seiner Äquivalenzklasse und entfernen alle Kanten $\{v, x\}$ aus F . Dadurch sparen wir $D[v]$ Modifikationen. Wir wollen nun v in der selben Äquivalenzklasse wie u unterbringen. Da u und v in G Zwillinge waren, müssen an v nur die Modifikationen getätigt werden, die auch an u getätigt wurden. Demnach fügen wir F alle Kanten $\{v, x\}$ hinzu, für die $\{u, x\} \in F$ gilt. Hierbei entstehen Kosten von $D[u]$ oder $D[u]-1$. Letzteres kann gelten, falls in $D[u]$ auch die Kante $\{u, v\}$ gezählt wird. Insgesamt werden Modifikationen in Höhe von mindestens $D[v] - D[u] \geq 0$ gespart und somit gilt $|\tilde{F}| \leq |F|$, was zu zeigen war. □

Aus dem nächsten Lemma erhalten wir eine hinreichende Bedingung, bei der ein Entscheidungsalgorithmus für Π-GM ablehnen darf.

Lemma 7.19. *Sei Π eine Menge mit einer Bag-Charakterisierung B und sei G ein Graph. Falls $H_{[G]}$ mehr als $(2k + |V[B]|)$ Knoten besitzt, so gilt $\langle G, i, j, k\rangle \notin \Pi$-GM.*

Beweis. Falls $\langle G, i, j, k\rangle \in \Pi$-GM, so gilt $H_{[G_F]} \subseteq B$. Somit gilt insbesondere auch $|V[H_{[G_F]}]| \leq |V[B]|$. Da F maximal k Kanten enthält, gibt es maximal $2k$ Knoten in G,

die eine angrenzende Kante aus F haben. Somit kann sich $H_{[G]}$ und $H_{[G_F]}$ um maximal $2k$ Knoten unterscheiden und es gilt $|V[H_{[G]}]| \leq 2k + |V[H_{[G_F]}]| \leq 2k + |V[B]|$. □

Nun können wir einen Entscheidungsalgorithmus für alle Graphen-Modifikations-Probleme angeben, bei denen die Menge Π eine Bag-Charakterisierung B besitzt.

Satz 7.20. *Für eine Menge von Graphen Π mit Bag-Charakterisierung B und $b := |V[B]|$ lässt sich* Π-GM *bei Eingabe $x = \langle G, i, j, k\rangle$ in $O(|x|^{O(1)} + b^{2k+b})$ entscheiden.*

Beweis. Es lässt sich der Algorithmus 7.16 verwenden.

Algorithmus 7.16 Entscheidungsalgorithmus für Π-GM

```
1:  function DECIDE(⟨G, i, j, k⟩)
2:      berechne H_[G]
3:      if |V[H_[G]]| > 2k + |V[B]| then
4:          reject
5:      for all v ∈ V[H_[G]] do
6:          weise nichtdeterministisch dem Knoten v einen Knoten aus B zu
7:      berechne die benötigten Kantenmodifikationen F
8:      if |F ∩ E| ≤ i and |F ∩ Ē| ≤ j and |F| ≤ k then
9:          accept
10:     else
11:         reject
```

Wegen Lemma 7.19 darf in Zeile 4 abgelehnt werden. B repräsentiert die Äquivalenzklassen von G. Da $\langle G, i, j, k\rangle \in \Pi$-GM genau dann, wenn $H_{[G_F]} \subseteq B$ gilt, muss für jeden Knoten aus G entschieden werden welcher Äquivalenzklasse beziehungsweise welchem Knoten aus B er angehören wird. Wegen Lemma 7.18 wissen wir jedoch ohne Einschränkungen der Allgemeinheit, dass Knoten der selben Äquivalenzklasse in G auch in der selben Äquivalenzklasse in G_F sind. Somit reicht es jedem Knoten aus $V[H_{[G]}]$ eine Äquivalenzklasse zuzuteilen. Schließlich ist noch zu prüfen ob F den Bedingungen aus Zeile 8 genügt.

Für die Laufzeitabschätzung nehmen wir an, dass die Laufzeit für Zeile 2 durch $|x|^{O(1)}$ beschränkt ist. Falls wir die Schleife deterministisch simulieren, so gibt es für jeden Knoten v aus $V[H_{[G]}]$ genau $|V[B]|$ mögliche Zuweisungen. Mit der Abschätzung aus Zeile 3 sind dies $|V[B]|^{|V[H_{[G]}]|} \leq |V[B]|^{2k+|V[B]|}$ deterministische Schritte und wir erhalten mit $|V[B]| = b$ eine Gesamtlaufzeit von $O(|x|^{O(1)} + b^{2k+b})$. □

Falls k der Parameter ist, so ist dies eine FPT-Laufzeit. In [DM14] wurde sogar ein Algorithmus mit SUBEPT-Laufzeit angegeben, auf den wir jedoch nicht eingehen werden.

Wir können Beispiel 7.17 verallgemeinern und einsehen, dass $B = (\{1, \ldots, p\}, \emptyset)$ eine Bag-Charakterisierung für die Menge der ${}^{p\geq}$Cluster-Graphen darstellt. In diesem Fall gilt $b = |V[B]| = p$. Somit liegt das parametrisierte Problem pk-${}^{\geq}$CE, bei dem $p + k$ der Parameter ist, in FPT, da sich p und k durch den Parameter abschätzen lassen. Diese Beobachtung folgt allerdings auch unmittelbar aus Korollar 6.8.

7.3 Uneingeschränkte Graph-Modifizierung

Falls wir die Menge Π durch keine der betrachteten Charakterisierungen beschreiben können, so können wir auch hier eine obere Schranke für die Laufzeit angeben.

Satz 7.21. *Sei* Π *eine Menge von Graphen, für die in* $|V[G]|^{O(1)}$ *entschieden werden kann, ob* $G \in \Pi$ *für einen Graphen* G *gilt.* Π-GM *lässt sich bei Eingabe* $x = \langle G, i, j, k\rangle$ *in* $O(|x|^{O(i+j+k)})$ *entscheiden.*

Beweis. Wir verwenden hierfür den Algorithmus 7.17, welcher ähnlich wie der Algorithmus 7.15 arbeitet. Falls der Algorithmus nicht akzeptiert, so soll abgelehnt werden.

Algorithmus 7.17 Entscheidungsalgorithmus für Π-GM

```
1:  function DECIDE(⟨G, i, j, k⟩)
2:      if i < 0 or j < 0 or k < 0 then
3:          return
4:      if G ∈ Π then
5:          accept
6:      for all e ∈ V[G]² do
7:          if e ∈ E[G] then
8:              DECIDE(⟨G_{e}, i − 1, j, k − 1⟩)
9:          else
10:             DECIDE(⟨G_{e}, i, j − 1, k − 1⟩)
11:     return
```

Der Algorithmus ist korrekt, da jede mögliche Modifikation an G durchprobiert wird. Nach Voraussetzung lässt sich Zeile 4 in Polynomialzeit berechnen. Dies ist durchaus denkbar, da die Menge Π nicht von der Eingabe abhängt. Die Schleife in Zeile 6 wird ebenfalls polynomiell in der Eingabelänge durchlaufen. Somit ist die Laufzeit eines Rekursionsschrittes durch ein Polynom beschränkt. Die maximale Rekursionstiefe beträgt auch hier $\frac{i+j+k+4}{2}$. Somit hat der Algorithmus eine Laufzeit von $O(|x|^{O(i+j+k)})$. □

8 Fazit

In meiner Arbeit haben wir das Graphen-Modifikations-Problem untersucht. Dazu haben wir uns speziell mit den Cluster-Graphen-Modifikations-Problemen beschäftigt. Außerdem haben wir die Cluster-Graphen hinsichtlich der Anzahl der Cliquen klassifiziert. Hierzu haben wir jene Cluster-Graphen betrachtet, die aus mindestens, genau und maximal p Cliquen bestehen. Auf diese Weise haben wir die Mengen der ${}^{p\leq}$Cluster-Graphen, ${}^{p=}$Cluster-Graphen und ${}^{p\geq}$Cluster-Graphen definiert. Wir haben gesehen, dass Cluster Editing für die Zielgraphenmenge der ${}^{p\leq}$Cluster-Graphen, der ${}^{p=}$Cluster-Graphen oder der ${}^{p\geq}$Cluster-Graphen NP-vollständig ist.

Wenn wir uns auf das Entfernen von Kanten einschränken, so sind die hierdurch definierten Cluster Deletion-Probleme auch für jede der drei Klassen von Cluster-Graphen NP-vollständig.

Falls wir lediglich das Hinzufügen von Kanten erlauben, so sprechen wir von Cluster Completion, welches sich als einfacher herausstellte. Die Cluster Completion-Probleme lassen sich für jede der drei Klassen von Cluster-Graphen in Polynomialzeit lösen. Für die Zielgraphenmenge der ${}^{p\leq}$Cluster-Graphen, lässt sich das Problem auch dann noch in Polynomialzeit lösen, falls p ein Teil der Eingabe ist.

Um herauszufinden welche Faktoren Cluster Editing und Cluster Deletion so schwer machen, haben wir diese parametrisiert betrachtet. k-${}^{=}$Cluster Editing und k-${}^{=}$Cluster Deletion können wir in FPT-Laufzeit lösen, solange p ein Teil der Eingabe und k der Parameter ist. Falls wir jedoch p als Parameter wählen und k ein Teil der Eingabe ist, so lässt sich das Problem für den Fall P $\neq$ NP nicht in XP-Laufzeit und folglich auch nicht in FPT-Laufzeit lösen. Hieraus lässt sich feststellen, dass große k die Laufzeit von ${}^{=}$Cluster Editing und ${}^{=}$Cluster Deletion hochtreiben. Der Wert von p spielt eine weniger große Rolle.

In ${}^{p\geq}$Cluster-Graphen darf es maximal p Cliquen geben. Die Anzahl der Cliquen muss daher entweder genau 1 oder genau 2 ... oder genau p entsprechen. Aus diesem Grund kann ${}^{\geq}$Cluster Editing und ${}^{\geq}$Cluster Deletion durch mehrfache Anwendung eines ${}^{=}$CE- bzw. ${}^{=}$CD-Algorithmus entschieden werden. Somit liegen für den Parameter k auch diese Probleme in FPT. Analog gilt dies auch für ${}^{\leq}$Cluster Editing und ${}^{\leq}$Cluster Deletion

Am Ende der Arbeit haben wir das allgemeine Graphen-Modifikations-Problem erneut aufgegriffen. Da Π im allgemeinen unendlich groß sein kann, müssen sich Gedanken gemacht werden, wie diese unendliche Menge durch eine endliche Charakterisierung beschrieben werden kann. Ein möglicher Ansatz bildet dabei die Verbotene-Mengen-Charakterisierung. Hierbei wird Π durch endlich viele Teilgraphen beschrieben, welche nicht in Graphen aus Π enthalten sein dürfen. Falls eine solche Charakterisierung möglich ist, so ist der Vorteil, dass nur lokal nach verbotenen Teilgraphen gesucht werden muss und diese unabhängig von der Gestalt des restlichen Graphen eliminiert werden können. Im Fall einer solchen Charakterisierung können wir das Graphen-Modifikations-Problem daher in FPT-Laufzeit lösen, falls wir k als Parameter wählen.

Eine weitere Möglichkeit, die Menge Π zu beschreiben, ergab sich durch eine Bag-Charakterisierung. Beispielsweise ist die Menge der ${}^{p=}$Cluster-Graphen unendlich groß. Um dies in den Griff zu bekommen, sind wir zu den Äquivalenzklassen der Knotenmen-

ge übergegangen und haben den Bag-Graphen betrachtet. Dieser hat die Eigenschaft, dass Knoten der selben Cliquen zu einer Äquivalenzklasse zusammengefallen sind und somit durch einen Knoten im Bag-Graphen repräsentiert werden können. Somit zerfällt die Menge der ${}^{p=}$Cluster-Graphen zu genau einem Bag-Graphen, nämlich zu dem Graphen mit p Knoten und leerer Kantenmenge. Der Vorteil an einer solchen Charakterisierung liegt darin, dass wir vor Beginn der Modifikationen schon zum Bag-Graphen des Eingabegraphen übergehen können, da sich herausstellte, dass Knoten der selben Äquivalenzklasse des Eingabegraphen auch in der selben Äquivalenzklasse des Zielgraphen sein müssen. Falls eine solche Bag-Charakterisierung möglich ist, so können wir das Graphen-Modifikations-Problem mit dem Parameter k in FPT-Laufzeit lösen.

9 Ausblick

Für die meisten Probleme konnten wir einen Polynomialzeitalgorithmus angeben oder die NP-Vollständigkeit der Sprache beweisen. Für die parametrisierten Varianten konnten wir meist entweder mindestens einen FPT-Algorithmus angeben oder argumentieren, dass die Suche nach einem solchen Algorithmus im Rahmen dieser Arbeit nicht machbar wäre, da sie nicht weniger aufwändig als das P-NP-Problem ist. Für $^{=}$Cluster Completion und $^{\geq}$Cluster Completion hingegen konnten wir keine Klassifizierung vornehmen. In Satz 5.2 konnten wir einen nicht trivialen Algorithmus für $^{p=}$Cluster Completion angeben, der bei Eingabe x in $O(|x|^p)$ arbeitet. Solange p jedoch ein Teil der Eingabe ist, könnten wir zwar $p \leq |x|$ durch Fallunterscheidung garantieren, jedoch hätten wir dadurch gerade einmal eine Laufzeit von $O(|x|^{|x|})$ erreicht. Dies ist keinesfalls polynomiell. Polynomielle Überprüfbarkeit hingegen ist gewährleistet. Die Modifikationsmenge F ist ein mögliches Zertifikat, da wir in Polynomialzeit G_F berechnen können und anschließend überprüfen können, ob G_F ein $^{p=}$Cluster-Graph ist. Somit liegt $^{=}$CC in NP. Es wäre denkbar, dass $^{=}$CC auch NP-vollständig ist, doch bisher ist kein Beweis dafür gelungen. Die gleichen Überlegungen gelten auch für $^{\geq}$CC.

Ein weiterer Aspekt, der in dieser Arbeit nicht beantwortet wurde, ist inwiefern die Laufzeiten unserer parametrisierten Probleme optimal sind oder ob sich die Sprachen effizienter entscheiden lassen. Für solche Fragestellungen wurde in [IP99] die exponential time hypothesis (ETH) aufgestellt, die bis heute nicht bewiesen werden konnte. Die Hypothese besagt, dass 3SAT nicht in $O(2^{o(n)})$ lösbar ist, wobei n die Anzahl der Variablen darstellt. k-$^{p\geq}$CE haben wir in Kapitel 6 in $O(|x|^{O(1)} \cdot 2^{O(\sqrt{k})})$ entschieden. In [FKP$^+$11] wurde gezeigt, dass für $p \geq 6$ unter Annahme der exponential time hypothesis kein Algorithmus mit einer Laufzeit von $O(|x|^{O(1)} \cdot 2^{o(\sqrt{k})})$ oder $O(2^{o(|x|)})$ existieren kann. Des Weiteren haben wir in Kapitel 6 das parametrisierte Problem k-$^{\geq}$CE in $O(|x|^{O(1)} \cdot 2^{O(k)})$ gelöst. Und auch hier wurde in [FKP$^+$11] gezeigt, dass es unter Annahme der ETH keinen Algorithmus mit einer Laufzeit von $O(|x|^{O(1)} \cdot 2^{o(k)})$ geben kann. Die gleichen Schranken lassen sich auch für k-$^{p=}$CE und k-$^{=}$CE angeben, da wir mit einer Mehrfachanwendung der Algorithmen auch k-$^{p\geq}$CE und k-$^{\geq}$CE in der gleichen Zeitkomplexität lösen können.

Trotzdem bleiben einige Fragen offen. Unter anderem wäre es interessant zu wissen, ob k-$^{=}$CC schneller als in FPT-Laufzeit lösbar ist, da die durch die ETH gegebene echte untere Schranke von $O(|x|^{O(1)} \cdot 2^{o(k)})$ nicht zwangsläufig für Cluster Completion gelten muss. Dies wäre denkbar, da Cluster Completion für den Fall P $\neq$ NP nicht schon für feste p NP-vollständig und somit intuitiv einfacher als Cluster Editing ist.

Für das allgmeine Graphen-Modifikations-Problem ist außerdem nicht geklärt, inwiefern weitere Charakterisierungen der Zielgraphenmenge Π existieren, welche eine bessere Laufzeit als die des uneingeschränkten Graphen-Modifikations-Problems garantieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[Cai96] Cai, Leizhen: Fixed-parameter tractability of graph modification problems for hereditary properties. In: *Information Processing Letters* Bd. 58. Elsevier, 1996. – ISSN 0020–0190, S. 171–178

[DM14] Damaschke, Peter ; Mogren, Olof: *Lecture Notes in Computer Science*. Bd. 8344: *Editing the Simplest Graphs*. Springer, 2014. – 249–260 S. – ISBN 978–3–319–04656–3

[FG06] Flum, J. ; Grohe, M.: *Parameterized Complexity Theory*. Springer, 2006 (Texts in Theoretical Computer Science. An EATCS Series). – ISBN 978–3–540–29953–0

[FKP+11] Fomin, Fedor V. ; Kratsch, Stefan ; Pilipczuk, Marcin ; Pilipczuk, Michal ; Villanger, Yngve: Subexponential fixed-parameter tractability of cluster editing. In: *The Computing Research Repository* (2011)

[IP99] Impagliazzo, Russell ; Paturi, Ramamohan: On the Complexity of k-SAT. In: *2012 IEEE 27th Conference on Computational Complexity*, 1999. – ISBN 0–7695–0075–7, S. 237–240

[Lov73] Lovasz, L.: Covering and coloring of hypergraphs. In: *Proceedings of the Fourth Southeastern Conference on Combinatorics Graph Theory and Computing*, Utilitas Mathematica, 1973, S. 3–12

[SST04] Shamir, Ron ; Sharan, Roded ; Tsur, Dekel: Cluster Graph Modification Problems. In: *Discrete Applied Mathematics* 144 (2004), S. 173–182