Diskrete Optimierung

Prof. Marc Hellmuth

Ernst-Moritz-Arndt Universität
Institut für Mathematik und Informatik
WS 2015/2016

erstellt mit freundlicher Unterstützung von Kristina Wicke

Letzte Aktualisierung 19. April 2016

Inhaltsverzeichnis

0 Einführung **5**

0.1 Was ist diskrete Optimierung? 5

1 Kurze Einführung in die Graphentheorie **9**

2 Mögliche Themen/ Betrachtungsweisen/ Problemstellungen der diskreten Optimierung **11**

2.1 Komplexitätsbetrachtungen 12

2.2 Formulierung als ganzzahliges lineares Programm (ILP) 13

2.3 Approximative Lösungen 13

2.4 Betrachtung von Spezialfällen 14

2.5 Parametrisierte Algorithmen 15

2.6 Heuristiken (ohne Gütegarantie) 17

3 Komplexitätsbetrachtungen **21**

3.1 Einführung 21

3.2 Komplexitätsklassen 22

3.3 Prinzip der Reduktion und NP-Vollständigkeit 23

3.3.1 Polynomialzeitreduktion 23

3.3.2 NP-Vollständigkeit 24

3.4 Das SAT-Problem 25

3.5 Weitere NP-Vollständige Probleme 27

3.5.1 Cliquen-Problem 27

3.5.2 Vertex-Cover-Problem 28

4 Kürzeste Wege in Graphen **31**

4.1 NP-schwere Kürzeste-Wege-Probleme 32

4.2 „Einfache“ Kürzeste-Wege-Probleme 38

4.2.1 Kürzeste Wege von einer Quelle 38

4.2.2 Kürzeste Wege zwischen allen Knotenpaaren 43

5 Matroide **47**

5.1 Das Minimum-Spanning-Tree Problem 47

5.2 Theorie der Matroide 51

5.2.1 Einführung 51

5.2.2 Andere Matroidaxiome 55
5.2.3 Der GREEDY-Algorithmus 64
5.2.4 Der Schnitt von Matroiden 72

6 Approximationsalgorithmen 81
6.1 Approximation mit absoluter Güte 81
6.1.1 Knotenfärbung 81
6.2 Nichtapproximierbarkeit mit absoluter Güte 88
6.3 Approximation mit relativer Güte 91
6.3.1 Minimum-Vertex-Cover-Problem 91
6.3.2 Das MAX-CUT-Problem 92
6.4 Nichtapproximierbarkeit mit relativer Güte 94
6.5 Approximationsschemata 96
6.5.1 Max-Simple-Knapsack 98

7 Fest-Parameter-berechenbare Algorithmen (fixed-parameter-tractable; FPT) 103
7.1 Reduzierung auf den Problemkern 104
7.1.1 Beispiel: Vertex Cover 104
7.2 Tiefenbeschränkte Suchbäume 109
7.2.1 Einführung 109
7.2.2 Beispiel – Vertex Cover 111
7.3 Vererbbare Grapheigenschaften und Graphmodifiifikation 112
7.3.1 Einführung 112
7.3.2 Beispiel: Cographen 113
7.3.3 Σ-Graphmodifikation 116

0 Einführung

Literaturhinweise:

- „Kombinatorische Optimierung – Theorie und Algorithmen“, Korte & Vygen, Springer Verlag
- „Combinatorial Optimization – algorithms and complexity“, Papadimitriou & Steiglitz, Dover

0.1 Was ist diskrete Optimierung?

Ein Beispiel der “klassischen Optimierung” ist folgendes: Finde Minima bzw. Maxima einer (stetigen) Funktion $f : \mathbb{R} \to \mathbb{R} \Rightarrow$ Lösung des Optimierungsproblems kann z.B. mittels Kurvendiskussion erfolgen, wobei alle $x \in \mathbb{R}$ als Lösung zulässig sind. Im Gegensatz dazu ist der Definitionsbereich in der diskreten Optimierung eine endliche (oder abzählbar unendliche Menge), also diskret.

Definition (Instanz)**.** Eine *Instanz* eines diskreten Optimierungsproblems ist ein Paar (X, f), bestehend aus einer endlichen (oder abzählbar unendlichen Menge) X und einer Abbildung $f : X \to \mathbb{R}$, die jedem Element aus X einen Zielfunktionswert zuweist. Wir suchen $x^* \in X$ sodass $f(x^*) = \min_{x \in X} f(x)$.

Anmerkungen.

1. Ein Diskretes Optimierungsproblem definiert sich über die Familie all seiner Instanzen.
2. Ein Minimum existiert, aber x^* muss nicht eindeutig sein.
3. Analog lassen sich diskrete Maximierungsprobleme definieren, z.B. durch Betrachtung von $-f(x)$.

Beispiele

Problem 0.1. JOB-DRILLING (Bohrpunktproblem)
Wir wollen möglichst schnell an vorgeschriebenen Stellen bohren, d.h. wir suchen einen optimalen Weg zwischen den Bohrpunkten.

Daten:
Menge an Punkten $p_1, \ldots, p_n \in \mathbb{R}^2$ (Bohrlöcher)

Ziel:
Bestimme eine Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, sodass

$$f(\pi) = \sum_{i=1}^{n-1} \| p_{\pi(i)} - p_{\pi(i+1)} \| \xrightarrow{!} \min .$$

Problem 0.2. TRAVELLING SALESMAN PROBLEM (Problem des Handlungsreisenden)
Wir suchen eine kürzeste Rundreise durch n Städte.

Daten:
Städte $1, \ldots, n$ und Distanzen $w_{ij} = w_{ji} > 0$ zwischen Städten i, j mit $i \neq j$.

Ziel:
Bestimme eine Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, sodass

$$f(\pi) = \sum_{i=1}^{n-1} w_{\pi(i)} w_{\pi(i+1)} + w_{\pi(n)} w_{\pi(1)} \xrightarrow{!} \min .$$

Problem 0.3. RUCKSACK PROBLEM (Knapsack Problem)
Gegeben n Gegenstände mit unterschiedlichem Wert und Volumen, wollen wir einen Rucksack optimal packen.

Daten:
Gegenstände $1, \ldots, n$ mit Volumen w_i und Wert/Preis $b_i, i = 1, \ldots, n$ sowie Gewichtsschranke C (Kapazität des Rucksacks)

Ziel:
Finde Teilmenge $S \subseteq \{1, \ldots, n\}$, sodass

$$\sum_{s \in S} w_s \leq C \text{ und } f(S) = \sum_{s \in S} b_s \xrightarrow{!} \max .$$

Lineare Programme Ein weiteres wichtiges Beispiel für Optimierungsprobleme sind sogenannte *Lineare Programme* (LP) der Form

$$\max_{x \in X} \{ c^\top x | \underbrace{Ax \leq b, x \geq 0, x \in X}_{B} \}.$$

B wird dabei auch *zulässiger Bereich* genannt. Aus der linearen Optimierung wissen wir, dass die Optimallösung in den Ecken des konvexen Polyeders B liegt. In der linearen Optimierung ist B i.d.R. überabzählbar, aber es gibt effiziente Methoden, wie den Simplex-Algorithmus, um die Optima zu bestimmen. Im Unterschied dazu ist B in der diskreten Optimierung endlich oder abzählbar, z.B. falls $X = \mathbb{Z}^2$.
Interessanterweise lassen sich nahezu alle diskreten Optimierungsprobleme als Lineares Programm schreiben. Dies erlaubt zwar nicht, die Standardmethoden der linearen Optimierung

zur Lösung zu nutzen (da diskreter Zustandsraum), aber es gibt effiziente „Solver“, die ohne großen Aufwand verwendet werden können. Anhand des Problems 0.3 (Rucksack-Problem) verdeutlichen wir, wie sich ein diskretes Optimierungsproblem in ein LP umschreiben lässt.

Beispiel (Formulierung von Problem 0.3 als ILP)**.** Definiere Variablen $x_s \in \{0,1\} \forall s = 1, \ldots, n$ mit $x_s = 1 \Leftrightarrow s \in S$ (d.h. $x_s = 1$, wenn s in den Rucksack kommt).
Ziel: Finde Belegung der x_s sodass

$$\max \sum_{s=1}^{n} x_s b_s \text{ und } \sum_{s=1}^{n} x_s w_s \leq C$$

1 Kurze Einführung in die Graphentheorie

Motivation Jede Binärrelation $R \subseteq V \times V$ lässt sich als Graph $G = (V, E)$ darstellen, wobei V die Knotenmenge und E die Kantenmenge bezeichnet. Dabei bilden $i, j \in V$ eine Kante $(i, j) \in E \Leftrightarrow (i, j) \in R$. Da sich auch viele diskrete Optimierungsprobleme als Graph modellieren lassen, sollen im Folgenden die ersten wichtigen Begriffe der Graphentheorie eingeführt werden. *Weitere Definitionen werden dann in den folgenden Kapiteln gegeben, wenn sie benötigt werden.*

Definition (Graph)**.** Ein *Graph* $G = (V, E)$ ist ein Paar bestehend aus einer Menge $V \neq \emptyset$ von Knoten und einer Menge E von Kanten. Falls

- $E \subseteq \binom{V}{2}$ heißt G *ungerichtet.*
- $E \subseteq V \times V$ heißt G *gerichtet.*

Wenn nicht explizit erwähnt, bedeutet *Graph* im Folgenden ungerichtet und *Di-Graph* gerichtet.
Wir werden Kanten $\{u, v\} \in E$ von ungerichteten Graphen als (u, v) schreiben.

Definition (adjazent)**.** Sei $G = (V, E)$. Zwei Knoten u, v heißen *adjazent* $\Leftrightarrow (uv) \in E$ oder $(vu) \in E$.

Definition (inzident)**.** Sei $G = (V, E)$. Ein Knoten u ist *inzident* zu einer Kante $e \in E \Leftrightarrow \exists e = (xu)$ oder $e = (ux)$ in E.

Definition (Grad)**.** Sei $G = (V, E)$. Der *Grad* eines Knotens v ist wie folgt definiert:

- G ungerichtet: $deg(v) = |\{u \in V : \{uv\} \in E\}|$
- G gerichtet:
  - $deg_{out}(v) = |\{u \in V : (vu) \in E\}|$
  - $deg_{in}(v) = |\{u \in V : (uv) \in E\}|$
  - $deg(v) = deg_{out}(v) + deg_{in}(v)$

Der *Maximalgrad* eines Graphen wird mit $\Delta G = \max_{v \in V}\{deg(v)\}$ bezeichnet.

Definition (isomorph)**.** Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ heißen *isomorph*, wenn es eine bijektive Abbildung $\varphi : V_1 \to V_2$ gibt, sodass $(uv) \in E_1 \Leftrightarrow (\varphi(u)\varphi(v)) \in E_2$.

Definition (Teilgraph)**.** $H = (W, F)$ ist *Teilgraph* von $G = (V, E)$, wenn $W \subseteq V$ und $F \subseteq E$. Wir schreiben dafür $H \subseteq G$. Wenn alle Paare von Knoten aus $H \subseteq G$, die in G adjazent sind, auch in H adjazent sind, so heißt H *induzierter Teilgraph* von G.

Definition (Vollständiger Graph)**.** Ein ungerichteter Graph $G = (V, E)$ ist ein *vollständiger Graph*, wenn $E = \binom{V}{2}$, d.h. $\forall u, v \in V, u \neq v, \exists (uv) \in E$. Wir schreiben dafür $K_{|V|}$.

Definition (bipartit)**.** Ein Graph $G = (V, E)$ heißt *bipartit*, wenn es eine Partition $V_1 \cup V_2$ von V gibt, sodass für alle Kanten $(uv) \in E$ gilt: $u \in V_i, v \in V_j, i \neq j$.

Definition (Komplement)**.** Sei $G = (V, E)$ ein (Di-)Graph. Dann ist das *Komplement* von G der Graph $\overline{G} = (V, \overline{E})$ mit $\overline{E} = \{(uv) | u, v \in V, (uv) \notin E, u \neq v\}$.

Übungsaufgaben

Zeigen oder widerlegen Sie folgende Aussagen:

- Für einen Graphen $G = (V, E)$ gilt:

$$\sum_{v \in V} deg(v) = 2 \cdot |E|.$$

- Jeder Graph $G = (V, E)$ hat eine gerade Anzahl an Knoten mit ungeradem Grad.
- Sei $G = (V, E)$ und $|V| \geq 2$. Dann hat G immer zwei Knoten mit gleichem Grad.
- Sei $G = (V, E)$ bipartit. Dann ist $\overline{G} = (V, \overline{E})$ bipartit.
- Ein Graph $G = (V, E)$ ist bipartit $\Leftrightarrow$ G enthält keine "Kreise" ungerader Länge.

2 Mögliche Themen/ Betrachtungsweisen/ Problemstellungen der diskreten Optimierung

Anhand des sogenannten *Vertex-Cover-Problems* (VCP) verschaffen wir uns nun einen Überblick über die Themengebiete der diskreten Optimierung.

Definition (Vertex-Cover-Problem (VCP))**.** Sei $G = (V, E)$ ein ungerichteter Graph. Gesucht ist die kleinste natürliche Zahl k, sodass eine Teilmenge $C \subseteq V$ existiert mit

(i) $|C| = k$

(ii) $C \cap e \neq \emptyset \, \forall e \in E$

$C \subseteq V$, welches (ii) erfüllt, heißt *vertex cover*. Ein vertex cover mit minimaler Kardinalität heißt *minimum vertex cover*. Das Problem, für einen gegebenen Graphen ein minimum vertex cover zu finden, heißt *Minimum-Vertex-Cover Problem*.

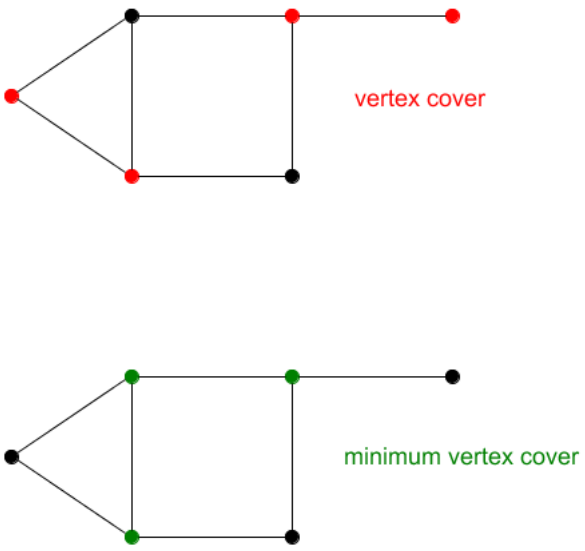

Abbildung 2.1: Beispiel eines *vertex covers* und eines *minimum vertex covers*

2.1 Komplexitätsbetrachtungen

(a) Holzhammermethode: Enumerieren aller möglichen Lösungen und Auswählen der besten Lösung.

Beispiel (VCP)**.** Um ein *minimum vertex cover* zu finden, teste für alle $C \subseteq V$, ob Bedingung (ii) des VCP erfüllt und wähle unter diesen das *vertex cover* minimaler Kardinalität. Im worst case müssten $2^{|V|}$ mögliche Lösungen überprüft werden. Die Holzhammermethode ist klarerweise eine schlechte Methode, was auch die folgenden theoretischen Rechenzeiten für mittelgroße Instanzen verdeutlichen (Annahme: 1 Rechenschritt, dh. Finden und Testen einer Teilmenge von $\mathbb{P}(V) \sim 1$ ns):

Tabelle 2.1: Rechenzeit der Holzhammermethode für VCP

$\lvert V\rvert$	10	20	30	40	50	60	100
Laufzeit	1 μs	1 ms	1 s	1100 s	13 Tage	37 Jahre	$4 \cdot 10^{19}$ Jahre

(b) Polynomialzeit-Algorithmen

Definition (Komplexität)**.** Ein Algorithmus hat die *Komplexität/ Laufzeit* $\mathcal{O}(f(n))$ für $f : \mathbb{N} \to \mathbb{N}$, falls es eine Konstante $c > 0$ gibt, sodass der Algorithmus für jeden Input der Länge n nach maximal $c \cdot f(n)$ Schritten terminiert.

Definition (Polynomialzeit-Algorithmus)**.** Ein Algorithmus heißt *Polynomialzeit-Algorithmus*, wenn es ein Polynom $f(n)$ gibt, sodass der Algorithmus die Laufzeit $\mathcal{O}(f(n))$ hat.

Anmerkungen.

- Das Polynom f kann immer in der Form $f(n) = n^k$ gewählt werden, d.h. wir erhalten $\mathcal{O}(n^k)$.
- Ein Polynomialzeit-Algorithmus ist nicht automatisch ein effizienter und schneller Algorithmus. Dies gilt in der Regel nur für $\mathcal{O}(n)$ oder $\mathcal{O}(\log n)$

Beispiel (VCP)**.** Bisher hat niemand einen Polynomialzeit-Algorithmus für das VCP entwickelt.

(c) $\mathcal{NP}$-Vollständigkeit

- $\mathcal{NP}$ und $\mathcal{P}$ sind Klassen von Entscheidungsproblemen, die entweder in nichtdeterministisch polynomieller Zeit ($\mathcal{NP}$) oder in deterministisch polynomieller Zeit ($\mathcal{P}$) gelöst werden können.
- $\mathcal{NP}$-vollständige Probleme gelten als die „schwierigsten“ Probleme und bisher ist es niemandem gelungen, einen Polynomialzeit-Algorithmus für ein $\mathcal{NP}$-vollständiges Problem zu finden, der das Problem löst.

- In der Regel können $\mathcal{NP}$-vollständige Probleme nur über Heuristiken oder andere Verfahren gelöst bzw. angenähert werden.

Wir werden auf den Begriff "$\mathcal{NP}$-Vollständigkeit" genauer in Kapitel 3 eingehen

Beispiel (VCP)**.** Das Vertex-Cover-Problem ist $\mathcal{NP}$-vollständig (Beweis später).

2.2 Formulierung als ganzzahliges lineares Programm (ILP)

Motivation Wenn es gelingt, das Optimierungsproblem als ILP zu formulieren, kann man hochentwickelte Solver nutzen, um das ILP und somit das Optimierungsproblem zu lösen. Beispiele für Solver:

- IBM CPLEX Optimizer (kommerziell)
- Gurobi Optimizer (kommerziell)
- GLPK, LP_ solve, CBC (open source)

Beispiel (ILP für VCP)**.** Sei $G = (V, E)$ der Eingabegraph. Gesucht ist $C \subseteq V$ minimaler Kardinalität, sodass C ein vertex cover von G ist. Formulierung als ILP:
Definiere Variable $x_v \in \{0, 1\} \forall v \in V$ mit $x_v = 1 \Leftrightarrow v \in C$. Das Optimierungsproblem lautet dann:

$$\min \sum_{v \in V} x_v \text{ s.d. für alle } (u, v) \in E \text{ gilt: } x_u + x_v \geq 1.$$

Anmerkungen.

- Man spricht von linearen Programmen, da sowohl die Zielfunktion als auch die Nebenbedingungen lineare Funktionen sind.
- Lineare Programme heißen ganzzahlig, wenn die Variablen nur ganze Zahlen als Werte annehmen dürfen.

2.3 Approximative Lösungen

Ziel Finde Polynomialzeit-Algorithmus, der eine Garantie liefert, dass die vom Algorithmus ausgegebene Lösung „nicht zu weit" vom Optimum entfernt ist, also eine gewisse Gütegarantie besitzt.

Beispiel (Approximative Lösung des VCP)**.**
Betrachte folgenden Algorithmus zur Lösung des VCP:

Anmerkungen.

- Approx_VC ist ein Polynomialzeit-Algorithmus mit Komplexität $\mathcal{O}(|E'|)$.
- C ist tatsächlich ein VC, weil aus E' nur Kanten entfernt werden, die mindestens einen Endpunkt in C haben.

Algorithm 1 Approx_VC ($G = (V, E)$)

1: $C \leftarrow \emptyset$
2: $E' \leftarrow E$
3: **while** $E' \neq \emptyset$ **do**
4: Wähle beliebige Kante $(uw) \in E'$.
5: $C \leftarrow C \cup \{u, w\}$
6: Entferne alle Kanten aus E', die inzident zu u oder w sind.
7: **end while**
8: **return** C als VC

- Approx_VC ist ein 2-Approximationsalgorithmus, d.h. C enthält höchstens doppelt so viele Knoten wie ein VC minimaler Kardinalität.

  Beweis: Sei C^* ein minimales VC von G. Sei $\tilde{E}$ die Menge aller in Schritt 4 ausgewählten Kanten. Diese Kanten sind paarweise disjunkt (wegen Schritt 6). Da C^* von jeder Kante aus $\tilde{E}$ mindestens einen Knoten enthält, gilt $|C^*| \geq |\tilde{E}|$. Nach Konstruktion von C gilt: $|C| = 2 \cdot |\tilde{E}|$. Daraus folgt $|C| = 2 \cdot |\tilde{E}| \leq 2 \cdot |C^*|$. □

- Die Menge $\tilde{E}$ im obigen Beweis bildet ein sogenanntes „maximales matching“ in G.

Definition (matching)**.** Sei $G = (V, E)$. Eine Teilmenge $M \subseteq E$ heißt *matching* von G, wenn alle Kanten in M paarweise disjunkt sind. Ein Matching maximaler Kardinalität heißt *maximum matching*. Ein Matching ist *maximal*, wenn $\nexists M' \subseteq E$, sodass M' ein matching ist und $M \subseteq M'$.

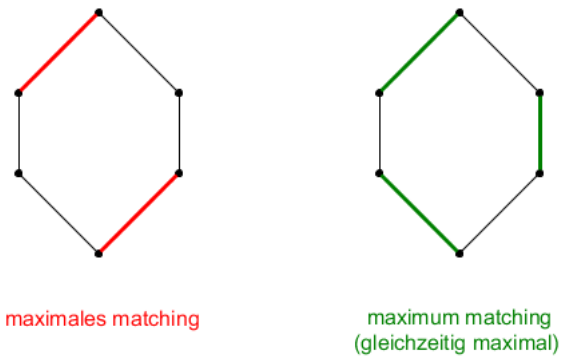

Abbildung 2.2: Beispiel für ein *maximales matching* und ein *maximum matching*

2.4 Betrachtung von Spezialfällen

Idee Oftmals lassen sich Spezialfälle (Teilinstanzen) diskreter Optimierungsprobleme in polynomieller Zeit lösen.

Beispiel (VCP)**.** Wir hatten bereits gesehen, dass für alle Graphen $G = (V, E)$ mit *minimalem vertex cover* C und *maximalem matching* M gilt: $|M| \leq |C|$ (da $|M| \leq |\tilde{E}| \leq |C^*|$).
Für einen Spezialfall, nämlich für den Fall, dass $G = (V, E)$ bipartit gilt sogar Gleichheit:

Theorem (König)**.** *Sei $G = (V, E)$ bipartit. Dann gilt $|M| = |C|$. (Beweis evtl. später)*

2.5 Parametrisierte Algorithmen

Motivation Oft hängt die Laufzeit eines Algorithmus von weiteren Parametern als der Eingabegröße ab. So können Instanzen $\mathcal{NP}$-vollständiger Probleme oft „effizient“ gelöst werden, wenn die entsprechenden Parameter bekannt und beschränkt sind.

Beispiel (VCP als Entscheidungsproblem)**.**
Gegeben: Graph $G = (V, E)$, natürliche Zahl $k \in \{0, 1, \ldots, |V|\}$
Frage: Gibt es ein VC mit $|C| \leq k$?
Beobachtung:

- Jeder Algorithmus, der dieses Entscheidungsproblem löst, kann dazu verwendet werden, das ursprüngliche Optimierungsproblem zu lösen.
- Das Entscheidungsproblem lässt sich in $\mathcal{O}(n^k)$ Zeit lösen:

  Beweis. Es müssen alle $\binom{n}{l}$ Teilmengen von V mit $l \leq k$ getestet werden. Dabei

  $$\binom{n}{k} = \frac{n!}{(n-k)!k!} = \frac{n(n-1)\cdots(n-k+1)}{k!} \leq \frac{n^k}{k!} \Rightarrow \mathcal{O}(n^k),$$

  also insbesondere

  $$\sum_{l=1}^{k} \binom{n}{l} \in \mathcal{O}(n^k + n^{k-1} + \ldots) = \mathcal{O}(n^k).$$

  □

  Eine Laufzeit von $\mathcal{O}(n^k)$ ist jedoch schlecht und damit nicht zufriedenstellend. Ziel ist es nun, durch Parametrisierung des Problems eine bessere Laufzeit zu erreichen.

Definition (FPT)**.** Existiert für ein Problem ein Algorithmus mit Laufzeit $\mathcal{O}(f(k) \cdot n^c)$, wobei c eine Konstante und f eine beliebige Funktion ist, die *nur* vom Parameter k abhängt, so heißt das Problem *Fixed-parameter tractable (FPT)* (parametrisierbar).

Intuitiv Die Komplexität bzw. „Schwierigkeit“ des Problems, lässt sich vollständig auf den Parameter k schieben.

Praxis Bezogen auf das VCP sind parametrisierte Algorithmen auch für mittelgroße Instanzen ($|V| \approx 50-100$) ausführbar, wohingegen Brute-Force-Ansätze oft schon für $n \geq 20$ nicht mehr anwendbar sind. Dies führt uns zu folgendem Algorithmus:

Algorithmus FPT_VC

Algorithm 2 FPT_VC ($G = (V, E)$, Variable j, Parameter k)

1: **if** $j > k$ **then**
2: **return false**
3: **else if** $E \neq \emptyset$ **then**
4: **return true**
5: **else**
6: Wähle beliebige Kante $(uw) \in E$ aus.
7: Setze $E_u = \{e \in E \mid u \notin E\}$ und $E_w = \{e \in E \mid w \notin E\}$
8: **end if**
9: **return** FTP_VC((V, E_u) , $j + 1$, k) oder FTP_VC((V, E_w) , $j + 1$, k)

- Starte mit FPT_VC($G = (V, E)$, 0, k)
- FPT_VC hat Laufzeit $\mathcal{O}(2^k \cdot |E|) = \mathcal{O}(2^k \cdot n^2)$ (siehe Tafel: Der Durchlauf von FPT_VC lässt sich durch einen Rekursionsbaum beschreiben, dessen Höhe durch k beschränkt ist $\Rightarrow$ höchstens 2^k Blätter).
- Es existieren sogar FPT-Algorithmen für das VCP mit Laufzeit $\mathcal{O}(1,274^k + k|V|)$.

Theorem. *FPT_ VC arbeitet korrekt.*

Beweisidee.
Zu zeigen: Wenn der gegebene Graph G ein VC mit höchstens k Knoten hat, gibt FPT_VC „true“ zurück.
Wegen Schritt 6 muss jede Kante $e = (uw)$ in E mindestens einen Knoten im VC haben.

- Falls u im VC, müssen alle Kanten, die inzident zu u sind, nicht mehr berücksichtigt werden. Lediglich die Kanten, die nicht inzident zu u sind, müssen noch betrachtet werden. Dies geschieht durch die Einführung der Menge E_u (Schritt 7) und den rekursiven Aufruf des Algorithmus.
- Analog, falls w im VC, müssen nur noch die Kanten betrachtet werden, die nicht indzident zu w sind. Dies geschieht über die Einführung der Menge E_w (Schritt 7) und den rekursiven Aufruf des Algorithmus.

□

Anmerkungen.

- Nicht für alle Probleme ist bekannt, ob sie FPT sind.
- Beim VCP entspricht der Parameter k der Kardinalität des *vertex covers*, es können aber auch andere Parameter, z.B. der Maximalgrad $\Delta(G)$ des Graphen einbezogen werden. In Abhängigkeit davon, wie der Parameter k gewählt wird, kann das Problem FPT oder Nicht-FPT sein, was folgendes Beispiel zeigt:

Beispiel (Cliquenproblem)**.**
Gegeben: Graph $G = (V, E)$, natürlich Zahl $k \in \{1, \ldots, |V|\}$
Frage: Existiert ein vollständiger Teilgraph $H \subseteq G$ (genannt Clique), sodass $|V(H)| \geq k$?

$\Rightarrow$ In diesem Fall entspricht k also der Größe der Clique und das Problem ist Nicht-FPT. Setzt man dagegen $k = \Delta(H)$, so wird das Problem parametrisierbar.

2.6 Heuristiken (ohne Gütegarantie)

Durch

- Schätzen, Beobachten, Vermuten
- intuitives bzw. intelligentes Raten oder
- zusätzliche Annahmen

versucht man, mit „geringem“ Rechenaufwand und „akzeptabler“ Laufzeit eine zulässige Lösung für das Optimierungsproblem zu finden, ohne dabei die Optimalitäts-Eigenschaft garantieren zu müssen.

Beispiele (Heuristiken)**.**

- A*-Algorithmen
- Simulated Annealing
- naturnahe Optimierungsverfahren (Ameisen-Algorithmus, Bienen-Algorithmus, Schwarm-Algorithmus)
- Greedy-Algorithmen (Klasse von Algorithmen, die schrittweise den Folgezustand wählen, der zum Zeitpunkt der Wahl den größten Gewinn bzw. das beste Ergebnis liefert.)

Beispiel (Algorithmus Greedy_VC)**.**

Algorithm 3 Greedy_VC ($G = (V, E)$)

1: $C \leftarrow \emptyset$
2: $E' \leftarrow E$
3: **while** $E' \neq \emptyset$ **do**
4: Wähle Knoten $v \in V$ mit höchstem Grad.
5: Entferne v aus V.
6: $C \leftarrow C \cup \{v\}$
7: Entferne alle Kanten aus E', die inzident zu v sind.
8: **end while**
9: **return** C als VC

Lemma. *Greedy_ VC hat keine beschränkte Gütegarantie.*

Beweis.

1. Konstruiere einen Graphen $G = (V, E)$ wie folgt:
   $V = L \dot\cup R$, $|L| = r$, $R = R_1 \dot\cup R_2 \dot\cup \ldots \dot\cup R_r$, sodass jeder Knoten aus R_i eine Kante zu i verschiedenen Knoten aus L hat und keine zwei Knoten aus R_i einen gemeinsamen Nachbarn in L haben, wobei $|R_i| = \lfloor \frac{r}{i} \rfloor$. Per Konstruktion gilt: $\forall\, v \in L : \; deg(v) \leq r$ und $\forall\, v \in R_i : \; deg(v) \leq i$.

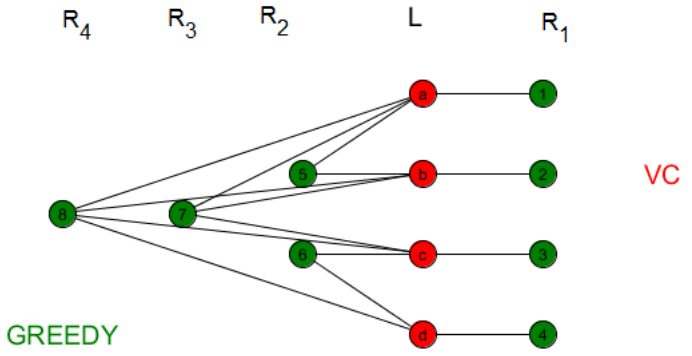

Abbildung 2.3: Graph zu GREEDY_VC

2. Sei also $G = (V, E)$ und $V = L \dot\cup R$ wie oben definiert.

   (i) Zeigen erst: $r \ln(r) \leq |V| \leq r \ln(r) + 2r$:
   Klar, es gilt $|V| = r + \sum_{i=1}^{r} \lfloor \frac{r}{i} \rfloor$. Betrachte dazu die Funktion $\frac{1}{x}$ und schätze das zugehörige Integral durch Unter- und Obersumme ab, d.h.

$$\sum_{i=2}^{r} \frac{1}{i} \leq \underbrace{\int_1^r \frac{1}{i} \mathrm{d}i}_{=\ln(r)} \leq \sum_{i=1}^{r-1} \frac{1}{i}$$

   Umstellen ergibt

$$\ln(r) \leq \sum_{i=1}^{r} \frac{1}{i} \leq \ln(r) + 1$$

   Also

$$r \ln(r) \leq r \sum_{i=1}^{r} \frac{1}{i} = r + \sum_{i=1}^{r} \frac{1}{i} - r \leq |V| = r + \sum_{i=1}^{r} \lfloor \frac{r}{i} \rfloor \leq r + \sum_{i=1}^{r} \frac{r}{i} = r\Big(1 + \sum_{i=1}^{r} \frac{1}{i}\Big) \leq r(\ln(r) + 2)$$

   Somit gilt

$$r \ln(r) \leq |V| \leq r(\ln(r) + 2)$$

   und damit $|V| \approx r \ln(r)$.

   (ii) Nach Konstruktion von G sind sowohl L als auch R *vertex cover*. Im "schlechten

Fall" wählt der Greedy Algorithmus R als VC. Dann erhalten wir

$$\ln(r) - 1 = \frac{r\ln(r) - r}{r} \leq \frac{|R|}{|L|} = \frac{|V| - |L|}{|L|} = \frac{|V| - r}{r} \leq \frac{r\ln(r) + 2r - r}{r} = \ln(r) + 1,$$

d.h. $\frac{|R|}{|L|} \approx \ln(r)$. Greedy_VC liefert im worst case also eine Lösung, die um $\ln(r)$ schlechter ist als die optimale Lösung.

□

3 Komplexitätsbetrachtungen

3.1 Einführung

Beobachtung Es gibt Probleme, für die Polynomialzeit-Algorithmen existieren, z.B.:

- Bestimmung des Maximalgrads eines Graphen $G = (V, E)$
- Test, ob ein Graph $G = (V, E)$ bipartit ist.
- Bestimmung des Komplements $\overline{G}$ eines Graphen.

Frage Gibt es für alle Probleme einen Algorithmus, der

(i) das Problem löst?

(ii) das Problem in polynomieller Zeit löst?

Antwort

(i) Nein. Betrachte dazu das sogenannte HALTE-Problem (Turing, 1936):

Frage: Gibt es einen Algorithmus, der entscheidet, ob ein beliebiger anderer Algorithmus terminiert?

Antwort: Einen solcher Algorithmus existiert nicht.

Beweisidee. Angenommen es gäbe einen solchen Algorithmus. Dann hätte er folgende Struktur:

```
HALT (Algorithmus A)
  If(A terminiert) then return true
  Else return false
```

Sei nun Algorithmus A der Form

```
test()
  while(HALT(test())
```

Dann gibt es zwei Fälle:

a) `test()` terminiert $\Rightarrow$ `HALT` liefert true $\Rightarrow$ while-Schleife läuft für immer $\Rightarrow$ `test()` terminiert nicht $\lightning$

b) `test()` terminiert nicht $\Rightarrow$ `HALT` liefert false $\Rightarrow$ while-Schleife terminiert $\Rightarrow$ `test()` terminiert $\lightning$

$\Rightarrow$ `HALT ()` existiert nicht. Das HALTE-Problem ist algorithmisch nicht entscheidbar. □

(ii) offen; führt auf $\mathcal{NP}$-Vollständigkeit (s.u.)

3.2 Komplexitätsklassen

Die Komplexität von Entscheidungsproblemen lässt sich im Wesentlichen in drei Klassen einteilen:

- Es existiert kein Algorithmus für das Problem, d.h. das Problem ist nicht entscheidbar.
- Es existiert ein Polynomialzeit-Algorithmus für das Problem.
- Es sind keine Polynomialzeit-Algorithmen für das Problem bekannt (z.B. nur Exponentialzeit-Algorithmen bekannt), aber es gibt keinen Beweis, dass kein Polynomialzeit-Algorithmus existiert.

Für die Klassifikation der Komplexität von Optimierungsproblemen werden deren Entscheidungsprobleme betrachtet (dies geht auf die Theorie von Turingmaschinen & Sprachen zurück und wird in anderen Vorlesungen behandelt).

Beispiel (VCP)**.**

- VC-Optimierungsproblem:

  Gegeben: Graph $G = (V, E)$

  Gesucht: VC minimaler Kardinalität

- VC-Entscheidungsproblem:

  Gegeben: Graph $G = (V, E)$, natürliche Zahl $k \in \{1, \ldots, |V|\}$

  Frage: Gibt es ein VC C mit $|C| \leq k$?

Wenn das Optimierungsproblem „einfach“, d.h. effizient lösbar ist, so ist auch das Entscheidungsproblem „einfach“. Umgekehrt gilt, wenn das Entscheidungsproblem „schwer“ ist, dann ist das Optimierungsproblem „mindestens genauso schwer“.

Definition (Komplexitätsklassen)**.**

Mit $\mathcal{P}$ bezeichnet man eine Klasse von Entscheidungsproblemen, die in polynomieller Zeit von einer deterministischen Turingmaschine gelöst werden können.

Mit $\mathcal{NP}$ bezeichnet man eine Klasse von Entscheidungsproblemen, die in polynomieller Zeit von einer nicht-deterministischen Turingmaschine gelöst werden können.

Dabei gilt für eine *deterministische Turingmaschine*: Für gegebene Eingabe, sind der momentane Rechenschritt sowie alle folgenden Rechenschritte eindeutig bestimmt, d.h. die Zwischenresultate sind für eine gegebene Eingabe in jedem Durchlauf gleich.
Für eine *nicht-deterministische Turingmaschine* gilt: Anstelle von einer Regel für jeden Rechenschritt bei gegebener Eingabe, sind verschiedene Übergangsregeln möglich (nicht eindeutig bestimmt), um zum nächsten Schritt zu gelangen. In jedem Schritt wird aber der richtige Folgezustand ausgewählt, wenn das Entscheidungsproblem eine Ja-Antwort hat.

Um zu zeigen, dass ein Entscheidungsproblem

- $\in \mathcal{P}$, genügt es, einen Polynomialzeit-Algorithmus anzugeben, der das Problem löst.
- $\in \mathcal{NP}$, genügt es zu zeigen, dass eine korrekte, vorgegebene Lösung in polynomieller Zeit als richtig klassifiziert werden kann.

Es ist klar, dass $\mathcal{P} \subseteq \mathcal{NP}$, aber die Frage, ob $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \neq \mathcal{NP}$ stellt eines der sogenannten Millennium-Probleme dar. Es wird im Allgemeinen angenommen, dass $\mathcal{P} \neq \mathcal{NP}$.

3.3 Prinzip der Reduktion und NP-Vollständigkeit

3.3.1 Polynomialzeitreduktion

Beispiel (Einführendes Beispiel für das Prinzip der Reduktion)**.** Angenommen wir haben ein Problem A unbekannter Komplexität und ein weiteres Problem B, das "einfach" ist, z.B.

Problem A:

Gegeben: Sack voll 1€ Stücke, natürliche Zahl k

Frage: Enthält der Sack genau k Münzen?

Problem B:

Gegeben: Item I, natürliche Zahl L

Frage: Wiegt Item I genau L Gramm?

Sei also eine beliebige Instanz $\alpha \in A$ gegeben, z.B. α = beliebiger Sack mit Münzen und $k = 1223313$. Dann ist die Instanz $\alpha \in A$ äquivalent zu folgender Instanz $\beta \in B$:
Nimm eine Münze und wiege diese (7,5 g).
Die Beantwortung der Frage "Wiegt Item I = (Münzen ohne Sack) genau $L = k \cdot 7{,}5$ g?" beantwortet damit die Frage des Problems A.
Somit kann man die Lösung für jede Instanz von A erhalten, indem man die zugehörige Instanz des Problem B löst. Da Problem B "einfach" ist (mit hinreichend gute Waage), ist auch das Problem A "einfach".

Grundidee Sei Problem $B \in \mathcal{P}$ und Problem A von unbekannter Komplexität. Angenommen es existiert eine „Prozedur“, die jede Instanz α von A in eine Instanz β von B transformiert, sodass

1. die Transformation in polynomieller Zeit erreicht werden kann und
2. die Antworten auf die entsprechenden Entscheidungsprobleme identisch sind, d.h. α hat Ja-Antwort $\Leftrightarrow$ β hat Ja-Antwort,

Als “Bild”:

$$\underbrace{\forall\, \alpha \in A \,\exists\, \beta \in B \text{ sodass } \alpha \leq_p \beta}_{\text{kurz: } A \leq_p B} \xrightarrow{Polynomialzeit-Alg.} \text{Lösung für } \beta = \begin{cases} \text{Ja} \longrightarrow & \text{Ja für } \alpha \\ \text{Nein} \longrightarrow & \text{Nein für } \alpha \end{cases},$$

Somit folgt $A \in \mathcal{P}$. Wir haben die Einfachheit von Problem B genutzt, um die Einfachheit von Problem A zu zeigen, formal:

$$A \leq_p B \,:\, B \in \mathcal{P} \Rightarrow A \in \mathcal{P}$$

Umgekehrt, falls

$$A \notin \mathcal{P} \text{ und } A \leq_p B \Rightarrow B \notin \mathcal{P},$$

d.h., wenn A nicht in Polynomialzeit gelöst werden kann, kann auch B nicht in Polynomialzeit gelöst werden oder in anderen Worten: Wenn A „schwer“, dann ist B mindestens genauso „schwer“.

Anmerkung. Letztere Grundidee und das Beispiel sollen nur eine “Gedankenstütze” sein. Im Folgenden betrachten wir “$\mathcal{NP}$-Vollständigkeit”, nehmen aber *nicht* an, dass kein Polynomialzeit-Algorithmus existiert.

Die Methodik ist aber ähnlich, d.h. wir werden das Prinzip der Reduktion nutzen, um die „Schwere“ von Problemen zu zeigen: Wenn die Komplexität eines Problems B unbekannt ist, es aber ein „schweres“ Problem A gibt, das sich auf B reduzieren lässt, so muss B mindestens genauso „schwer“ sein wie A.

3.3.2 NP-Vollständigkeit

Definition ($\mathcal{NP}$-vollständig)**.** Ein Entscheidungsproblem D ist *$\mathcal{NP}$-vollständig* $\Leftrightarrow$

1. D in der Klasse $\mathcal{NP}$ liegt, d.h. $D \in \mathcal{NP}$ und
2. D ist $\mathcal{NP}$-schwer, d.h. $\forall\, D' \in \mathcal{NP} : D' \leq_p D$.
   (Jedes Problem D' in $\mathcal{NP}$ kann durch Polynomialzeitreduktion auf D reduziert werden. Das bedeutet auch, dass ein Algorithmus zur Lösung von D dazu genutzt werden kann, um alle anderen Probleme in $\mathcal{NP}$ zu lösen.)

Nachweis der $\mathcal{NP}$-Vollständigkeit von Problemen:

1. Um zu zeigen, dass $D \in \mathcal{NP}$, gibt man eine Lösung an (z.B. durch Raten) und zeigt, dass in polynomieller Zeit verifiziert werden kann, ob die Lösung richtig ist.

2. Um die $\mathcal{NP}$-Schwere von D zu zeigen, nimmt man ein beliebiges $\mathcal{NP}$-vollständiges Problem D' und zeigt, dass es auf das betrachtete Problem D reduziert werden kann, d.h. $D' \leq_p D$. Aus der Transitivität der Polynomialzeitreduktion folgt dann, dass alle anderen Probleme D'' aus $\mathcal{NP}$ ebenfalls auf das betrachtete Problem D reduzierbar sind:
$$\forall\, D'' \in \mathcal{NP} : D'' \leq_p D' \text{gilt: wenn } D' \leq_p D \text{ dann auch } D'' \leq_p D$$

Klar, es muss ein erstes Problem gegeben haben, dessen $\mathcal{NP}$-Schwere bewiesen wurde, das sogenannte SAT-Problem (COOK-LEVIN-THEOREM (1971): SAT ist $\mathcal{NP}$-vollständig). Der Beweis dieses Theorems ist aber außerhalb des Fokus dieser Vorlesung. Das Standardbuch zu diesem Thema ist "Computers and Intractability: A Guide to the Theory of NP-Completeness" von M.R. Garey und D.S. Johnson.

3.4 Das SAT-Problem

Erfüllbarkeitsproblem der Aussagenlogik (SAT, von englisch *satisfiability* = Erfüllbarkeit)

Gegeben: Boolescher Ausdruck φ (oBdA. in konjunktiver Normalform) mit

- AND($\wedge$)-, OR($\vee$)- und NOT-Operationen
- Literalen (Negation oder Nicht-Negation von booleschen Variablen)
- Klammern „ (“, „) “

z.B.
$$\varphi = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{\text{Klausel 1}} \wedge \underbrace{(x_1 \vee \overline{x_4} \vee x_3 \vee x_5)}_{\text{Klausel 2}}$$

Frage: Gibt es eine Wahrheitsbelegung der booleschen Variablen, sodass φ wahr (erfüllbar) ist (dazu muss jede OR-Verknüpfung von Literalen, genannt Klausel, wahr sein.)

Variante: k-SAT: SAT, sodass jede Klausel genau k Literale besitzt, d.h.
$$\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_n \text{ gilt } |C_i| = k, \text{ also } C_i = l_1^i \vee \ldots \vee l_k^i$$

Theorem. *3-SAT ist $\mathcal{NP}$-vollständig.*

Beweis.

1. 3-SAT ist in $\mathcal{NP}$:
   Sei φ ein Boolescher Ausdruck und die X eine Menge der zugehörigen Booleschen Variablen. Gegeben eine Wahrheitsbelegung $f : X \to \{0, 1\}$, so lässt sich in polynomieller Zeit überprüfen, ob φ wahr.

2. 3-SAT ist $\mathcal{NP}$-schwer:
   Dazu zeigen wir SAT $\leq_p$ 3-SAT (SAT ist $\mathcal{NP}$-vollständig und aus SAT $\leq_p$ 3-SAT folgt damit, dass 3-SAT $\mathcal{NP}$-schwer).
   Sei also $\varphi \in$ SAT, $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$, $|C_i| \in \{1, 2, 3, \ldots, n\}$.
   Nun betrachten wir die folgenden möglichen Fälle:

   - Wenn $|C_i| = 3 \, \forall i \Rightarrow \varphi \in$ 3-SAT, d.h. es ist nichts zu zeigen.
   - Es gibt eine Klausel C_i mit $|C_i| = 2$, z.B. $C_i = (x \vee y)$. Dann führen wir eine neue Variable u und zwei neue Klauseln $C_i' = (x \vee y \vee u)$ und $C_i'' = (x \vee y \vee \overline{u})$ ein, die C_i ersetzen. Damit gilt:
     - C_i wahr $\Rightarrow x = 1$ oder $y = 1 \Rightarrow C_i'$ und C_i'' erfüllbar.
     - C_i' und C_i'' wahr $\Rightarrow \begin{cases} u = 1 \Rightarrow \overline{u} = 0 \\ u = 0 \Rightarrow \overline{u} = 1 \end{cases}$ $\Rightarrow$ in jedem Fall muss $x = 1$ oder $y = 1$ sein $\Rightarrow C_i$ erfüllbar
   - Es gibt eine Klausel C_i mit $|C_i| = 1$, z.B. $C_i = (x)$. Dann führen wir eine neue Variable u ein und ersetzen C_i durch zwei neue Klauseln $C_i' = (x \vee u)$ und $C_i'' = (x \vee \overline{u})$. Es ist klar, dass C_i erfüllbar $\Leftrightarrow C_i'$ und C_i'' erfüllbar. Analog zum Fall $|C_i| = 2$, führen wir nun noch eine Variable v ein und ersetzen

     C_i' durch $\widetilde{C_i} = (x \vee u \vee v)$ und $\widetilde{\widetilde{C_i}} = (x \vee u \vee \overline{v})$

     C_i'' durch $\widehat{C_i} = (x \vee \overline{u} \vee v)$ und $\widehat{\widehat{C_i}} = (x \vee \overline{u} \vee \overline{v})$

     Wie oben folgt C_i' und C_i'' erfüllbar $\Leftrightarrow \widetilde{C_i}, \widetilde{\widetilde{C_i}}, \widehat{C_i}$ und $\widehat{\widehat{C_i}}$ erfüllbar.
   - Betrachte nun noch den Fall $|C_i| > 3$, d.h. $C_i = (x_1 \vee x_2 \vee \ldots \vee x_k)$. Dann führen wir neue Variablen $u_1, \ldots u_{k-3}$ ein und ersetzen C_i durch Klauseln der Form

$$
\begin{aligned}
C_i^1 &= (x_1 \vee x_2 \vee u_1) \\
C_i^2 &= (x_3 \vee \overline{u_1} \vee u_2) \\
C_i^3 &= (x_4 \vee \overline{u_2} \vee u_3) \\
&\vdots \\
C_i^{l-2} &= (x_{l-1} \vee \overline{u_{l-3}} \vee u_{l-2}) \\
C_i^{l-1} &= (x_l \vee \overline{u_{l-2}} \vee u_{l-1}) \\
C_i^l &= (x_{l+1} \vee \overline{u_{l-1}} \vee u_l) \\
&\vdots \\
C_i^{k-3} &= (x_{k-2} \vee \overline{u_{k-4}} \vee u_{k-3}) \\
C_i^{k-2} &= (x_{k-1} \vee x_k \vee \overline{u_{k-3}}).
\end{aligned}
$$

     Zeigen nun C_i erfüllbar $\Leftrightarrow C_i^j$ erfüllbar, $j = 1, \ldots, k-2$:

     „$\Rightarrow$“ Sei C_i erfüllbar $\Rightarrow$ mindestens ein Literal x_l in C_i ist wahr, d.h. $x_l = 1$.

$$\Rightarrow \text{Setze } \begin{cases} u_j = \text{ wahr }, j < l-2 \\ u_j = \text{ falsch, sonst} \end{cases}$$
$\Rightarrow$ Alle C_i^j erfüllbar, $j = 1, \ldots, k-2$. (siehe Tafel oder nachrechnen)

„$\Leftarrow$“ Seien alle C_i^j erfüllbar, $j = 1, \ldots, k-2$. Angenommen C_i ist nicht erfüllbar.
$\Rightarrow x_i = 0$ für alle $i = 1, \ldots, k-2$
$\Rightarrow$ insbesondere $x_{k-1} = 0$ und $x_k = 0$
Da aber alle C_i^j erfüllbar, $j = 1, \ldots, k-2$, folgt $\overline{u_{k-3}} = 1$, also $u_{k-3} = 0$
$\Rightarrow u_{k-4} = 0$ usw.
$\Rightarrow$ alle $u_i = 0$, $i = 1, \ldots, k-3$
$\Rightarrow C_i^1$ nicht erfüllbar $\lightning$
Also war die Annahme falsch, d.h. C_i muss erfüllbar sein.

□

3.5 Weitere NP-Vollständige Probleme

3.5.1 Cliquen-Problem

CLIQUE-Problem

Gegeben: Graph $G = (V, E)$ natürliche Zahl $L \in \{1, \ldots, |V|\}$
Frage: Existiert ein Teilgraph $H \subseteq G$, sodass $H \cong K_j$, $j \geq L$
(d.h. H ist vollständiger Teilgraph mit $j \geq L$ Knoten)

Theorem. *CLIQUE ist $\mathcal{NP}$-vollständig.*

Beweis.

1. CLIQUE $\in \mathcal{NP}$ (Prüfe bei k gegebenen Knoten in polynomieller Zeit, ob zwischen je zwei paarweise verschiedenen Knoten eine Kante existiert.)

2. CLIQUE ist $\mathcal{NP}$-schwer. Dazu zeigen wir 3-SAT $\leq_p$ CLIQUE:
   Konkret transformieren wir eine 3-SAT Formel φ in einen Graphen $G_\varphi = (V, E)$ und eine Zahl $k \in \mathbb{N}$, sodass gilt: φ erfüllbar $\Leftrightarrow$ G hat eine k-Clique. Dazu

   - Seien $C_1, \ldots, C_k$ die Klauseln von φ.
   - Seien l_1^i, l_2^i und l_3^i die Literale in Klausel C_i.
   - Identifiziere Literale und Knoten. d.h. setze

   $$V = \{l_i^j \mid 1 \leq j \leq k, 1 \leq i \leq 3\}.$$

   - Jedes Knotenpaar l_r^i und l_s^j wird durch eine Kante verbunden, außer

     a) die assoziierten Literale gehören zur selben Klausel (d.h. es muss gelten $i \neq j$)

b) eines der beiden Literale ist die Negation des anderen Literals (d.h. l_r^i und l_s^j sind nicht durch eine Kante verbunden, wenn z.B. $l_r^i = x$ und $l_s^j = \overline{x}$).

Wir zeigen nun die Korrektheit der Transformation:

(i) φ erfüllbar $\Rightarrow$ G hat eine k-Clique:
Da φ erfüllbar, beinhaltet jede Klausel C_i mindestens ein wahres Literal l_r^i, $r \in \{1, 2, 3\}$. Wähle pro Klausel eines der wahren Literale beliebig aus. Die damit assoziierten k Knoten bilden eine k-Clique, da:

- Alle ausgewählten Literale gehören zu verschiedenen Klauseln. Es kann also keine Kante aufgrund von Regel a) fehlen.
- Alle ausgewählten Literale werden gleichzeitig erfüllt, widersprechen sich also nicht. Es kann somit auch keine Kante wegen Regel b) fehlen.

(ii) G hat eine k-Clique $\Rightarrow$ φ erfüllbar:

- Wenn G eine k-Clique hat, so müssen aufgrund von Regel a), die Knoten in dieser Clique zu verschiedenen Klauseln gehören.
- Die k-Clique wählt somit ein Literal pro Klausel aus.
- Diese Literale können alle gleichzeitig auf "1" gesetzt werden, da sie sich aufgrund von Regel b) nicht widersprechen.
- Also ist φ erfüllbar, da mindestens ein Literal für jede Klausel auf wahr gesetzt wurde.

□

3.5.2 Vertex-Cover-Problem

Theorem. *VCP ist $\mathcal{NP}$-vollständig.*

Beweis.

1. VCP $\in \mathcal{NP}$ (Es lässt sich in polynomieller Zeit überprüfen, ob eine vorgegebene Knotenmenge $C \subseteq V$ tatsächlich ein *vertex cover* bildet, indem man überprüft, ob für alle Kanten aus E mindestens ein Endpunkt in C liegt.)

2. VCP ist $\mathcal{NP}$-schwer. Dazu zeigen wir CLIQUE $\leq_p$ VCP. Insbesondere zeigen wir:
G hat Clique V' der Größe $k \Leftrightarrow \overline{G}$ hat VC $C = V \setminus V'$ der Größe $|V| - k$.

   „$\Rightarrow$" Sei $V' \subseteq V$ eine Clique in G,
   d.h. für den (Knoten)induzierten Teilgraph $\langle V' \rangle$ gilt $\langle V' \rangle \simeq K_k$
   Angenommen $C = V \setminus V'$ ist kein VC.
   $\Rightarrow$ $\exists$ Kante $(uv) \in E(\overline{G})$, sodass weder u noch v in C.
   $\Rightarrow$ $u, v \in V'$. Da aber $(uv) \in E(\overline{G}) \Rightarrow (uv) \notin E(G)$
   $\Rightarrow$ V' ist keine Clique ↯.

Klar, wenn V' Clique der Größe k,
dann ist somit $V \setminus V'$ ein VC der Größe $|V| - k$.

„⇐“ Sei $C = V \setminus V'$ ein VC in $\overline{G}$.
$\Rightarrow \forall\, (uv) \in E(\overline{G})$ gilt $u \in V \setminus V'$ oder $v \in V \setminus V'$
$\Rightarrow$ mindestens einer der Knoten u und v ist nicht in V'
$\Rightarrow$ also gilt für alle Knoten $x, y \in V'$, dass $(xy) \notin E(\overline{G})$
$\Rightarrow\ (xy) \in E(G)$
$\Rightarrow\ V'$ ist Clique in G.

Klar, wenn $V \setminus V'$ ein VC der Größe $|V| - k$,
dann ist somit V' eine Clique der Größe k.

□

4 Kürzeste Wege in Graphen

Motivation Dieses Kapitel stellt eines der fundamentalsten Probleme der Optimierung vor und zeigt gleichzeitig, wie „leichte Modifikationen“ der Problemformulierung ein Problem „einfach“ oder „schwer“ machen können.
Um kürzeste Wege in Graphen bestimmen zu können, brauchen wir noch folgende Definitionen:

Definition (Weg und Spaziergang)**.** In einem (gerichteten oder ungerichteten) Graphen $G = (V, E)$ ist ein *Weg (path)* $W = (v_1, \ldots, v_{k+1})$ von v_1 nach v_{k+1} eine Folge von Knoten aus V, sodass:

(i) $v_i \neq v_j \; \forall \, 1 \leq i < j \leq k+1$

(ii) $e_i = (v_i, v_{i+1}) \in E \; \forall \, 1 \leq i < j \leq k$

(iii) $e_i \neq e_j \; \forall \, 1 \leq i < j \leq k$

Wenn für W nur die Bedingungen (ii) und (iii) erfüllt sind, dann heißt W *Spaziergang (walk)*.

Definition (Kreisspaziergang und Kreis)**.** Ein *Kreisspaziergang* $W = (v_1, \ldots, v_k)$ mit $k \geq 2$, ist eine Folge von Knoten aus V, sodass $(v_1, \ldots, v_k, v_1)$ ein Spaziergang ist. Ein (echter, elementarer) *Kreis* ist ein Kreisspaziergang, sodass alle Knoten $v \in W$ auf diesem Kreis Grad 2 haben.

Definition (Länge eines Wegs/ Kreises)**.** Sei $G = (V, E)$ und $c : E \to \mathbb{R}$. Dann ist die *Länge* eines Wegs/ Kreises W in G definiert als

$$C(W) = \sum_{e \in W} c(e).$$

Wenn $c(e) = 1 \; \forall \, e \in E$, dann entspricht die Länge von W der Anzahl der Kanten von W.

Definition (Distanz)**.** Die *Distanz* zweier Knoten $x, y \in V$ ist definiert als

$$d_G(x, y) = \min_{W \in W_{xy}} c(W),$$

wobei W_{xy} die Menge aller Wege von x nach y beschreibt. Falls es keinen Weg von x nach y gibt, d.h. $W_{xy} = \emptyset$, setzen wir $d_G(x, y) = \infty$.

Definition (zusammenhängend)**.** Ein ungerichteter Graph heißt *zusammenhängend*, wenn für alle $x, y \in V : \; W_{xy} \neq \emptyset$.

Ein gerichteter Graph heißt *(schwach) zusammenhängend*, wenn der ungerichtete Graph $G' = (V, \{\{u,v\} \,|(uv) \in E \text{ oder } (vu) \in E\})$ zusammenhängend ist.

Definition (Baum und DAG)**.** Ein *Baum* ist ein ungerichteter, azyklischer, zusammenhängender Graph.
Ein *DAG* (directed acyclic graph) ist ein gerichteter, azyklischer, zusammenhängender Graph.

Kürzeste Wege in der Praxis Betrachte Landkarte und suche z.B.

a) den kürzesten Weg von Greifswald nach Leipzig.

b) den kürzesten Weg von Greifswald nach Leipzig, der zusätzlich die Städte Berlin, Potsdam und Rostock besucht.

Problem 4.1 (Ein kürzester Weg)**.**
Geg.: $G = (V, E)$, $c : E \to \mathbb{R}$, $s, t \in V$
Ziel: Bestimme den kürzesten $s-t-$Weg, d.h. $\min\limits_{W \in W_{st}} c(W)$

Problem 4.2 (Alle kürzesten Wege von s)**.**
Geg.: $G = (V, E)$, $c : E \to \mathbb{R}$, $s \in V$
Ziel: Bestimme alle kürzesten Wege, die von s ausgehen, d.h. $\min\limits_{W \in W_{st}} c(W) \,\forall\, t$

Problem 4.3 (Alle kürzesten Wege)**.**
Geg.: $G = (V, E)$, $c : E \to \mathbb{R}$
Ziel: Finde $\min\limits_{W \in W_{st}} c(W) \,\forall\, s, t$

Problem 4.4 ($s-t-$Weg über alle Knoten)**.**
Geg.: $G = (V, E)$, $c : E \to \mathbb{R}$
Ziel: Finde kürzesten Weg von s nach t, der auch alle anderen Knoten von $V \setminus \{s, t\}$ besucht.

4.1 NP-schwere Kürzeste-Wege-Probleme

Hamiltonische Wege

Definition (hamiltonisch)**.** Sei $G = (V, E)$ ein Digraph. Ein $s-t-$Weg in G, der jeden Knoten genau einmal besucht, heißt *hamiltonisch*.

Anmerkung. Sei $G = (V, E)$ ein Digraph, $c : E \to \mathbb{R}$ so, dass $\forall e \in E : c(e) = -1$. Alle $s-t-$Wege mit Gewicht $c(W) = 1 - |V|$ sind dann die Wege, die jeden Knoten genau einmal besuchen, d.h. sie sind hamiltonisch.

Problem 4.5 (Hamiltonischer Weg)**.**
Geg.: $G = (V, E)$ Digraph.
Frage: Gibt es einen hamiltonischen Weg in G?

Theorem. *Das Problem Hamiltonischer Weg ist $\mathcal{NP}$-vollständig.*

Beweis.

1. Hamiltonischer Weg ist in $\mathcal{NP}$ $\surd$

2. Hamiltonischer Weg ist $\mathcal{NP}$-schwer. Wir zeigen dazu: 3-SAT $\leq_p$ Hamiltonischer Weg
   Sei $\varphi \in$ 3-SAT, $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$, $|C_i| = 3$, $C_i = (a_i \vee b_i \vee c_i)$. Die Booleschen Variablen seien $\{x_1, \ldots, x_l\}$.
   Wir konstruieren nun einen gerichteten Graph G', bestehend aus sogenannten *gadgets* für die Variablen (siehe Abbildung 4.1) und *gadgets* für die Klauseln und zeigen schließlich: φ erfüllbar $\Leftrightarrow$ G' hat einen hamiltonischen Weg.

   Die booleschen Variablen werden entsprechend ihrer Indizes geordnet: $x_1, \ldots, x_l$. Wir schreiben/zeichnen die *gadgets* der x_i von oben nach unten und identifizieren den TOP-Knoten von x_{i+1} mit dem BOTTOM-Knoten von x_i, $1 \leq i \leq l-1$ (siehe Abbildung 4.2).

   In jedem *gadget* x_i zerlegen wir die $3k+3$ Knoten in k Paare $1, \ldots, k$. Um den Graph G' zu konstruieren, verknüpfen wir die *gadgets* der Variablen und der Klauseln wie folgt (siehe auch Abbildung 4.3): Für jede in C_j vorkommende Variable x_i gibt es zwei Möglichkeiten des Auftretens der Variable; (i) x_i kommt in C_j vor oder (ii) $\overline{x}_i$ kommt in C_j vor. Seien die Knoten $3j$ und $3j+1$ aus dem gadget für diese Variable x_i. Im Fall (i) setze Kante $(3j, C_j)$ und $(C_j, 3j+1)$, und im Fall (ii) setze Kante $(C_j, 3j)$ und $(3j+1, C_j)$

   Betrachte zum Beispiel $\varphi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}_{C_2}$. Der daraus resultierende Graph G' ist in Abbildung 4.4 dargestellt.

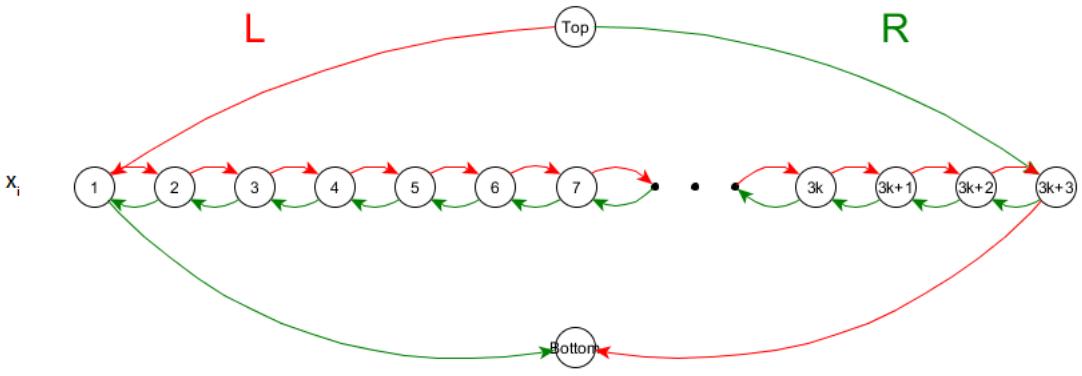

Abbildung 4.1: Gadget für Variable x_i

Wir zeigen nun die Korrektheit der Transformation:

(i) φ erfüllbar $\Rightarrow$ $\exists$ ein Hamiltonischer Weg in G':
Sei φ erfüllbar. Um zu zeigen, dass ein Hamiltonischer Weg in G' existiert, traversiere den G' von oben nach unten wie folgt:

- Gehe von L nach R, wenn x_i = wahr.

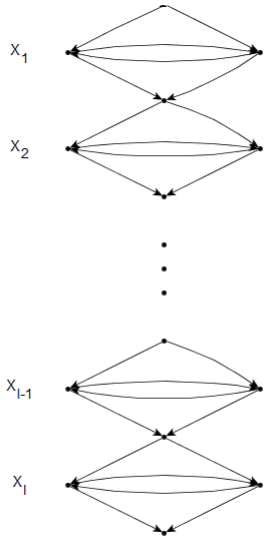

Abbildung 4.2: Graph der Gadgets der booleschen Variablen

(a) Situation: x_i in C_j

(b) Situation: $\overline{x_i}$ in C_j

Abbildung 4.3: Verknüpfung von Gadget für x_i mit Gadget für C_j

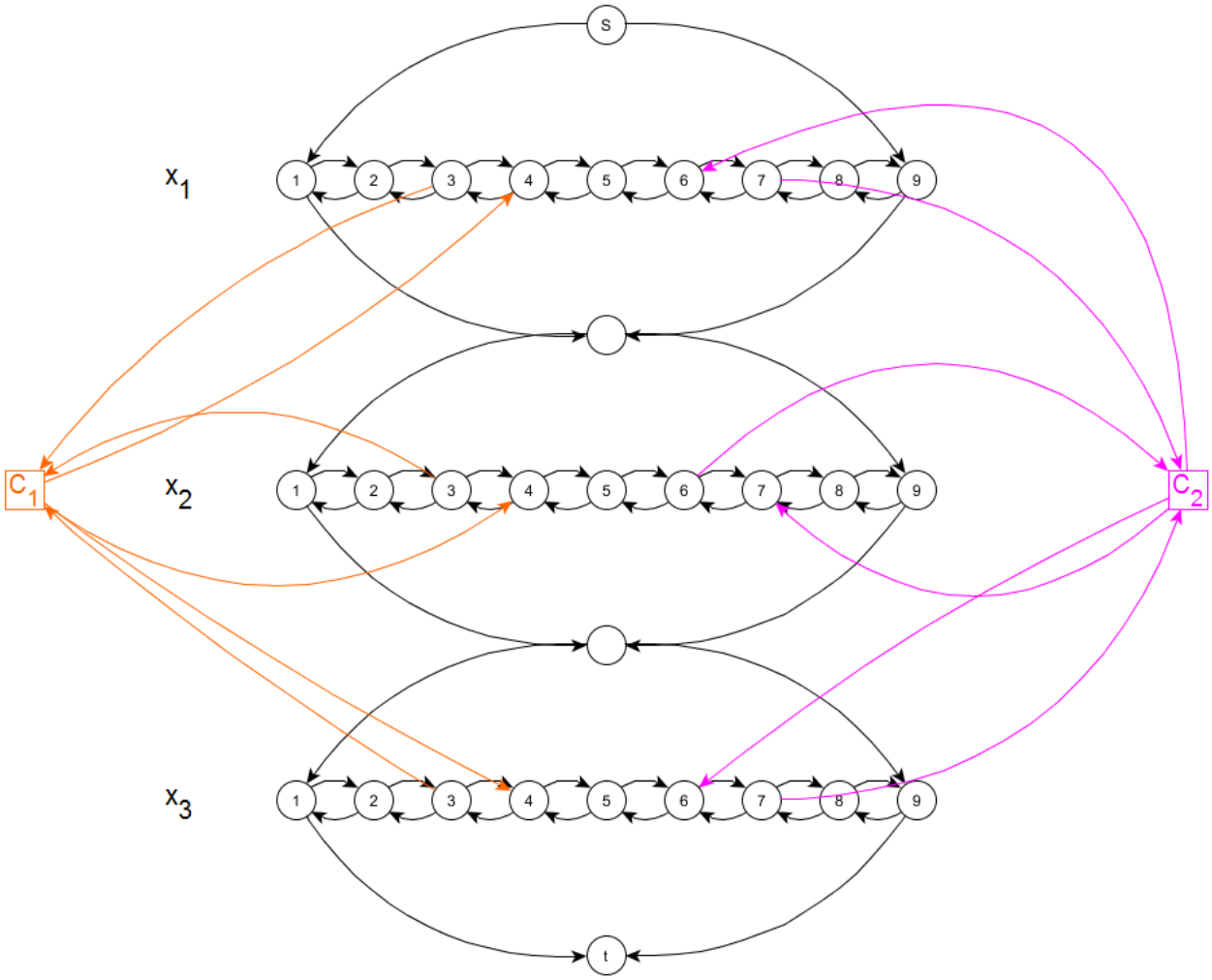

Abbildung 4.4: Graph G' für 3-SAT Formel $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$

- Gehe von R nach L, wenn x_i = falsch.

Da φ erfüllbar, enthält jede Klausel C_j mindestens ein wahres Literal. Nimm dabei das erste wahre Literal auf dem Weg zu C_j. Dabei können zwei Fälle auftreten, die in Abbildung 4.3 angedeutet sind. Entweder wir „betreten" das Literal von „links" und folgen dem Weg zu C_j (das heißt der zu C_j korrespondierende Knoten wird jetzt besucht, und danach nie wieder) und gehen weiter nach „links" oder wir „betreten" es von „rechts", besuchen dann C_j und folgen dem Weg weiter nach „rechts". In beiden Fällen ist es per Konstruktion möglich den Knoten C_j zu besuchen und dies wird insbesondere genau einmal für jedes C_j gemacht (beim ersten auftretenden wahren Literal in C_j). $\Rightarrow$ Es existiert ein Hamiltonischer Weg in G'.

(ii) Sei ein Hamiltonischer Weg in G' gegeben. Dabei gilt:

- Wenn der Hamiltonische Weg die Kante $(3j, C_j)$ in einem *gadget* x_l enthält, so muss auch die Kante $(C_j, 3j+1)$ im *gadget* x_l enthalten sein (Würde man nämlich von C_j direkt in ein anderes *gadget* x_k „springen", käme man nicht mehr zum Knoten $3j$+1 des *gadgets* für x_l zurück). (siehe dazu auch Abbildung 4.3 (a))
- Analog: Wenn der Hamiltonische Weg die Kante $(C_j, 3j+1)$ in einem *gadget* x_l enthält, so muss auch die Kante $(3j, C_j)$ aus diesem *gadget* x_l Teil des Hamiltonischen Weges sein (vergleiche dazu Abbildung 4.3 (b)).

Im ersten Fall (d.h. im *gadget* für x_l gilt: $3j \to C_j \to 3j+1$), setzen wir x_l auf wahr. Per Konstruktion wird dadurch die Klausel C_j wahr.
Im zweiten Fall (d.h. im *gadget* für x_l gilt: $3j+1 \to C_j \to 3j$), setzen wir x_l auf falsch, wodurch C_j per Konstruktion wahr wird.
$\Rightarrow$ Insgesamt erhalten wir eine Wahrheitsbelegung, die alle Klauseln und somit φ erfüllt.

□

Folgerungen

Theorem. *Das Problem Hamiltonischer Weg für ungerichtete Graphen ist $\mathcal{NP}$-vollständig.*

Beweisidee.

1. Problem $\in \mathcal{NP}$ $\surd$
2. $\mathcal{NP}$-Schwere:
   Hamiltonischer Weg Digraph (gerichtet) $\leq_p$ Hamiltonischer Weg Graph (ungerichtet).
   Sei G Digraph. Konstruiere einen ungerichteten Graphen G' aus G wie folgt:
   - Ersetze alle Knoten $v \in V(G)$ durch einen Graph bestehend aus drei Knoten v_1, v_2 und v_3 und den Kanten (v_1v_2) und (v_2v_3).
   - Für alle Kanten $(vw) \in E(G)$ (gerichtet: $v \to w$), führe Kante (v_3w_1) in G' ein.

Dann gilt: $\exists$ Hamiltonischer Weg in $G \Leftrightarrow \exists$ Hamiltonischer Weg in G'.

□

Theorem. *Das Problem Hamiltonischer Kreis ist $\mathcal{NP}$-vollständig.*

Beweisidee.

1. Problem $\in \mathcal{NP}$ $\surd$
2. $\mathcal{NP}$-Schwere:
   Hamiltonischer Weg $\leq_p$ Hamiltonischer Kreis.
   Sei $G = (V, E)$ gegeben. Konstruiere daraus $G' = (V \cup \{v\}, E \cup \{(vw), w \in V\})$. Dann gilt: $\exists$ Hamiltonischer Weg in $G \Leftrightarrow \exists$ Hamiltonischer Kreis in G'.

□

Theorem. *Das Travelling-Salesman-Problem (TSP) ist $\mathcal{NP}$-vollständig.*

Beweisidee.

1. Problem $\in \mathcal{NP}$ $\surd$
2. $\mathcal{NP}$-Schwere:
   Hamiltonischer Kreis $\leq_p$ TSP.
   Sei $G = (V, E)$ gegeben.
   Konstruiere $H = (V, \binom{V}{2}) \cong K_{|V|}$ und definiere Gewichte

$$c : E(H) \to \mathbb{R} \text{ mit } c((ij)) = \begin{cases} 1, \text{ falls } (ij) \in E(G) \\ 2, \text{ sonst} \end{cases}$$

   . Dann gilt: $\exists$ Hamiltonischer Kreis in $G \Leftrightarrow H$ hat Rundtour der Länge $|V|$.

□

Zusammenfassung Um kürzeste Wege „effizient“ zu bestimmen, sind wie beobachtet, bestimmte Gewichtsfunktionen (z.B. $c : E \to \{-1\}$, d.h. alle Kanten erhalten das Gewicht -1) oder bestimmte Anforderungen an den kürzesten Weg (z.B. TSP: Besuche alle Knoten) nicht geeignet.

Einfacher wird es, wenn man nur nicht-negative Gewichte zulässt, oder aber zumindest Kreise mit negativem Gewicht ausschließt. Dies führt zu folgender Definition:

Definition (konservativ)**.** Sei $G = (V, E)$ ein Graph (gerichtet oder ungerichtet) mit $c : E \to \mathbb{R}$. Dann heißt c *konservativ*, wenn es keinen Kreis mit negativem Gewicht in G gibt.

4.2 „Einfache“ Kürzeste-Wege-Probleme

4.2.1 Kürzeste Wege von einer Quelle

Optimalitätsprinzip von Bellman

Proposition (Optimimalitätsprinzip von Bellman)**.** *Sei G ein Digraph mit konservativer Gewichtsfunktion $c : E \to \mathbb{R}$. Weiterhin seien $s, w \in V(G)$, sowie $k \in \mathbb{N}$. Sei P_{sw} ein kürzester Weg von s nach w mit höchstens k Kanten. Die letzte Kante dieses Weges sei durch $e = (vw)$ gegeben. Dann gilt: P_{sv} ($= P_{sw}$ ohne Kante $e = (vw)$ und ohne Knoten w) ist der kürzeste Weg von s nach v mit höchstens $k - 1$ Kanten.*

Beweis. Angenommen es gibt einen Weg Q von s nach v mit $|E(Q)| \leq k - 1$, der kürzer ist als P_{sv}, also

$$c(E(Q)) + c(e) < c(E(P_{sw})).$$

Jetzt unterscheiden wir zwei Fälle:

(i) $w \notin Q$: Ergänzen wir den Weg Q um die Kante $e = (vw)$, so erhalten wir einen kürzeren Weg von s nach w als durch P_{sw} ↯

(ii) $w \in Q$: Betrachte Teilweg Q_{sw} von Q.
$\Rightarrow Q_{sw}$ hat Länge

$$\begin{aligned} c(E(Q_{sw})) &= c(E(Q)) - c(E(Q_{wv})) \\ &= c(E(Q)) + c(e) - c(E(Q_{wv} \cup \{e\}) \\ &< c(E(P_{sw})) - c(E(Q_{wv} \cup \{e\})) \\ &\leq c(E(P_{sw})) \qquad // \text{ da } c \text{ konservativ und } E(Q_{wv} \cup \{e\} \text{ ein Kreis ist} \end{aligned}$$

Also: $c(E(Q_{sw})) < c(E(P_{sw}))$ ↯

In beiden Fällen erhalten wir einen Widerspruch zu der Annahme, dass P_{sw} kürzester Weg von s nach w ist. Somit muss P_{sv} auch der kürzeste Weg von s nach v sein. □

Kreisfreie Graphen (DAG)

Theorem. *Ein Digraph G ist kreisfrei $\Leftrightarrow$ Knoten von $V(G)$ können von 1 bis $V(G)$ enumeriert werden, sodass für alle Kanten $(ij) \in E(G)$ gilt $i < j$. Man nennt diese Enumeration auch „Topologische Ordnung“.*

Beweisidee.

„$\Leftarrow$“ Angenommen G ist nicht kreisfrei, d.h. es existiert ein Kreis in G. Es ist klar, dass in diesem Fall keine topologische Ordnung möglich ist.

„$\Rightarrow$“ Diese Richtung kann man durch Induktion über die Knotenanzahl zeigen. Falls $|V| = 1$, existiert offensichtlich eine topologische Ordnung. Die Induktionsannahme lautet nun,

dass die Behauptung für Graphen mit $|V| = n$ gilt. Jetzt betrachten wir einen kreisfreien Graphen G mit $|V(G)| = n+1$. Dieser muss (aufgrund der Kreisfreiheit) einen Knoten v mit $deg^+(v) = 0$ (Ausgangsgrad 0, d.h. es gibt keine Kanten der Form (v, w)) enthalten. Wir entfernen den Knoten v und alle inzidenten Kanten aus G und erhalten einen Graphen G' mit $|V(G')| = n$, der nach Induktionsannahme eine topologische Ordnung besitzt. Wir können diese topologische Ordnung auf die Knoten von G übertragen und geben dem Knoten v die Nummerierung $n+1$ und erhalten so eine topologische Ordnung für G.

□

Algorithmus Mittels des Optimalitätsprinzips von Bellman können wir für kreisfreie Graphen direkt folgenden Algorithmus angeben, um die kürzesten Wege von einem Startknoten s zu allen anderen Knoten zu berechnen. Wir nutzen dazu einen dynamischen Programmieransatz und verwenden folgende Rekursion:

Setze $d(s, s) = 0$

$$d(s, w) = \min\{d(s, v) + c(v, w) \,:\, (vw) \in E(G)\} \text{ für } w \in V \setminus \{s\}$$

Anmerkung. Das Optimialitätsprinzip von Bellman liefert auch den Grund, warum die meisten Algorithmen kürzeste Wege von s zu allen anderen Knoten bestimmen. Bei der Berechnung eines kürzesten $s-t-$Wegs P hat man nämlich bereits einen kürzesten $s-v-$Weg für jeden Knoten v auf P berechnet. Da man im Voraus nicht weiß, welche Knoten auf P liegen, ist es natürlich, dass man kürzeste $s-v-$Wege für alle v berechnet. Diese lassen sich effizient speichern, indem man die jeweils letzte Kante speichert.

Allgemeine Digraphen – Dijkstra-Algorithmus

Von den kreisfreien Digraphen kommen wir nun zu allgemeinen Digraphen. Einer der bekanntesten Algorithmen um in einem allgemeinen Digraphen, kürzeste Wege von einem Startknoten s zu allen anderen Knoten zu bestimmen, ist der sogenannte *Dijkstra*-Algorithmus, den wir im Folgenden einführen:

Beispiel (Anwendung des Dijkstra-Algorithmus)**.**
Betrachte Abbildung 4.5 und Tabelle 4.1.

Algorithm 4 Dijkstra (Digraph $G = (V, E)$, Gewichtsfunktion $c : E \to \mathbb{R}^+$, Startknoten $s \in V$)

```
1:  Q ← V
2:  l(v) ← ∞ ∀v ∈ V                    ▷ Länge des kürzesten Wegs von s nach v
3:  pred(v) ← nil ∀v ∈ V               ▷ (pred(v), v) ist Kante des kürzesten Wegs
4:  l(s) ← 0
5:
6:  while Q ≠ ∅ do
7:      u ← Knoten aus Q mit l(u) = min_{w∈Q} l(w)
8:      Q ← Q \ {u}
9:      for ∀ (uv) ∈ E mit v ∈ Q do
10:         if l(u) + c(uv) < l(v) then
11:             l(v) ← l(u) + c(uv)
12:             pred(v) ← u
13:         end if
14:     end for
15: end while
16:
17: return Kürzeste Wege von s zu allen anderen Knoten sowie deren Länge, genauer: l(v) =
    Länge des kürzesten s−v−Wegs, welcher wiederum aus einem kürzesten s−pred(v)−Weg
    sowie der Kante (pred(v), v) besteht. Wenn v von s nicht erreichbar, dann l(v) = ∞ und
    pred(v) = nil.
```

Tabelle 4.1: Durchlauf Dijkstra-Algorithmus

Schritt	Q	Knoten 1	Knoten 2	Knoten 3	Knoten 4	Knoten 5
Init	V	$l(1) = 0$ $p(1) = -$	$l(2) = \infty$ $p(2) = -$	$l(3) = \infty$ $p(3) = -$	$l(4) = \infty$ $p(4) = -$	$l(5) = \infty$ $p(5) = -$
1 $(u = 1)$	$\{2, 3, 4, 5\}$	$l(1) = 0$ $p(1) = -$	$l(2) = 1$ $p(2) = 1$	$l(3) = 5$ $p(3) = 1$	$l(4) = \infty$ $p(4) = -$	$l(5) = \infty$ $p(5) = -$
2 $(u = 2)$	$\{3, 4, 5\}$	$l(1) = 0$ $p(1) = -$	$l(2) = 1$ $p(2) = 1$	$l(3) = 4$ $p(3) = 2$	$l(4) = 3$ $p(4) = 2$	$l(5) = \infty$ $p(5) = -$
3 $(u = 4)$	$\{3, 5\}$	$l(1) = 0$ $p(1) = -$	$l(2) = 1$ $p(2) = 1$	$l(3) = 4$ $p(3) = 2$	$l(4) = 3$ $p(4) = 2$	$l(5) = \infty$ $p(5) = -$
4 $(u = 3)$	$\{5\}$	$l(1) = 0$ $p(1) = -$	$l(2) = 1$ $p(2) = 1$	$l(3) = 4$ $p(3) = 2$	$l(4) = 3$ $p(4) = 2$	$l(5) = 5$ $p(5) = 3$
5 $(u = 5)$	$\{\}$	$l(1) = 0$ $p(1) = -$	$l(2) = 1$ $p(2) = 1$	$l(3) = 4$ $p(3) = 2$	$l(4) = 3$ $p(4) = 2$	$l(5) = 5$ $p(5) = 3$

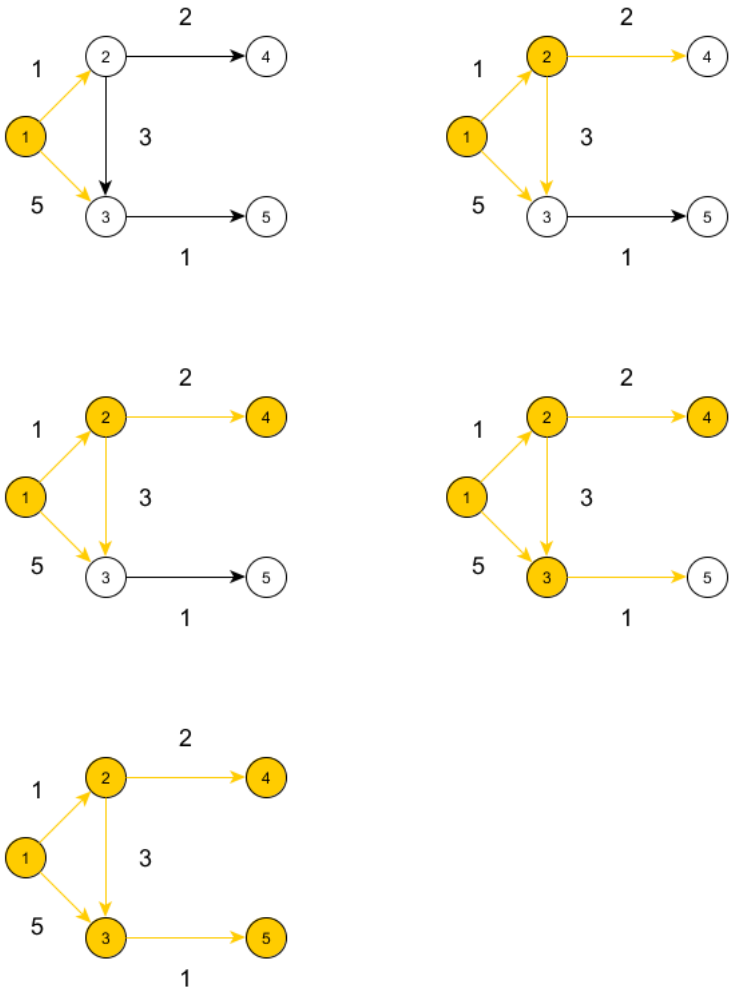

Abbildung 4.5: Beispiel Dijkstra-Algorithmus

Lemma. *Dijkstra ist korrekt und hat Laufzeit* $\mathcal{O}(|V|^2)$.

Beweisidee.

- Laufzeit klar
- Korrektheit $\rightarrow$ Induktion über „besuchte Knoten“ $V \setminus Q$:
  Wir müssen zeigen:
  (i) Für alle $u \in V \setminus Q$ gilt $l(u) = dist(s, u)$
  (ii) Für alle $v \in Q$ gilt: $l(v)$ ist die kürzeste Distanz von v zu s über schon besuchte Knoten (wenn so ein Weg existiert; Achtung: Hier werden nur schon besuchte Knoten verwendet, d.h. $l(v)$ muss nicht die kürzeste Distanz bezogen auf den gesamten Graphen G sein).

  Induktionsanfang: $V = \{s\}$ $\surd$

  Induktionsvoraussetzung: Dijkstra ist korrekt für $n-1$ besuchte Knoten, d.h. (i) und (ii) sind erfüllt.

  Induktionsschritt: Betrachte nun n-ten Knoten $u \in Q$ mit $l(u) = \min_{w \in Q} l(w)$. Für einen Beweis durch Widerspruch nehmen wir an, dass es einen $s-u-$Weg P der Länge $< l(u)$ gibt. Sei y der erste Knoten auf P (von s nach u), sodass $y \in Q \cup \{u\}$. Ferner sei x der Vorgänger von y, d.h. die Kante $(xy) \in E$ sei Teil von P. Da aber y der erste Knoten auf P mit $y \in Q \cup \{u\}$, folgt dass $x \notin Q$, also insbesondere $x \in V \setminus Q$.
  Aus $x \in V \setminus Q$ und der Induktionsvoraussetzung folgt nun:

$$\begin{aligned} l(y) &\overset{(ii)}{\leq} l(x) + c((xy)) \\ &\overset{(i)}{=} dist(s, x) + c((xy)) \\ &\leq c(E(P_{sy})) \\ &\leq c(E(P_{su})) = c(E(P)) \\ &< l(u) \end{aligned}$$

  Also insgesamt $l(y) < l(u)$ $\lightning$
  Dies ist ein Widerspruch zur Wahl von u mit $l(u) = \min_{w \in Q} l(w)$.

□

Anmerkungen.

- Der Dijsktra-Algorithmus kann so implementiert werden, dass er Laufzeit $\mathcal{O}(|V| \log(V) + |E|)$ hat. Dies ist dann vorteilhaft, wenn $|E| << |V|^2$. Graphen mit dieser Eigenschaft bezeichnet man auch als „dünne Graphen“.

- Während wir für den Dijkstra-Algorithmus explizit eine positive Gewichtsfunktion $c : E \to \mathbb{R}^+$ vorausgesetzt haben, ist der beste bekannte Algorithmus für allgemeine Digraphen mit konservativer Gewichtsfunktion der sogenannte *MOORE-BELLMAN-FORD*-Algorithmus mit Laufzeit $\mathcal{O}(|V||E|)$.

4.2.2 Kürzeste Wege zwischen allen Knotenpaaren

Im Folgenden wollen wir erneut einen Dynamischen Programmieransatz nutzen, um kürzeste Wege für alle Knoten $s, t \in V$ eines Graphen mit konservativer Gewichtsfunktion $c : E \to \mathbb{R}$ zu bestimmen.

Um einen Algorithmus herzuleiten, betrachten wir zunächst die Struktur kürzester Wege.

Struktur kürzester Wege

Definition (innerer Knoten)**.** Sei $Q = (v_1 v_2 \dots v_{l-1} v_l)$ ein Weg von v_1 nach v_l. Als *innere Knoten* von Q bezeichnen wir alle Knoten $x \in Q$ mit $x \neq v_1$ und $x \neq v_l$.

Herleitung des FLOYD-WARSHALL-Algorithmus Sei nun also ein Graph $G = (V, E)$ gegeben mit $V = \{1, \dots, n\}$. Ferner sei $V_k = \{1, \dots, k\} \subseteq V$ die Menge der ersten k Knoten von V und sei $V_0 = \emptyset$. Für zwei Knoten $i, j \in V$ betrachten wir nun alle Wege von i nach j, die ausschließlich Knoten aus V_k als innere Knoten besitzen. Sei P einer dieser Wege mit minimaler Länge (Gewicht). Der folgende Algorithmus nutzt den Zusammenhang zwischen dem Weg P und dem kürzesten Weg von i nach j, der nur innere Knoten aus V_{k-1} enthält. Hierbei lassen sich zwei Fälle unterscheiden:

- k ist kein innerer Knoten von P:
  $\Rightarrow$ alle inneren Knoten von P stammen aus V_{k-1}
  $\Rightarrow$ ein kürzester Weg von i nach j mit allen inneren Knoten aus V_{k-1} ist damit auch ein kürzester Weg von i nach j mit allen inneren Knoten aus V_k

- k ist innerer Knoten von P:
  $\Rightarrow$ P lässt sich unterteilen in P_1 (Weg von i nach k) und P_2 (Weg von k nach j). $\overset{Bellman}{\Rightarrow}$
  P_1 ist kürzester Weg von i nach k mit inneren Knoten nur aus V_{k-1} und analog: P_2 ist kürzester Weg von k nach j mit inneren Knoten nur aus V_{k-1}.

Diese Beobachtung erlaubt folgende rekursive Formulierung des Kürzeste-Wege-Problems zwischen allen Knotenpaaren $i, j \in V$:

Sei $d_{ij}^{(k)}$ = Gewicht des kürzesten Wegs von i nach j, der nur innere Knoten aus V_k hat.
Setze

$$d_{ij}^{(0)} = \begin{cases} c(ij), \text{ wenn } (ij) \in E \, (i \neq j) \\ \infty, \text{ sonst} \end{cases}$$

($k = 0$ bedeutet, dass der Weg von i nach j keine inneren Knoten hat, d.h. wenn er existiert, besteht er nur aus der Kante (ij).)
Den obigen Überlegungen folgend, ergibt sich $d_{ij}^{(k)}$ nun rekursiv als

$$d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$$

Da für alle Wege in $G = (V, E)$ gilt, dass alle inneren Knoten aus $V = V_n$ kommen, ist $d_{ij}^{(n)}$ somit die Länge des kürzesten Weges von i nach j in G.

Im Folgenden geben wir nun noch den sogenannten FLOYD-WARSHALL-ALGORITHMUS an, der genau diese Methode nutzt, um kürzeste Wege zwischen allen Knotenpaaren $i, j \in V$ zu bestimmen.

Algorithm 5 FLOYD-WARSHALL-ALGORITHMUS ($G = (V, E)$, $V = \{1, \ldots, n\}$, konservative Gewichtsfunktion $c : E \to \mathbb{R}$)

$d_{ij}^{(0)} = \begin{cases} c(ij), \text{ wenn } (ij) \in E\,(i \neq j) \\ \infty, \text{ sonst} \end{cases}$
$d_{ii}^{(0)} = 0$
$p_{ij} = \begin{cases} i\ \forall (ij) \in E \\ nil, \text{ sonst} \end{cases}$ ▷ Vorgänger

for $k = 1, \ldots, n$ **do**
    Definiere neue Matrix $\left(d_{ij}^{(k)}\right)_{1 \leq i,j \leq n}$
    for $i = 1, \ldots, n$ **do**
        for $j = 1, \ldots, n$ **do**
            $d_{ij}^{(k)} \leftarrow \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$
            $p_{ij} \leftarrow p_{kj}$ ▷ p_{ij} wird durch p_{kj} ersetzt, wenn besserer Weg von i nach j über k läuft.
        end for
    end for
end for
return Matrizen $\left(d_{ij}^{(n)}\right)_{1 \leq i,j \leq n}$ und $\left(p_{ik}\right)_{1 \leq i,j \leq n}$, wobei
(p_{ij}, j) = letzte Kante auf kürzestem Weg von i nach j (falls dieser existiert)

Lemma. *Der Floyd-Warshall-Algorithmus ist korrekt und hat Laufzeit $\mathcal{O}(|V|^3)$.*

Beweis.

- Korrektheit folgt aus obigen Überlegungen zur Herleitung des Algorithmus.
- Laufzeit klar (siehe 3 for-Schleifen).

□

Zusammenfassung Wir haben in diesem Kapitel gesehen, dass Kürzeste-Wege-Probleme, abhängig von der Instanz, „einfach“ oder „schwer“ sein können und „leichte“ Modifikationen der Problemformulierung die Schwierigkeit eines Problems verändern können.

Als schwere Instanzen haben wir z.B. die Gewichtsfunktion $c : E \to \{-1\}$ für beliebige Graphen und als

leichte Instanzen z.B. kreisfreie Graphen (mit beliebiger Gewichtsfunktion) und beliebige Graphen mit konservativer Gewichtsfunktion kennengelernt.

Außerdem wurden Polynomialzeit-Algorithmen zur Bestimmung kürzester Wege vorgestellt.

5 Matroide

Wir beginnen mit einem einführenden Beispiel eines diskreten Optimierungsproblems, welches mittels GREEDY gelöst werden kann. Ziel dieses Kapitels ist es, neben der Charakterisierung von Matroiden, Bedingungen an das Optimierungsproblem herzuleiten, unter welchen GREEDY immer eine optimale Lösung liefert.

5.1 Das Minimum-Spanning-Tree Problem

Bevor wir das Problem, des Minimum-Spanning-Trees formulieren können, benötigen wir noch folgende Definitionen:

Definition (Baum)**.** Ein ungerichteter Graph $G = (V, E)$ ist ein *Baum*, wenn G zusammenhängend und azyklisch ist.

Definition (Zusammenhangskomponente)**.** Sei $G = (V, E)$ ein ungerichteter Graph. Eine *Zusammenhangskomponente* von G ist ein maximal zusammenhängender Teilgraph $H \subseteq G$, d.h. $\nexists H' \subseteq G$ mit H' zusammenhängend und $H \subset H'$.

Definition (Wald)**.** Ein *Wald* $G = (V, E)$ ist ein Graph, dessen Zusammenhangskomponenten ausschließlich Bäume sind.

Definition (Spannbaum)**.** Ein *Spannbaum* T eines Graphen $G = (V, E)$ ist ein Teilgraph $T \subseteq G$, sodass $V(T) = V$ und T ist ein Baum.

Lemma. *Folgende drei Aussagen für einen Graphen $T = (V, E)$ sind äquivalent:*

(i) T *ist ein Baum*
(ii) $\forall\, u, v \in V\ \exists!$ *Weg von* u *nach* v *in* T
(iii) T *ist zusammenhängend und* $|E| = |V| - 1$

Beweis.

(i) $\Rightarrow$ (ii) Sei $T = (V, E)$ ein Baum und somit zusammenhängend.
Also existiert mindestens ein Weg von u nach v $\forall u, v \in V$.
Angenommen es gäbe zwei unterschiedliche Wege von u nach v.
Diese müssen sich in mindestens einem Knoten unterscheiden.
Daraus folgt aber, dass T einen Kreis enthält. Widerspruch $\lightning$

(ii) $\Rightarrow$ (iii) Induktion über die Knotenanzahl $|V|$:
Induktionsanfang:

Sei $|V| = 1$, d.h. wir haben einen isolierten Knoten.
Dann gilt $|E| = 0 = |V| - 1 = 1 - 1 = 0$ $\surd$

Induktionsannahme:
Die Behauptung sei wahr für Bäume mit $|V| < k$, $k \in \mathbb{N}, k > 1$.

Induktionschritt:
Betrachte Baum T mit $|V| = k$.
Durch Entfernen einer beliebigen Kante $e = (uv) \in E$, erhalten wir einen Wald $T' = T - e$, der nicht zusammenhängend ist.
Insbesondere hat T' dann genau zwei Zusammenhangskomponenten T_1, T_2, welche Bäume sind und weniger als k Knoten besitzen.
Also gilt: $|E_1| = |V_1| - 1$ und $|E_2| = |V_2| - 1$.
Damit folgt: $|E| = |E_1| + |E_2| + 1 = |V_1| - 1 + |V_2| - 1 + 1 = |V| - 1$.

(iii) $\Rightarrow$ (i) Angenommen für T gilt $|E| = |V| - 1$ und T ist zusammenhängend, aber T ist kein Baum, d.h. T enthält einen Kreis.
Dann können wir Kanten aus diesem Kreis entfernen, sodass der modifizierte Graph $T' = (V, E')$ kreisfrei und damit ein Baum ist.
Angenommen, es wurden k Kanten aus T entfernt.
Dann gilt $|E'| = |V| - 1 - k$.
Andererseits ist T' aber ein Baum, d.h. $|E'| = |V| - 1$.
Daraus folgt $k = 0$, d.h. T enthielt keinen Kreis und ist somit ein Baum.

□

Lemma. *$T = (V, E)$ ist ein Wald mit c Zusammenhangskomponenten $\Leftrightarrow |E| = |V| - c$.*

Korollar. *Sei $G = (V, E)$ zusammenhängend. Dann folgt*

1. *$|E| \geq |V| - 1$*
2. *G hat einen Spannbaum $T \subseteq G$.*

Minimum-Spanning-Tree Problem (MST)

Definition (Minimum-Spanning-Tree Problem)**.** Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph mit Gewichtsfunktion $c : E \to \mathbb{R}$. Dann lautet das *Minimum-Spanning-Tree Problem (MST)*: Finde einen Spannbaum $T = (V, F) \subseteq G$, sodass

$$f(T) = \sum_{e \in F} c(e) \xrightarrow{!} \min .$$

Motivation Gegeben eine Menge von Punkten (z.B. Computer, Haushalte) und eine Menge von Verbindungen zwischen diesen Punkten (z.B. Stromkabel, Telefonleitungen), bedeutet das MST anschaulich, eine Verknüpfung zwischen allen Punkten mit minimalen Kosten zu finden.

Es gibt effiziente Algorithmen zur Lösung des MST-Problems, die im Wesentlichen auf folgendem Theorem beruhen.

Theorem. *Sei $G = (V, E)$ ein beliebiger zusammenhängender Graph und $\{T_1, \ldots, T_k\}$ eine Menge von Bäumen, sodass $T_i \subseteq G \,\forall\, i$ und $\bigcup_{i=1}^{k} V(T_i) = V$ („Spannwald“, T_i paarweise disjunkt). Sei $(uv) \in E$ eine Kante mit minimalem Gewicht, sodass*

- $(uv) \notin E(T_i) \,\forall\, i$
- $u \in V(T_i)$ *und* $v \in V(T_j)$ *für* $i \neq j$

Dann gilt: Unter allen Spannbäumen aus $\mathcal{T} = \left\{ T = (V, F) \text{ mit } \bigcup_{i=1}^{k} E(T_i) \subseteq F \right\}$ *gibt es einen optimalen Spannbaum (d.h. einen Spannbaum mit minimalen Kosten), der die Kante $e = (uv)$ enthält.*

Beweis. Angenommen es existiert ein Spannbaum $T = (V, F)$ von G, sodass $(uv) \notin F$ und $f(T) < f(T') \,\forall\, T' \in \mathcal{T}$, $T' \neq T$, $(uv) \in E(T')$ (d.h. T enthält die Kante (uv) nicht, hat aber ein kleineres Gewicht als alle anderen Spannbäume T' aus $\mathcal{T}$, die die Kante (uv) enthalten). Sei $u \in V(T_i)$ und $v \in V(T_j)$, also insbesondere $v \in V \setminus V(T_i)$. Nun betrachten wir den Graphen $H = (V, F \cup \{(uv)\})$. Dieser muss einen Kreis C enthalten und es existiert eine Kante $(u'v')$ in C mit $u' \in V(T_i)$ und $v' \in V \setminus V(T_i)$. Wegen Wahl von (uv) gilt dabei $c(uv) \leq c(u'v')$. Wir können nun aber $(u'v')$ aus T entfernen und stattdessen (uv) einfügen. Dadurch erhalten wir einen Spannbaum T' mit $f(T') \leq f(T)$ (da $c(uv) \leq c(u'v')$). Widerspruch ↯ □

Korollar. *Wenn T ein Minimum-Spannbaum von G ist, so gilt für alle Kanten $(xy) \in E(G) \setminus E(T)$, dass keine Kanten des $x - y-$Wegs in T ein höheres Gewicht hat als (xy).*

KRUSKAL-Algorithmus Aus obigen Betrachtungen erhalten wir nun den sogenannten *KRUSKAL*-Algorithmus zur Bestimmung von Minimum-Spannbäumen:

Algorithm 6 KRUSKAL($G = (V, E)$ ungerichtet und zusammenhängend, $c : E \to \mathbb{R}$)

```
1: Sortiere die Kanten von G, sodass c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m).
2: Setze T = (V, F) mit F = ∅.
3: for i = 1, ..., m do
4:     if T = (V, F ∪ e_i) ist kreisfrei then
5:         F ← F ∪ e_i
6:     end if
7: end for
8: return Spannbaum T von G mit minimalem Gewicht
```

Theorem. *Der KRUSKAL-Algorithmus zur Bestimmung von Minimum-Spannbäumen ist korrekt und hat Laufzeit $\mathcal{O}(|E||V|)$.*

Beweis.

Korrektheit folgt aus obigem Theorem und Korollar zur Herleitung des Algorithmus.

Für die Laufzeit betrachte die Schritte des Algorithmus:

Schritt 1: $\mathcal{O}(|E| \cdot \log |E|)$ (Laufzeit für Sortieren)

Schritt 2: $\mathcal{O}(|V|)$

Schritt 3-5: $\mathcal{O}(|E||V|)$ (Testen der Kreisfreiheit geht mittels BFS in $\mathcal{O}(|V|)$)

Insgesamt ergibt sich also eine Laufzeit von $\mathcal{O}(|E||V|)$.

□

Wir betrachten nun noch das sogenannte *Maximal-gewichteter-Wald Problem*, von dem wir später sehen werden, dass es äquivalent zum Minimum-Spannbaum Problem ist.

Definition (Maximal-gewichteter-Wald Problem (MGW))**.** Sei $G = (V, E)$ ein ungerichteter Graph mit Gewichtsfunktion $c : E \to \mathbb{R}$. Das *Maximal-gewichteter-Wald Problem* lautet dann: Finde Wald in G mit maximalem Gewicht.

Proposition. *Das Maximal-gewichteter-Wald Problem (MGW) und das Minimum-Spanning-Tree Problem (MST) sind äquivalent, d.h. jede Instanz (G, c) von MGW kann in eine Instanz (G', c') von MST überführt werden, sodass die optimale Lösung des einen Problems die optimale Lösung des anderen Problems impliziert (und umgekehrt).*

Beweis. Sei (G, c) eine Instanz von MGW. Zur Vereinfachung betrachten wir positive Gewichtsfunktionen $c : E(G) \to \mathbb{R}^+$ (für $c : E(G) \to \mathbb{R}$ erfolgt der Beweis analog, indem negativ gewichtete Kanten ignoriert werden). Wir nehmen weiter an, dass G zusammenhängend ist (sonst wende die folgenden Argumente auf die Zusammenhangskomponenten an). Da $c(xy) \geq 0 \, \forall \, (xy) \in E(G)$, ist jeder maximal-gewichtete Wald ein Baum. Wir wissen außerdem, dass alle Spannbäume von G genau $|V(G)| - 1$ Kanten enthalten.Nun konstruieren wir eine Gewichtsfunktion c' für $E(G)$ wie folgt:

$$c'(xy) = K - c(xy) \text{ für alle } (xy) \in E(G), \text{ wobei } K = \max_{(xy) \in E} c(xy)$$

Nun betrachten wir das MST bezüglich c'. Sei also T ein Spannbaum von G. Dann ist die Summe der Gewichte

$$f'(T) = \sum_{(xy) \in E(T)} c'(xy) \text{ und } f(T) = \sum_{(xy) \in E(T)} c(xy).$$

Damit folgt

$$\begin{aligned} f(T) &= \sum_{(xy) \in E(T)} K - c'(xy) \\ &= \sum_{(xy) \in E(T)} K - \sum_{(xy) \in E(T)} c'(xy) \\ &= K \cdot (|V(G)| - 1) - f'(T) \end{aligned}$$

Somit impliziert eine Lösung von MGW ist immer eine Lösung von MST. Rückrichtung analog.

□

Anwendbarkeit von Greedy Der KRUSKAL-Algorithmus für MST ist ein Greedy-Algorithmus. Wie wir bereits gesehen haben, zeichnen sich Greedy-Algorithmen dadurch aus, dass sie schrittweise den Folgezustand wählen, der zum Zeitpunkt der Wahl den größten Gewinn bzw. das optimale Ergebnis liefert. Greedy-Algorithmen sind oft schnell, aber nicht auf alle Probleme anwendbar. Im Folgenden werden wir uns daher die Frage stellen, für welche Probleme Greedy immer funktioniert. D.h. wir wollen Probleme charakterisieren, für die Greedy-Algorithmen immer die optimale Lösung liefern.

5.2 Theorie der Matroide

5.2.1 Einführung

Viele kombinatorische Optimierungsprobleme können folgendermaßen definiert werden:

Definition (Kombinatorisches Optimierungsproblem)**.** Sei E eine endliche Menge und $\mathbb{F} \subseteq \mathbb{P}(E)$ eine Teilmenge der Potenzmenge von E. Für das Mengensystem $(E, \mathbb{F})$ und eine Kostenfunktion $c : \mathbb{F} \to \mathbb{R}$ bestimme man ein Element aus $\mathbb{F}$ mit minimalen oder maximalen Kosten.

Anmerkung. Hier betrachten wir Kostenfunktionen $c : E \to \mathbb{R}$, sodass $\forall\, X \subseteq E$ gilt: $c(X) = \sum_{x \in X} c(x)$

Definition (Unabhängigkeitssystem)**.** Ein Mengensystem $(E, \mathbb{F})$ heißt *Unabhängigkeitssystem* (manchmal auch Teilmengensystem genannt), falls gilt

(M1) $\emptyset \in \mathbb{F}$

(M2) $X \subseteq Y \in \mathbb{F} \Rightarrow X \in \mathbb{F}$

Die Elemente aus $\mathbb{F}$ heißen *unabhängig*, die Elemente aus $\mathbb{P}(E) \setminus \mathbb{F}$ *abhängig*.

Definition ((inklusions)-minimal/-maximal)**.** Sei $\mathbb{M} \subseteq \mathbb{P}(E)$. Dann heißt $M \in \mathbb{M}$

(inklusions)-minimal, wenn $\nexists\, M' \in \mathbb{M}$, sodass $M' \subset M$.

(inklusions)-maximal, wenn $\nexists\, M' \in \mathbb{M}$, sodass $M \subset M'$.

Definition (Kreise und Basen)**.** Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem. Dann heißen

minimale abhängige Mengen *Kreise* und

maximale unabhängige Mengen *Basen*.

Beispiel (0)**.** Betrachte die Matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

und definiere das Unabhängigkeitssystem $(E, \mathbb{F})$, wobei $E = \{1, 2, 3, 4, 5\}$ die Menge der Spaltenindizes (von A) sei und $\mathbb{F}$ eine Teilmenge der Potenzmenge $\mathbb{P}(E)$, sodass die zu den Spaltenindizes gehörigen Vektoren linear unabhängig sind, d.h.

$$\begin{aligned}\mathbb{F} &= \{F \subseteq E : \text{ Vektoren bezüglich der Spaltenindizes sind linear unabhängig}\} \\ &= \{\emptyset, \{1\}, \{2\}, \{4\}, \{5\}, \{1,2\}, \{1,5\}, \{2,4\}, \{2,5\}, \{4,5\}\}\end{aligned}$$

Kreise sind in diesem Fall die Mengen $\{3\}, \{1,4\}, \{1,2,5\}, \{2,4,5\}$ und *Basen* alle $B \in \mathbb{F}$ mit $|B| = 2$.

Im Folgenden betrachten wir nun Maximierungs- und Minimierungsprobleme für Unabhängigkeitssysteme:

Maximierungsproblem für Unabhängigkeitssysteme
Gegeben: $(E, \mathbb{F})$ Unabhängigkeitssystem und $c : E \to \mathbb{R}$ Gewichtsfunktion.
Ziel: Bestimme $X \in \mathbb{F}$, sodass $c(X) = \sum_{e \in X} c(e) \stackrel{!}{\to} \max$.

Minimierungsproblem für Unabhängigkeitssysteme
Gegeben: $(E, \mathbb{F})$ Unabhängigkeitssystem und $c : E \to \mathbb{R}$ Gewichtsfunktion.
Ziel: Bestimme Basis B von $\mathbb{F}$, sodass $c(B) = \sum_{e \in B} c(e) \stackrel{!}{\to} \min$.

Anmerkung. Oft sind nur E und c spezifiziert, während $\mathbb{F}$ nicht explizit angegeben wird. Deswegen nutzt man ein „Unabhängigkeitsorakel“ welches $\mathbb{F}$ vorgibt. In der Regel kann man effizient testen ob $X \in \mathbb{F}$ oder nicht.

Beispiele.

(1) TSP
Gegeben: vollständiger Graph G und Gewichtsfunktion $c : E(G) \to \mathbb{R}^+$
Ziel: Bestimme einen hamiltonischen Kreis minimalen Gewichts in G
$\Rightarrow E = E(G)$, $\mathbb{F} = \{F \subseteq E \,|\, F$ ist Teilmenge der Kanten eines hamiltonischen Kreises $\}$.

(2) Kürzeste-Wege Probleme
Gegeben: Graph G, Gewichtsfunktion $c : E(G) \to \mathbb{R}$ und Knoten $s, t \in V(G)$.
Ziel: Bestimme kürzesten s-t-Weg W_{st} in G.
$\Rightarrow E = E(G)$, $\mathbb{F} = \{F \subseteq E \,|\, F$ ist Teilmenge der Kanten eines s-t-Weges $\}$.

(3) Rucksack-Problem
Gegeben: Gegenstände $1, \ldots, n$ mit Volumen w_i und Wert/Preis $b_i, i = 1, \ldots, n$ sowie Gewichtsschranke W (Kapazität des Rucksacks)
Ziel: Finde Teilmenge $S \subseteq \{1, \ldots, n\}$, sodass $\sum_{s \in S} w_s \leq W$ und $f(S) = \sum_{s \in S} b_s \stackrel{!}{\to} \max$.
$\Rightarrow E = \{1, \ldots, n\}$, $\mathbb{F} = \{F \subseteq E \,|\, \sum_{j \in F} w_j \leq W\}$.

(4) Minimum-Spanning-Tree Problem
Gegeben: Graph G, Gewichtsfunktion $c : E(G) \to \mathbb{R}$.
Ziel: Bestimme MST
$\Rightarrow E = E(G), \mathbb{F} = \{F \subseteq E \,|\, F$ ist Kantenmenge eines Walds $\}$.

(5) Maximal-gewichteter-Wald Problem
Gegeben: Graph G, Gewichtsfunktion $c : E(G) \to \mathbb{R}$.
Ziel: Bestimme MGW
$\Rightarrow (E, \mathbb{F})$ wie in Beispiel (4).

(6) Maximum-weight matching
Gegeben: Graph G, Gewichtsfunktion $c : E(G) \to \mathbb{R}$.
Ziel: Bestimme ein matching von G mit maximalem Gewicht.
$\Rightarrow E = E(G), \mathbb{F} = \{F \subseteq E \,|\,$ Elemente aus F sind disjunkt $\}$

(7) Betrachte Abbildung 5.1.

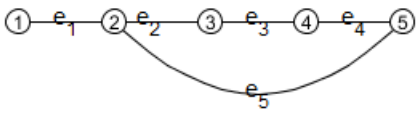

Abbildung 5.1: Beispiel (7)

$E = E(G)$,
$\mathbb{F}_G = \{\text{Menge der Wälder}\} = \mathbb{P}(E) \setminus \{\underbrace{\{e_2, e_3, e_4, e_5\}}_{\text{Kreis}}, \underbrace{\{e_1, e_2, e_3, e_4, e_5\}}_{\text{kein Kreis, aber abhängig}}\}$
Wir betrachten nun die Inzidenzmatrix zu G, wobei die Inzidenzmatrix eines Graphen eine $|V(G)| \times |E(G)|$ Matrix A ist mit

$$A_{ij} = \begin{cases} 1, & \text{wenn Knoten } i \text{ inzident zu Kante } j \\ 0, & \text{sonst} \end{cases}$$

D.h. in unserem Fall erhalten wir

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Betrachten wir A nun als Matrix über $\mathbb{Z}_2$, so erhalten wir mit der üblichen Vektoraddition $\oplus$ und Multiplikation $\odot$ in $\mathbb{Z}_2$: $e_2 \oplus e_3 \oplus e_4 \oplus e_5 = \vec{0}$ und analog $(0 \odot e_1) \oplus e_2 \oplus e_3 \oplus e_4 \oplus e_5 = \vec{0}$.

Die linear (un)abhängigen Vektoren (Spalten von A) und somit die (un)abhängigen Mengen (wie in Beispiel (0)) entsprechen genau den (un)abhängigen Mengen von $(E(G), \mathbb{F}_G)$.

Definition (Matroid)**.** Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem. Dann ist $(E, \mathbb{F})$ ein *Matroid*, wenn gilt:

(M3) $X, Y \in \mathbb{F}$ mit $|X| > |Y| \Rightarrow \exists\, x \in X \setminus Y$, sodass $Y \cup \{x\} \in \mathbb{F}$ („Austauscheigenschaft“)

Beispiele.

- Beispiel (0),(4),(5),(7)
- Triviale Matroide: $(\emptyset, \{\emptyset\})$ oder $(E, \mathbb{P}(E))$

Proposition 5.1. *Folgende Unabhängigkeitssysteme $(E, \mathbb{F})$ sind Matroide:*

(1) E = *Menge der Spaltenindizes einer Matrix A über einem Körper $\mathbb{K}$*
$\mathbb{F} = \{F \subseteq E : \textit{die Elemente in } F \textit{ sind linear unabhängig bzgl. } \mathbb{K}\}$

(2) E = *Kantenmenge eines ungerichteten Graphen G*
$\mathbb{F} = \{F \subseteq E : (V(G), F) \textit{ ist Wald }\}$

(3) E = *endliche Menge,* $k \in \mathbb{N}_0$
$\mathbb{F} = \{F \subseteq E : |F| \leq k\}$

Beweis. (M1) und (M2) sind klar, aber (M3) ist zu zeigen:

- (M3) für (1) folgt aus dem „Steinitz'schen Austauschsatz“ (Lineare Algebra).
- (M3) für (2):
  Seien $X, Y \in \mathbb{F}$ mit $|X| > |Y|$.
  Annahme: Für alle $x \in X \setminus Y$ gilt $Y \cup \{x\} \notin \mathbb{F}$, d.h. $(V, Y \cup \{x\})$ ist kein Wald.
  Wir wissen, dass (V, Y) ein Wald ist. Da $(V, Y \cup \{x\})$ kein Wald ist, muss es in genau einer Zusammenhangskomponente von $(V, Y \cup \{x\})$ einen Kreis geben.
  Also, für jede Kante $x = (uv) \in X$ liegen die Knoten u und v in der gleichen Zusammenhangskomponente von (V, Y). D.h. jede Zusammenhangskomponente von (V, X) ist somit Teilmenge einer Zusammenhangskomponente von (V, Y). Sei nun p die Anzahl der Zusammenhangskomponenten von (V, X) und q die Anzahl der Zusammenhangskomponenten von (V, Y). Dann folgt $p \geq q$.
  Wir wissen aber, dass für einen Wald gilt:
  $|E(\text{Wald})| = |V(\text{Wald})| -$ # Zusammenhangskomponenten, also insbesondere
  $|X| = |V| - p$ und $|Y| = |V| - q$. Daraus folgt
  $|V| - |X| = p \geq q = |V| - |Y|$, d.h. $|Y| \geq |X|$ ↯
- (M3) für (3): Übungsaufgabe

□

Anmerkung. Matroide aus Prop. 5.1 der Form

(1) heißen *Vektormatroide*.

(2) heißen *Kreismatroide* (Wenn G Schleifen enthalten darf, d.h. Kanten (xx), so spricht man auch von einem *graphischen Matroid*).

(3) heißen *uniforme Matroide*.

Anmerkung. Aus obigen Beispielen (1)-(6) sind nur (4) (MST) und (5) (MGW) Matroide. (Beweis: Übungsaufgabe)

Theorem 5.2. *Alle Basiselemente eines Matroids $(E, \mathbb{F})$ haben die gleiche Kardinalität.*

Beweis. Angenommen es existieren Basen $B, B' \in \mathbb{F}$ (maximal unabhängig), sodass $|B| < |B'|$. $\overset{(M3)}{\Rightarrow} \exists\, x \in B' \setminus B$, sodass $B \cup \{x\} \in \mathbb{F}$ ↯ □

Theorem 5.3. *Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem. Dann sind folgende Aussagen äquivalent:*

(M3) $\forall\, X, Y \in \mathbb{F}$ *mit* $|X| > |Y| \Rightarrow \exists\, x \in X \setminus Y$, *sodass* $Y \cup \{x\} \in \mathbb{F}$.
(M3') $\forall\, X, Y \in \mathbb{F}$ *mit* $|X| = |Y| + 1 \Rightarrow \exists\, x \in X \setminus Y$, *sodass* $Y \cup \{x\} \in \mathbb{F}$.
(M3") $\forall\, A \subseteq E$ *gilt, dass alle Teilmengen von A, welche maximal unabhängig sind ($X \subseteq A$, $X \in \mathbb{F}$, X (inklusions)-maximal bzgl. $\mathbb{F}$), die gleiche Kardinalität haben.*

Beweis.

(M3) ⇒ (M3') $\forall\, X, Y \in \mathbb{F}$ mit $|X| = |Y| + 1 \Rightarrow |X| > |Y| \Rightarrow$ Behauptung

(M3') ⇒ (M3") Angenommen (M3') gilt, aber (M3") gilt nicht.
$\Rightarrow \exists\, X, Y \subseteq A \subseteq E$, sodass X, Y maximal unabhängig und $|X| \geq |Y| + 1$.
Sei $\tilde{X} \subseteq X$ mit $|\tilde{X}| = |Y| + 1$.
$\overset{(M3')}{\Rightarrow} \exists\, x \in \tilde{X} \setminus Y$, sodass $Y \cup \{x\} \in \mathbb{F}$.
↯Widerspruch zur Wahl von Y als maximal unabhängige Teilmenge von A, da $Y \cup \{x\} \in A$.

(M3") ⇒ (M3) Seien $X, Y \in \mathbb{F}$ mit $|X| > |Y|$.
Setze $A = X \cup Y$.
Da wegen (M3") alle maximal unabhängigen Teilmengen von A gleiche Kardinalität haben, kann Y nicht maximal unabhängig sein.
$\Rightarrow \exists\, x \in A \setminus Y = (X \cup Y) \setminus Y = X \setminus Y \subseteq A$, sodass $Y \cup \{x\} \in \mathbb{F}$.

□

5.2.2 Andere Matroidaxiome

Im Folgenden werden wir Matroide über ihre Basen, Kreise und ihre Rang-Funktion definieren.

Basis

Die Basis eines Matroids $(E, \mathbb{F})$ ist die Menge der maximalen Elemente aus $\mathbb{F}$.
Sei $\mathbb{B}$ die Menge der Basen von $(E, \mathbb{F})$.
Wenn E und $\mathbb{B}$ bekannt sind, ist auch $\mathbb{F}$ bekannt, denn wir können $\mathbb{F}$ als die Menge des zu $\mathbb{B}$ gehörigen Unabhängigkeitssystems setzen, d.h. $\mathbb{F} = \{F \subseteq B : B \in \mathbb{B}\}$.
Frage: Wie muss $\mathbb{B}$ aussehen, sodass das resultierende Paar $(E, \mathbb{F})$ ein Matroid ist?

Kreis

Kreise C sind minimal abhängige Mengen von E, d.h. $C - e \in \mathbb{F} \, \forall e \in C$.
Wenn die Menge $\mathbb{C}$ der Kreise eines Matroids $(E, \mathbb{F})$ bekannt ist, so erhalten wir

$$\begin{aligned} \mathbb{F} &= \{F \subseteq E : C \not\subseteq F \, \forall C \in \mathbb{C}\} \\ &= \{F \subseteq E : C \setminus F \neq \emptyset \, \forall C \in \mathbb{C}\} \end{aligned}$$

Frage: Wie muss $\mathbb{C}$ aussehen, sodass das resultierende Paar $(E, \mathbb{F})$ ein Matroid ist?

Rang und unterer Rang

Für eine endliche Menge E ist die Rang-Funktion gegeben als $r : \mathbb{P}(E) \to \mathbb{Z}^+$.
Für ein Unabhängigkeitssystem $(E, \mathbb{F})$ sei der Rang von $X \subseteq E$ definiert als

$$r(X) \coloneqq \max\{|Y| : Y \subseteq X, Y \in \mathbb{F}\}.$$

Wenn die Rang-Funktion r des Matroids bekannt ist, setzen wir $\mathbb{F} = \{X \subseteq E : r(X) = |X|\}$.
Frage: Wie muss die Rang-Funktion $r : \mathbb{P}(E) \to \mathbb{Z}^+$ aussehen, sodass das resultierende Paar $(E, \mathbb{F})$ ein Matroid ist?

Für spätere Betrachtungen definieren wir noch die *untere Rangfunktion* $\rho : \mathbb{P}(E) \to \mathbb{Z}^+$ für ein Unabhängigkeitssystem $(E, \mathbb{F})$ für alle $X \subseteq E$ wie folgt:

$$\rho(X) := \min\{|Y| : Y \subseteq X, Y \in \mathbb{F} \text{ und } Y \cup \{x\} \notin \mathbb{F} \text{ für alle } x \in X \setminus Y\}.$$

Anmerkung: Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem mit entsprechender (unterer) Rangfunktion r und ρ, sowie $X \subseteq E$. Dann ist $(E, \mathbb{F}_{|x})$ mit $\mathbb{F}_{|x} = \{Y \subseteq X : Y \in \mathbb{F}\}$, auch wieder ein Unabhängigkeitssystem. Der Wert $r(X)$ der Rangfunktion von $(E, \mathbb{F})$ ist die maximale Kardinalität der Basen von $(E, \mathbb{F}_{|x})$. Der Wert $\rho(X)$ der unteren Rangfunktion von $(E, \mathbb{F})$ ist die minimale Kardinalität der Basen von $(E, \mathbb{F}_{|x})$.
Wenn $(E, \mathbb{F})$ Matroid dann ist auch $(E, \mathbb{F}_{|x})$ ein Matroid (Übungsaufgabe).

Basisaxiome

Theorem 5.4. *Sei E eine endliche Menge und $\mathbb{B} \subseteq \mathbb{P}(E)$. $\mathbb{B}$ ist genau dann die Menge der Basen eines Matroids $(E, \mathbb{F})$, wenn die folgenden Bedingungen erfüllt sind:*

(B1) $\mathbb{B} \neq \emptyset$;

(B2) Für alle $B_1, B_2 \in \mathbb{B}$ *und* $x \in B_1 \setminus B_2$ *gibt es ein* $y \in B_2 \setminus B_1$, *sodass* $(B_1 \setminus \{x\}) \cup \{y\} \in \mathbb{B}$.

Beweis.

„⇒“

(B1): gilt, da $\emptyset \in \mathbb{F}$ (nach (M1)).

(B2): Seien $B_1, B_2 \in \mathbb{B}$, $B_1 \neq B_2$. Da Basen von Matroiden die gleiche Kardinalität haben und $B_1 \neq B_3$ folgt $\exists\, x \in B_1 \setminus B_2$.
$\overset{(M2)}{\Rightarrow} B' \coloneqq B_1 \setminus \{x\} \in \mathbb{F}$ und $|B'| < |B_2| = |B_1|$.
$\overset{(M3)}{\Rightarrow} \exists\, y \in B_2 \setminus B'$, sodass $B' \cup \{y\} \in \mathbb{F}$, d.h. $(B_1 \setminus \{x\}) \cup \{y\} \in \mathbb{F}$.
Es gilt: $|(B_1 \setminus \{x\}) \cup \{y\}| = |B_1|$.
Da alle Basen eines Matroids die gleiche Kardinalität haben, ist $(B_1 \setminus \{x\}) \cup \{y\} \in \mathbb{B}$.

„⇐“

$\mathbb{B}$ erfülle die Bedingungen (B1) und (B2). Zuerst zeigen wir, dass alle Elemente aus $\mathbb{B}$ die gleiche Kardinalität haben, d.h. $\forall\, B_i, B_j \in \mathbb{B}$ gilt: $|B_i| = |B_j|$. Angenommen $\exists\, B_i, B_j \in \mathbb{B}$ mit $|B_i| > |B_j|$. Dann wählen wir unter diesen Elementen B_i und B_j so, dass $|B_i \cap B_j|$ maximal ist. oBdA seien dies B_1 und B_2, wobei $|B_1| > |B_2|$. Sei $x \in B_1 \setminus B_2$. Nach (B2) existiert ein $y \in B_2 \setminus B_1$, sodass $B' \coloneqq (B_1 \setminus \{x\}) \cup \{y\} \in \mathbb{B}$. Daraus folgt $|B' \cap B_2| > |B_1 \cap B_2|$.
↯ Widerspruch zur Maximalität von $|B_1 \cap B_2|$.

Nun setzen wir $\mathbb{F} \coloneqq \{F \subseteq B_i : B_i \in \mathbb{B}\}$ und zeigen, dass $(E, \mathbb{F})$ ein Matroid ist. Die Bedingungen (M1) und (M2) sind trivialerweise erfüllt, aber (M3)(äquivalent (M3')) ist zu zeigen.
Seien also $X, Y \in \mathbb{F}$ mit $|Y| = |X| + 1$, wobei $X \subseteq B_1$, $Y \subseteq B_2$, $B_1, B_2 \in \mathbb{B}$. D.h. X, Y, B_1 und B_2 sind von der Form:

$$
\begin{aligned}
X &= \{x_1, \ldots, x_k\} \\
B_1 &= \{x_1, \ldots, x_k, b_1, \ldots, b_q\} \\
Y &= \{y_1, \ldots, y_k, y_{k+1}\} \\
B_2 &= \{y_1, \ldots, y_k, y_{k+1}, c_1, \ldots, c_{q-1}\}
\end{aligned}
$$

Betrachte nun $B_1 \setminus \{b_q\}$. Dann existiert ein $z \in B_2$, sodass $(B_1 \setminus \{b_q\}) \cup \{z\} \in \mathbb{B}$ (Falls $b_q \in B_1 \setminus B_2$ folgt dies wegen (B2); falls $b_q \notin B_1 \setminus B_2$, ist insbesondere $b_q \in B_2$ und wir können $z \coloneqq b_q$ setzen). In beiden Fällen ist $z \notin X$, denn entweder ist $z \in B_2 \setminus B_1$ oder

$z = b_q$.
Weiterhin gilt per Konstruktion:
$X \subseteq B_1 \setminus \{b_q\} \subseteq (B_1 \setminus \{b_q\}) \cup \{z\} \in \mathbb{B}$
$\Rightarrow X \cup \{z\} \subseteq (B_1 \setminus \{b_q\}) \cup \{z\} \in \mathbb{B}$
$\Rightarrow X \cup \{z\} \in \mathbb{F}$
Wenn also $z \in Y \Rightarrow z \in Y \setminus X$ und $X \cup \{z\} \in \mathbb{F} \Rightarrow$ (M3) erfüllt.
Wenn $z \notin Y$, betrachten wir $((B_1 \setminus \{b_q\}) \cup \{z\}) \setminus \{b_{q-1}\})$.
Analog (wie oben) existiert ein $z_1 \in B_2$, sodass $((B_1 \setminus \{b_q\}) \cup \{z\}) \setminus \{b_{q-1}\}) \cup z_1 \in \mathbb{B}$.
Wenn $z_1 \in Y \Rightarrow X \cup \{z_1\} \in \mathbb{F}$
Wenn $z_1 \notin Y$, dann entferne b_{q-2} usw. Da $|\{b_1, \ldots, b_q\}| > |\{c_1, \ldots, c_{q-1}\}|$, wird nach höchstens q Schritten ein $z_i \in Y$ erreicht $\Rightarrow$ (M3') gilt.

□

Rangaxiome

Erinnerung: Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem und $X \subseteq E$. Dann ist der *Rang* definiert als $r(X) = \max\{|Y| : Y \subseteq X, Y \in \mathbb{F}\}$. Dabei gilt immer: $0 \leq r(X) \leq |X|$.

Beispiele.

- $(E, \mathbb{F})$ ist ein beliebiger Matroid:

  $r(X) = |X|$ für alle $X \in \mathbb{F}$

  $r(C) = |C| - 1$ für alle Kreise C von $(E, \mathbb{F})$

- $(E, \mathbb{F})$ ist ein uniformer Matroid mit $\mathbb{F} = \{X \subseteq E : |X| \leq k\}$:

$$r(X) = \begin{cases} |X|, \text{ wenn } X \in \mathbb{F} \\ k, \text{ wenn } X \notin \mathbb{F} \text{ d.h. } |X| > k \end{cases}$$

- $(E, \mathbb{F})$ ist ein Vektormatroid:
  Rang des Matroids $\widehat{=}$ Rang in der linearen Algebra.

- $(E, \mathbb{F})$ ist ein Kreismatroid:
  Für $X \subseteq E$ gilt: $r(X) = |V(X)| - k(X)$, wobei $V(X) = \{v : \exists e \in X$ mit v inzident zu $e\}$ und $k(X) =$ Anzahl der Zusammenhangskomponenten von $(V(X), X)$ (Beweis: Übungsaufgabe).

Theorem 5.5. *Sei E eine endliche Menge und $r : \mathbb{P}(E) \to \mathbb{Z}^+$. Dann sind die folgenden drei Aussagen äquivalent:*

(a) r ist die Rangfunktion eines Matroids $(E, \mathbb{F})$.

(b) Für alle $X, Y \subseteq E$ gilt:

(R1) $r(X) \leq |X|$;

(R2) Ist $X \subseteq Y$, *so gilt* $r(X) \leq r(Y)$;

(R3) $r(X \cup Y) + r(X \cap Y) \leq r(X) + r(Y)$.

(c) Für alle $X \subseteq E$ *und* $x, y \in E$ *gilt:*

(R1') $r(\emptyset) = 0$;

(R2') $r(X) \leq r(X \cup \{y\}) \leq r(X) + 1$;

(R3') Ist $r(X \cup \{x\}) = r(X \cup \{y\}) = r(X)$, *so gilt* $r(X \cup \{x, y\}) = r(X)$.

Beweis.

(a) $\Rightarrow$ (b):
Ist r eine Rangfunktion eines Unabhängigkeitssystems $(E, \mathbb{F})$, so sind (R1) und (R2) offensichtlich erfüllt. Ist $(E, \mathbb{F})$ ein Matroid, so gilt auch (R3), da:
Seien $X, Y \subseteq E$ und sei $A \in \mathbb{F}$ eine maximal unabhängige Teilmenge von $X \cap Y$. Da $|X| \geq |X \cap Y|$, gilt für maximal unabhängige Teilmengen A' von X, dass $|A'| \geq |A|$. Falls $|A'| = |A|$, ist nichts zu zeigen. Wenn $|A'| > |A|$ folgt mit (M3), dass ein $x \in A' \setminus A$ existiert, sodass $A \cup \{x\} \in \mathbb{F}$. D.h. A kann zu einer maximal unabhängigen Teilmenge $A \dot\cup B$ von X und analog zu einer maximal unabhängigen Teilmenge $(A \dot\cup B) \dot\cup C$ von $X \cup Y$ erweitert werden. Per Konstruktion $A, C \subseteq Y$ und $A \dot\cup C \in \mathbb{F}$, also ist $A \dot\cup C$ eine unabhängige Teilmenge von Y und es gilt $r(X) = |A \dot\cup B|$ und $r(Y) \geq |A \dot\cup C|$. Also folgt:

$$\begin{aligned} r(X) + r(Y) &\geq |A \dot\cup B| + |A \dot\cup C| \\ &= 2|A| + |B| + |C| \\ &= |A \dot\cup B \dot\cup C| + |A| \\ &= r(X \cup Y) + r(X \cap Y). \end{aligned}$$

(b) $\Rightarrow$ (c):

- Aus (R1) folgt (R1').
- Aus (R2) folgt $r(X) \leq r(X \cup \{y\})$.
  Aus (R1) und (R3) folgt dann:

$$\begin{aligned} r(X \cup \{y\}) &\leq r(X) + r(\{y\}) - r(X \cap \{y\}) \\ &\leq r(X) + r(\{y\}) \\ &\leq r(X) + 1, \end{aligned}$$

  womit (R2') bewiesen ist.
- (R3') ist trivial für $x = y$. Für $x \neq y$ folgt mit (R2), dass $r(X) \leq r(X \cup \{y\})$ und somit $2r(X) \leq r(X) + r(X \cup \{x, y\})$. Setze $A = X \cup \{x\}$ und $B = X \cup \{y\}$.

Wegen (R3) gilt: $r(X) + r(X \cup \{x, y\}) = r(A \cap B) + r(A \cup B) \leq r(A) + r(B) = r(X \cup \{x\}) + r(X \cup \{y\})$.
Wegen der Voraussetzung an (R3') gilt zusätzlich $r(X \cup \{x\}) + r(X \cup \{y\}) = 2r(X)$.
Also $r(X) = r(X \cup \{x, y\})$, womit (R3') bewiesen ist.

(c) $\Rightarrow$ (a):
Sei $r : \mathbb{P}(E) \to \mathbb{Z}^+$ mit (R1'), (R2'), (R3') gegeben. Dann definieren wir

$$\mathbb{F} = \{F \subseteq E \,:\, r(F) = |F|\}.$$

Zu zeigen ist nun, dass $(E, \mathbb{F})$ ein Matroid ist und r die dazugehörige Rangfunktion:

(i) Zu zeigen: $(E, \mathbb{F})$ ist ein Matroid.

- (M1): Aus (R1') folgt, dass $\emptyset \in \mathbb{F}$, also (M1) erfüllt.
- (M2):
  - Aus (R1') und (R2') folgt, dass $r(X) \leq |X|$ für alle $X = \{x_1, \ldots, x_n\} \subseteq E$, denn (induktiv) gilt $r(\emptyset) \leq r(\emptyset \cup \{x_1\}) \leq r(\emptyset) + 1 = 1$ und durch hinzufügen weiterer x_i und erhalten wir $r(X) \leq |X|$.
  - Sei nun $Y \in \mathbb{F}$, $y \in Y$ und $X \coloneqq Y \setminus \{y\}$. Dann folgt

    $$|X| + 1 = |Y| = r(Y) = r(X \cup \{y\}) \overset{(R2')}{\leq} r(X) + 1 \leq |X| + 1,$$

    also insbesondere $r(X) = |X|$ und somit $X \in \mathbb{F}$. Wiederholte Anwendung dieses Schrittes auf X und den resultierenden Teilmengen zeigt, dass *alle* Teilmengen von Y in $\mathbb{F}$ enthalten sind, also ist (M2) erfüllt.
- (M3): Seien $X, Y \in \mathbb{F}$ mit $|X| = |Y| + 1$. Sei $X \setminus Y = \{x_1, \ldots, x_k\}$. Angenommen (M3') gilt nicht, d.h. $Y \cup \{x_i\} \notin \mathbb{F} \forall\, 1 \leq i \leq k$. Da $Y \cup \{x_i\} \notin \mathbb{F}$, folgt $r(Y \cup \{x_i\}) \neq |Y| + 1$.
  - Aus (R2') folgt dann:

    $$|Y| = r(Y) \leq r(Y \cup \{x_i\}) \leq r(Y) + 1 = |Y| + 1,$$

    d.h.

    $$r(Y \cup \{x_i\}) = |Y| = r(Y) \,\forall\, 1 \leq i \leq k.$$

  - Aus (R3') folgt schließlich:

    $$r(Y \cup \{x_1, x_i\}) = r(Y) \,\forall\, 2 \leq i \leq k.$$

    Diesen Schritt können wir mehrfach wiederholen und erhalten

    $$|Y| = r(Y) = r(Y \cup X \setminus Y) = r(Y \cup X) \geq r(X) = |X|,$$

also insbesondere $|Y| \geq |X|$.
↯Widerspruch zu $|X| = |Y| + 1$.

Also ist $(E, \mathbb{F})$ ein Matroid.

(ii) Zu zeigen: r ist Rangfunktion des Matroids $(E, \mathbb{F})$, d.h. $\forall X \subseteq E$ gilt: $r(X) = \max\{|Y| : Y \subseteq X, \underbrace{Y \in \mathbb{F}}_{r(Y)=|Y|}\}$.
Sei $X \subseteq E$ und sei $Y \subseteq X$ eine unabhängige Teilmenge von X maximaler Kardinalität mit $r(Y) = |Y|$.
Für alle $x \in X \setminus Y$ gilt:

$$r(Y \cup \{x\}) < |Y| + 1.$$

Diesen Schritt können wir mehrfach wiederholen und erhalten

$$|Y| = r(Y) = r(Y \cup (X \setminus Y)) = r(X).$$

□

Kreisaxiome

Erinnerung: Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem. Dann ist ein *Kreis* C von $(E, \mathbb{F})$ eine (inklusions-)minimale abhängige Menge. Weiterhin sind der *Rang* und der *untere Rang* von $X \subseteq E$ definiert als

$$r(X) \coloneqq \max\{|Y| : Y \subseteq X, Y \in \mathbb{F}\},$$

bzw.

$$\rho(X) \coloneqq \min\{|Y| : Y \subseteq X, Y \in \mathbb{F} \text{ und } Y \cup \{x\} \notin \mathbb{F} \text{ für alle } x \in X \setminus Y\}.$$

In einem späteren Beweis werden wir noch den sogenannten *Rangquotient* verwenden, der wie folgt definiert ist:

Definition (Rangquotient)**.** Der *Rangquotient* eines Unabhängigkeitssystems $(E, \mathbb{F})$ wird definiert als

$$q(E, \mathbb{F}) \coloneqq \min_{F \subseteq E} \frac{\rho(F)}{r(F)}.$$

Dabei gilt $q(E, \mathbb{F}) \leq 1$ (da $\rho(F) \leq r(F)$).
Zudem gilt: $q(E, \mathbb{F}) = 1 \Leftrightarrow$ (M3") (Übungsaufgabe).

Proposition 5.6. *Sei E eine endliche Menge und $\mathbb{C} \subseteq \mathbb{P}(E)$. $\mathbb{C}$ ist genau dann die Menge der Kreise des Unabhängigkeitssystems $(E, \mathbb{F})$, wenn folgende Bedingungen erfüllt sind:*

(C1) $\emptyset \notin \mathbb{C}$;

(C2) Für alle $C_1, C_2 \in \mathbb{C}$ mit $C_1 \subseteq C_2$ folgt $C_1 = C_2$.

Beweis.

„⇒“ Sei $(E, \mathbb{F})$ ein Matroid. Da wegen (M1) $\emptyset \in \mathbb{F}$ (d.h. $\emptyset$ ist eine unabhängige Menge), folgt (C1). Da Kreise in $(E, \mathbb{F})$ minimal abhängige Mengen sind, folgt auch (C2).

„⇐“ Setze $\mathbb{F} = \{F \subseteq E : \nexists C \in \mathbb{C} \text{ mit } C \subseteq F\}$.

- Wenn $\mathbb{C}$ die Bedingung (C1) erfüllt, so gilt $\emptyset \in \mathbb{F}$ und nach Konstruktion von $\mathbb{F}$ ist $(E, \mathbb{F})$ ein Unabhängigkeitssystem.
- Wenn $\mathbb{C}$ zusätzlich (C2) erfüllt, so ist $\mathbb{C}$ die Menge der Kreise dieses Unabhängigkeitssystems $(E, \mathbb{F})$.

□

Theorem 5.7. *Sei E eine endliche Menge und $\mathbb{C} \subseteq \mathbb{P}(E)$, sodass $\mathbb{C}$ die Menge der Kreise eines Unabhängigkeitssystems $(E, \mathbb{F})$ ist. Dann sind die folgenden Aussagen äquivalent:*

(a) *$(E, \mathbb{F})$ ist ein Matroid.*

(b) *Für alle $X \in \mathbb{F}$ und $e \in E$ gilt: $X \cup \{e\}$ enthält höchstens einen Kreis.*

(C3) *Für alle $C_1, C_2 \in \mathbb{C}$ mit $C_1 \neq C_2$ und für alle $e \in C_1 \cap C_2$ gibt es ein $C_3 \in \mathbb{C}$ mit $C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$.*

(C3') *Für alle $C_1, C_2 \in \mathbb{C}$, $e \in C_1 \cap C_2$ und $f \in C_1 \setminus C_2$ gibt es ein $C_3 \in \mathbb{C}$, sodass $f \in C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$.*

Beweis.

(a) ⇒ (C3'):
Sei $\mathbb{C}$ die Menge der Kreise eines Matroids. Ferner seien $C_1, C_2 \in \mathbb{C}$, $e \in C_1 \cap C_2$ und $f \in C_1 \setminus C_2$. Es ist $r(C_i) = |C_i| - 1$ (C_i sind minimal abhängig, d.h. $C_i \setminus \{x\} \in \mathbb{F}$ für alle $x \in C_i$).
Im Folgenden wenden wir (R3) zweimal an und erhalten:

$$\begin{aligned} & |C_1| - 1 + r((C_1 \cup C_2) \setminus \{e, f\}) + |C_2| - 1 \\ =\, & r(C_1) + r((C_1 \cup C_2) \setminus \{e, f\}) + r(C_2) \\ \overset{(R3)}{\geq}\, & r(C_1) + r((C_1 \cup C_2) \setminus \{f\}) + r(C_2 \setminus \{e\}) \\ \overset{(R3)}{\geq}\, & r(C_1 \setminus \{f\}) + r(C_1 \cup C_2) + r(C_2 \setminus \{e\}) \\ =\, & |C_1| - 1 + r(C_1 \cup C_2) + |C_2| - 1, \end{aligned}$$

also insbesondere $r((C_1 \cup C_2) \setminus \{e, f\}) \geq r(C_1 \cup C_2)$. Wegen (R2) gilt $r((C_1 \cup C_2) \setminus \{e, f\}) \leq r(C_1 \cup C_2)$, also folgt insgesamt

$$r((C_1 \cup C_2) \setminus \{e, f\}) = r(C_1 \cup C_2).$$

Wähle nun eine maximal unabhängige Teilmenge $B \subseteq (C_1 \cup C_2) \setminus \{e, f\}$.
Wegen (M3") gilt: $|B| = r((C_1 \cup C_2) \setminus \{e, f\})$.

Außerdem ist B offensichtlich auch eine Teilmenge von $(C_1 \cup C_2)$.
Insbesondere muss $B \cup \{f\}$ abhängig sein (enthält also einen Kreis, welcher f beinhaltet), denn wäre dies nicht der Fall, so wäre $B \cup \{f\} \subseteq (C_1 \cup C_2)$ unabhängig und somit $r(C_1 \cup C_2) \geq |B|+1$, welches im Widerspruch steht zu $r((C_1 \cup C_2) \setminus \{e, f\}) = r(C_1 \cup C_2) = |B|$.
Somit enthält $B \cup \{f\}$ einen Kreis C_3 mit $f \in C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$.

(C3') $\Rightarrow$ (C3): klar $\surd$

(C3) $\Rightarrow$ (b):
Sei $X \in \mathbb{F}$ und $X \cup \{e\}$ enthalte zwei Kreise C_1, C_2. Dann folgt aus (C3): Es gibt ein $C_3 \in \mathbb{C}$ mit $C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$. Damit gilt $(C_1 \cup C_2) \setminus \{e\} \notin \mathbb{F}$. Aber $(C_1 \cup C_2) \setminus \{e\}$ ist Teilmenge von $X \in \mathbb{F}$.
↯Widerspruch.

(b) $\Rightarrow$ (a):
Es gelte für alle $X \in \mathbb{F}$ und $e \in E$, dass $X \cup \{e\}$ höchstens einen Kreis enthält.
Wir zeigen jetzt, dass unter dieser Voraussetzung für den Rangquotient gilt: $q(E, \mathbb{F}) = 1$ (welches äquivalent zu (M3") ist)
Sei $F \subseteq E$ und J, K zwei maximal unabhängige Teilmengen von F. Wir zeigen zuerst, dass

$$\frac{|J|}{|K|} \geq 1.$$

Da F, J und K beliebig sind, folgt $q(E, \mathbb{F}) \geq 1$. Nach Definition ist $q(E, \mathbb{F}) \leq 1$, also folgt $q(E, \mathbb{F}) = 1$, und somit ist das Unabhängigkeitssystem $(E, \mathbb{F})$ ein Matroid.
Betrachte $J \setminus K$:
Falls $J \setminus K = \emptyset$ gilt entweder

(i) $J = K$, woraus $\frac{|J|}{|K|} = 1$ folgt, oder

(ii) $J \subseteq K$, was ein Widerspruch zu maximalen Unabhängigkeit von J ist.

Wir können also im Folgenden annehmen, dass $J \setminus K \neq \emptyset$. Sei $J \setminus K = \{e_1, \ldots, e_t\}$.
Wir konstruieren eine Folge $K \coloneqq K_0, K_1, \ldots, K_t \in \mathbb{F}$, sodass gilt:

- $J \cap K \subseteq K_i \subseteq J \cup K$ für $0 \leq i \leq t$,
- $K_i \cap \{e_1, \ldots, e_t\} = \{e_1, \ldots, e_i\}$ für $1 \leq i \leq t$ und
- $|K_{i-1} \setminus K_i| \leq 1$ für $1 \leq i \leq t$.

Induktiv zeigen wir dass jedes $K_j \in \mathbb{F}$:
Für $K_0 = K$ gilt die Aussage. Annahme sie gelte für alle $j \leq i$.
Also $K_i \in \mathbb{F}$, woraus folgt, dass $K_i \cup \{e_{i+1}\}$ höchstens einen Kreis $C \notin \mathbb{F}$ enthält.
Für einen solchen Kreis C gilt, dass $C \cap K_i \setminus J \neq \emptyset$, d.h. C beinhaltet mindestens ein Element aus K_i welches nicht in J liegt, andernfalls würde C nur Elemente aus J beinhalten und somit $C \subseteq J$, Widerspruch zu $J \in \mathbb{F}$. Sei $z \in C \cap K_i \setminus J$. Da $z \notin J$ und $e_{i+1} \in J$ folgt $z \neq e_{i+1}$. Dann ist $(K_i \setminus \{z\}) \cup \{e_{i+1}\}$ kein Kreis, also Element von $\mathbb{F}$.

Setze $K_{i+1} \coloneqq (K_i \setminus \{z\}) \cup \{e_{i+1}\} \in \mathbb{F}$
Per Konstruktion ist $J \subseteq K_t \in \mathbb{F}$.
Da J eine maximal unabhängige Teilmenge von F ist, folgt $J = K_t$. Damit gilt:

$$|K \setminus J| = \sum_{i=1}^{t} |K_{i-1} \setminus K_i| \leq t = |J \setminus K|.$$

Daraus folgt

$$|K| \leq |J|, \text{ also } \frac{|J|}{|K|} \geq 1.$$

□

Korollar. $\mathbb{C}$ *ist die Menge der Kreise eines Matroids*

$\Leftrightarrow$ *(C1), (C2) und (C3) sind erfüllt.*

$\Leftrightarrow$ *(C1), (C2) und (C3') sind erfüllt.*

5.2.3 Der GREEDY-Algorithmus

Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem und $c : E \to \mathbb{R}^+$ eine Gewichtsfunktion. Wir betrachten im Folgenden das Maximierungsproblem für dieses Unabhängigkeitssystem, wobei wir von einer positiven Gewichtsfunktion ausgehen können, da eine optimale Lösung nie Elemente mit negativem Gewicht enthalten wird. (Angenommen E^* ist eine optimale Lösung, die Elemente mit negativem Gewicht enthält. Dann gilt für $F = \{x \in E^* : \text{alle } x \in F \text{ haben positives Gewicht }\}$, dass $c(F) > c(E^*)$, d.h. E^* war nicht optimal ↯.)

Das System $(E, \mathbb{F})$ ist durch ein „Unabhängigkeits-Orakel“ gegeben (oft ist $\mathbb{F}$ z.B. nicht vollständig bekannt, aber man kann testen, ob für ein gegebenes $F \subseteq E$ gilt, dass $F \in \mathbb{F}$ oder nicht). Damit können wir nun den GREEDY-Algorithmus formulieren:

Algorithm 7 GREEDY(Unabhängigkeitssystem $(E, \mathbb{F})$, Gewichtsfunktion $c : E \to \mathbb{R}^+$)

```
1: Sortiere E = {e_1, ..., e_n}, sodass c(e_1) ≥ c(e_2) ≥ ... ≥ c(e_n).
2: F ← ∅
3: for i = 1, ..., n do
4:     if F ∪ {e_i} ∈ 𝔽 then
5:         F ← F ∪ {e_i} ∈ 𝔽
6:     end if
7: end for
8: return F ∈ 𝔽
```

Anmerkungen.

- Wenn das „Unabhängigkeits-Orakel“ Laufzeit $f(n)$ hat und E sortiert ist, so hat der GREEDY-Algorithmus Laufzeit $\mathcal{O}(n \cdot f(n))$.

- Die Lösung von GREEDY muss nicht eindeutig sein (z.B. ist schon die Sortierung von Elementen mit gleichem Gewicht nicht eindeutig).

Theorem 5.8. *Sei $(E, \mathbb{F})$ ein Unabhängigkeitssystem und $c : E \to \mathbb{R}^+$ eine Gewichtsfunktion. Sei weiter $G(E, \mathbb{F}, c)$ die GREEDY-Lösung des Maximierungsproblems und $OPT(E, \mathbb{F}, c)$ die entsprechende optimale Lösung. Dann gilt:*

$$q(E, \mathbb{F}) = \min_{F \subseteq E} \frac{\rho(F)}{r(F)} \leq \frac{G(E, \mathbb{F}, c)}{OPT(E, \mathbb{F}, c)} \leq 1,$$

für alle $c : E \to \mathbb{R}^+$.

Beweis. Sei $E = \{e_1, e_2, \ldots, e_n\}$, $c : E \to \mathbb{R}^+$ und $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$. Sei $G_n \subseteq E$ die Lösung von GREEDY und $O_n \subseteq E$ die optimale Lösung, wobei $G_0 \coloneqq \emptyset$ und $O_0 \coloneqq \emptyset$. Weiter definieren wir $E_j \coloneqq \{e_1, \ldots, e_j\}$, $G_j \coloneqq G_n \cap E_j$ und $O_j \coloneqq O_n \cap E_j$ und setzen außerdem $d_n \coloneqq c(e_n)$ und $d_j \coloneqq c(e_j) - c(e_{j+1})$ für $j = 1, \ldots, n-1$.

Beobachtung: $G_{j-1} \subseteq G_j$ und G_j enthält höchstens ein Element mehr als G_{j-1}, also $|G_j| - |G_{j-1}| \in \{0, 1\}$ und $|G_j| - |G_{j-1}| = 1 \Leftrightarrow e_j \in G_j$. Also

$$\begin{aligned}
c(G_n) &= \sum_{e_j \in G_n} c(e_j) \\
&= \sum_{j=1}^{n} (|G_j| - |G_{j-1}|)\, c(e_j) \\
&= \sum_{j=1}^{n} |G_j|\, c(e_j) - \sum_{j=1}^{n} |G_{j-1}|\, c(e_j) \\
&= \sum_{j=1}^{n} |G_j|\, c(e_j) - \sum_{j=2}^{n} |G_{j-1}|\, c(e_j) \quad (\text{da } |G_0| = 0) \\
&= \sum_{j=1}^{n} |G_j|\, c(e_j) - \sum_{j=1}^{n-1} |G_j|\, c(e_{j+1}) \\
&= G_n\, c(e_n) + \sum_{j=1}^{n-1} |G_j|\, (c(e_j) - c(e_{j+1})) \\
&= \sum_{j=1}^{n} |G_j|\, d_j
\end{aligned}$$

Weiterhin ist G_j eine maximal unabhängige Teilmenge von E_j, d.h.

$$|G_j| \geq \rho(E_j).$$

Da $O_j \in \mathbb{F}$ und $O_j \subseteq E_j$ gilt

$$|O_j| \leq r(E_j).$$

Damit folgt

$$\begin{aligned} c(G_n) &= \sum_{j=1}^{n} |G_j|\, d_j \\ &\geq \sum_{j=1}^{n} \rho(E_j)\, d_j \\ &\geq \sum_{j=1}^{n} r(E_j)\, q(E,\mathbb{F})\, d_j \\ &\geq \sum_{j=1}^{n} |O_j|\, q(E,\mathbb{F})\, d_j \\ &= q(E,\mathbb{F}) \sum_{j=1}^{n} (|O_j| - |O_{j-1}|) c(e_j) \\ &= q(E,\mathbb{F})\, c(O_n) \end{aligned}$$

□

Theorem 5.9 (Edmonds-Rado-Satz)**.** *Ein Unabhängigkeitssystem $(E,\mathbb{F})$ ist genau dann ein Matroid, wenn der GREEDY-Algorithmus eine optimale Lösung für das Maximierungsproblem $(E,\mathbb{F},c)$ für alle $c : E \to \mathbb{R}^+$ findet.*

Beweis. $(E,\mathbb{F})$ ist kein Matroid ((M3") verletzt) $\Leftrightarrow$ $q(E,\mathbb{F}) < 1$ $\Leftrightarrow$ GREEDY findet keine optimale Lösung. □

Alternativer Beweis.

„$\Leftarrow$" Angenommen $(E,\mathbb{F})$ ist kein Matroid.
Dann gilt insbesondere (M3') nicht, d.h. es existieren $I_p, I_{p+1} \in \mathbb{F}$ mit $|I_p| = p$, $|I_{p+1}| = p+1$ und $\nexists\, e \in I_{p+1} \setminus I_p$ mit $I_p \cup \{e\} \in \mathbb{F}$. $(*)$

$$\text{Sei } c : E \to \mathbb{R}^+ \text{ mit } c(e) = \begin{cases} p+2, & e \in I_p \\ p+1, & e \in I_p + 1 \setminus I_p \\ 0, & \text{sonst} \end{cases} .$$

Dann gilt:

$$\begin{aligned} c(I_{p+1}) &= \sum_{e \in I_{p+1} \setminus I_p} p+1 + \sum_{e \in I_p \cap I_{p+1}} p+2 \\ &\geq |I_{p+1}|\,(p+1) \\ &= (p+1)\,(p+1) \\ &= p^2 + 2p + 1 \\ &> p^2 + 2p \\ &= p(p+2) \\ &= c(I_p) \end{aligned}$$

GREEDY wählt aber zuerst alle Elemente aus I_p. Da (M3') nicht gilt, kann zu I_p kein Element aus $I_{p+1} \setminus I_p$ hinzugefügt werden (siehe $(*)$).
$\Rightarrow$ GREEDY liefert eine Lösung mit $c(I_p) < c(I_{p+1})$, also insbesondere keine optimale Lösung.

„$\Rightarrow$“ Sei $(E, \mathbb{F})$ ein Matroid und sei die Lösung von GREEDY $I = \{e_1, \ldots, e_i\} \in \mathbb{F}$ mit $c(e_1) \geq \ldots \geq c(e_i)$. Sei $J = \{f_1, \ldots, f_j\} \in \mathbb{F}$ mit $c(f_1) \geq \ldots \geq c(f_j)$ eine beliebige andere zulässige Lösung, sodass J maximal unabhängig. Da maximal unabhängige Teilmengen eines Matroids gleiche Kardinalität haben, folgt $|I| = |J|$ (d.h. $i = j$).
Wir zeigen, dass für alle $m \in \{1, \ldots, i\}$ gilt: $c(e_m) \geq c(f_m)$ (dies impliziert $c(I) \geq c(J)$, d.h. die Lösung von GREEDY ist optimal).
Angenommen, dies wäre nicht der Fall. Dann sei k der kleinste Index mit $c(e_k) < c(f_k)$. Nun setzen wir $I' \coloneqq \{e_1, \ldots, e_{k-1}\}$ und $J' \coloneqq \{f_1, \ldots, f_{k-1}, f_k\}$.
Da offensichtlich $|I'| < |J'|$, folgt mit (M3): $\exists\, f_t \in J' \setminus I'$, sodass $I' \cup \{f_t\} \in \mathbb{F}$. Für alle Elemente $f_t \in J' \setminus I'$ gilt aber, dass $c(f_t) \geq c(f_k) > c(e_k)$, d.h. in der sortierten Menge E stehen f_t und f_k vor e_k, was bedeutet, dass entgegen unserer Annahme $f_t \in I'$
↯Widerspruch

□

Anmerkung. Der Edmonds-Rado-Satz gilt auch für das Minimierungsproblem von Unabhängigkeitssystemen, d.h. das Minimierungsproblem $(E, \mathbb{F}, c)$ mit $c : E \to \mathbb{R}$, kann ebenfalls mit GREEDY optimal gelöst werden gdw. $(E, \mathbb{F})$ ein Matroid ist. Dazu betrachten wir das äquivalente Maximierungsproblem $(E, \mathbb{F}, c')$, $c' : E \to \mathbb{R}^+$ mit $c'(e) = M + c(e)$ für alle $e \in E$ und $M \coloneqq \max\{|c(e)| : e \in E\}$, welches mittels GREEDY gelöst wird.

Wir betrachten nun die Charakterisierung von optimalen Teilmengen der Kardinalität k.

Theorem 5.10. *Sei $(E, \mathbb{F})$ ein Matroid, $c : E \to \mathbb{R}$, $k \in \mathbb{N}$ und $X \in \mathbb{F}$ mit $|X| = k$. Dann gilt $c(X) = \max\{c(Y) : Y \in \mathbb{F}, |Y| = k\}$ genau dann, wenn die folgenden Bedingungen erfüllt sind:*

(a) Für alle $y \in E \setminus X$ mit $X \cup \{y\} \notin \mathbb{F}$ und alle $x \in C(X, y)$ (eindeutig bestimmter Kreis mit Elementen aus X, der y enthält) gilt: $c(x) \geq c(y)$;

(b) Für alle $y \in E \setminus X$ mit $X \cup \{y\} \in \mathbb{F}$ und alle $x \in X$ gilt $c(x) \geq c(y)$.

Beweis.

„⇒“ Angenommen (a) ist verletzt, d.h. $c(y) > c(x)$. Dann gilt für $X' := (X \cup \{y\}) \setminus \{x\} \in \mathbb{F}$, dass $c(X') > c(X)$.
↯Widerspruch zur Optimalität von X.
Analog Widerspruch, falls (b) verletzt.

„⇐“ Seien (a) und (b) erfüllt.
Weiter sei $\mathbb{F}' := \{F \in \mathbb{F} : |F| \leq k\} \subseteq \mathbb{F}$ und $c'(e) := c(e) + M$ für alle $e \in E$, wobei $M = \max\{|c(e)| : e \in E\}$, d.h. $c' : E \to \mathbb{R}^+$.
Sei $E = \{e_1, \ldots, e_n\}$ sortiert, sodass $c'(e_1) \geq \ldots \geq c'(e_n)$ und dass für beliebiges i gilt:

$$c'(e_i) = c'(e_{i+1}) \text{ und } e_{i+1} \in X \Rightarrow e_i \in X$$

(d.h. aus Elementen mit gleichem Gewicht, kommen diejenigen aus X zuerst in der Sortierung vor).
Sei X' die GREEDY-Lösung für $(E, \mathbb{F}', c')$ (mit obiger Sortierung). OBdA können wir annehmen, dass $|X'| = |X| = k$, andernfalls könnte man wegen (M3) X' um Elemente von X erweitern (welche positive Gewichte c' haben) und dies ist ein Widerspruch zur maximalen Unabhängigkeit von X' in der GREEDY Konstruktion. Es gilt:

$$c(X') = \sum_{e \in X'} c(e) = \sum_{e \in X'} (c'(e) - M) = \Big(\sum_{e \in X'} c'(e) \Big) - kM = c'(X') - kM.$$

Nach dem Edmonds-Rado-Satz (5.9) gilt , dass X' eine optimale Lösung für $(E, \mathbb{F}', c')$ ist, also

$$\begin{aligned} c(X') + kM &= c'(X') \\ &= \max\{c'(Y) : Y \in \mathbb{F}'\} \\ &= \max\{c(Y) : Y \in \mathbb{F}, |Y| = k\} + kM. \end{aligned}$$

Jetzt ist noch zu zeigen, dass $X = X'$. Wir wissen, dass $|X| = k = |X'|$.
Wenn $X \neq X'$, dann existiert ein $e_i \in X' \setminus X$ mit kleinstem Index i, d.h. insbesondere $X \cap \{e_1, \ldots, e_{i-1}\} = X' \cap \{e_1, \ldots, e_{i-1}\}$. Nun unterscheiden wir zwei Fälle:

(i) $X \cup \{e_i\} \notin \mathbb{F}$

$\Rightarrow$ Da $e_i \notin X$ ist $c(x) \geq c(e_i)$ für alle $x \in C(X, e_i)$ (wegen (a)).

$\Rightarrow$ Alle Elemente e aus $C(X, e_i)$ mit $e \neq x_i$ kommen vor e_i.

$\Rightarrow$ $C(X, e_i) \subseteq (X \cap \{e_1, \ldots, e_{i-1}\}) \cup \{e_i\} = X' \cap \{e_1, \ldots, e_{i-1}, e_i\}$.

$\Rightarrow$ $C(X, e_i) \subseteq X'$, also $X' \cup \{e_i\} \notin \mathbb{F}$ $\lightning$Widerspruch.

(ii) $X \cup \{e_i\} \in \mathbb{F}$

$\Rightarrow$ Für *alle* $x \in X$ gilt: $c(x) \geq c(e_i)$ (wegen (b)).

$\Rightarrow$ $X \subseteq \{e_1, \ldots, e_{i-1}\} \subsetneq X'$

$\Rightarrow$ $|X| < |X'|$ $\lightning$Widerspruch.

□

Das Münzwechselproblem Person Y geht in den Supermarkt, bezahlt und bekommt Rückgeld von X Eurocent in Form von Münzen. Dabei soll das Rückgeld aus so wenigen Münzen wie möglich bestehen → Wie muss der/die Kassierer(in) vorgehen?

Die Antwort ist eine GREEDY-Strategie: Gib solange die Münzen mit dem größten Wert zurück wie möglich.

Anmerkung. Hierbei handelt es sich um eine "Form" des Rucksack-Problems:
Gegeben: Münzzerlegung $M = (m_1, \ldots, m_k)$ mit $m_k > m_{k-1} > \ldots > m_1$, Rückgeld X
Ziel: Finde Anzahlen $x_i \geq 0$ für alle Münzen m_i, $1 \leq i \leq k$, sodass

$$\sum_{i=1}^{k} x_i m_i = X \qquad \text{und} \qquad \sum_{i=1}^{k} x_i \xrightarrow{!} \min$$

Algorithm 8 GREEDY_ Münzen (Rückgeld X, Münzzerlegung M)

```
1: N ← 0                                   ▷ Anzahl der Münzen
2: U ← X
3: for i = k, ..., 1 do
4:     x_i ← ⌊U / m_i⌋
5:     U ← U − x_i m_i
6:     N ← N + x_i
7: end for
8: return (x_1, ..., x_k), N
```

Frage: Funktioniert die GREEDY-Strategie für alle Münzzerlegungen $M = (m_1, \ldots, m_k)$?

- Beobachtung: Es muss gelten $m_1 = 1$, denn sonst kann z.B. ein Rückgeld von $X = 1$ nicht ausgegeben werden.
- GREEDY funktioniert nicht immer, z.B. $M = (1, 3, 4)$ und $X = 6$. Dann wählt GREEDY die Münzen $\{4, 1, 1\}$, optimal wäre jedoch $\{3, 3\}$.
- GREEDY funktioniert für:

  €: $M_€ = (200, 100, 50, 20, 10, 5, 2, 1)$

  \$: $M_\$ = (100, 50, 25, 10, 5, 1)$

Im Folgenden wollen wir das Münzwechselsystem mit der Matroid-Theorie in Verbindung bringen.
Dazu betrachten wir folgendes Unabhängigkeitssystem $(E, \mathbb{F})$ mit

- E ist eine Multimenge (M, l), wobei $l : M \to \mathbb{N}^+$, $m \mapsto l(m) =$ Häufigkeit von m.
  Für das Eurosystem ist dies $(E_{€} = \{1, \ldots, 1, 2, \ldots, 2, 5, \ldots, 5, \ldots\ldots, 200, \ldots, 200\}$. Dabei sei E „genügend groß“, d.h. alle Münzsorten seien in ausreichender Zahl vorhanden. Formal kann man die Multimenge E auch als einfache Menge auffassen, indem alle Elemente durchnummeriert werden, d.h. $E = \{1_1, 1_2, \ldots, 1_{l(1)}, \ldots\ldots, 200_1, \ldots, 200_{l(200)}\}$.
- $\mathbb{F} = \{F \subseteq E : \nexists F' \subseteq E \text{ mit } \sum_{f \in F} f = \sum_{f' \in F'} f' \text{ und } |F'| < |F|\}$, d.h. $\mathbb{F}$ enthält alle Teil(multi)mengen von E, für die die Summe der Münzwerte nicht durch weniger Münzen erreicht werden kann.

Beispiel. Sei $M = (1, 3, 4)$ und $X = 6$. Dann ist $E = \{1, \ldots, 1, 3, \ldots, 3, 4, \ldots, 4\}$ und $\mathbb{F} = \{\{1\}, \{1, 1\}, \{3\}, \{4\}, \{1, 4\}, \{3, 3\}, \{3, 4\}, \ldots, \{3, 4, 4\}, \ldots\}$. Das Unabhängigkeitssystem $(E, \mathbb{F})$ ist kein Matroid, da die Austauscheigenschaft (M3) nicht gilt:
Betrachte z.B. $\{1, 1\}$ und $\{3, 4, 4\}$. Dann

$\{1, 1, 3\} \notin \mathbb{F}$ (da $\{1, 4\} \in \mathbb{F}$)

$\{1, 1, 4\} \notin \mathbb{F}$ (da $\{3, 3\} \in \mathbb{F}$)
↯

Für Euro und Dollar ist $(E, \mathbb{F})$ jedoch ein Matroid.

Lemma 5.11. *Sei $M_{€}$ die Euromünzzerlegung und X ein Rückgabewert (in Cent). Dann gilt für die optimale Rückgabemenge M^*, dass das größte $m_i \in M_{€}$ mit $m_i \leq X$ in M^* enthalten ist.*

Beweis. „Quasi Enumeration“:

$X = 1 \to 1 \in M^* = \{1\}$

$X = 2 \to 2 \in M^* = \{2\}$

$X = 3 \to 2 \in M^* = \{2, 1\}$

$\vdots$

$X = 9 \to 5 \in M^* = \{5, 2, 2\}$

Jetzt betrachten wir $10 \leq X < 19$ und zeigen, dass die 10-Centmünze in M^* enthalten ist. Angenommen, dies wäre nicht der Fall. Dann muss X in Form von 1-, 2- und 5-Centmünzen ausgezahlt werden.
Sei also $X = \underbrace{k \cdot 5}_{k \times \text{ 5-Cent}} + \underbrace{R}_{\text{Rest in 1-, 2-Cent}}$, also $R = X - 5k$.

Wir wissen, dass für alle Rückgabewerte Y mit $5 \leq Y < 10$ die 5-Centmünze verwendet wird, also muss gelten $R = X - 5k < 5$ (sonst enthält R auch die 5-Centmünze). Daraus folgt:

$$10 \leq X < 5(k+1) \Rightarrow k \geq 2,$$

d.h. es existieren mindestens zwei 5-Centmünzen, die durch eine 10-Centmünze ersetzt werden können.
↯Widerspruch.
Analog erfolgt der Beweis für die 20-, 50-, 100- und 200-Centmünzen sowie für das Dollar-System. □

Folgendes Korollar (Item 1) kann man sich leicht aus dem letzten Lemma herleiten, Item 2 folgt dann aus der Tatsache, dass GREEDY_Münzen die maximal unabhängigen Teilmengen von $(E_€, \mathbb{F})$ bzw. $(E_\$, \mathbb{F})$ liefert (Übungsaufgabe).

Korollar 5.12. *(1) GREEDY_Münzen liefert eine optimale Anzahl von Münzen für jeden Rückgabewert X, wenn das Münzsystem $M_\$$ oder $M_€$.*
(2) $(E_€, \mathbb{F})$ bzw. $(E_\$, \mathbb{F})$ ist ein Matroid.

Die Matroideigenschaft kann man auch direkt zeigen.

Lemma 5.13. *$(E_€, \mathbb{F})$ (wie oben definiert) ist ein Matroid.*

Beweiskizze.

(M1) $\emptyset \in \mathbb{F}$ √ (entspricht Rückgeld $X = 0$ Cent)

(M2) Zu zeigen: Für alle $F \in \mathbb{F}$ und alle $F' \subseteq F$ gilt: $F' \in \mathbb{F}$.
Angenommen es existiert ein $F \in \mathbb{F}$, sodass $\exists\, F' \subseteq F$ mit $F' \notin \mathbb{F}$.
Sei $\sum\limits_{f' \in F'} f' = c$. Da nach Annahme $F' \notin \mathbb{F}$, folgt

$$\exists\, F'' \in \mathbb{F}, \text{ sodass } \sum_{f' \in F'} f' = \sum_{f'' \in F''} f'' = c \text{ und } |F''| < |F'|.$$

Das bedeutet aber $|F| > |F \setminus F' \cup F''|$.
↯Widerspruch.

Somit ist $(E_€, \mathbb{F})$ ein Unabhängigkeitssystem. Die Austauscheigenschaft $(M3)$ ist aufwendiger zu zeigen, man kann stattdessen aber äquivalent zeigen, dass $r(X) = \max\{|Y| : Y \subseteq X, Y \in \mathbb{F}\}$ die Eigenschaften (R1'), (R2') und (R3') erfüllt und somit die Rangfunktion eines Matroids ist (Übungsaufgabe). □

Anmerkungen.

- Charakterisierungen von Münzsystemen, für die GREEDY funktioniert, sind bisher nur für 3, 4, und 5 Münztypen bekannt (d.h. für $(1, m_1, m_2)$, $(1, m_1, m_2)$ und $(1, m_1, m_2, m_3, m_4)$).

- Zusätzlich ist bekannt, aus welchem Rückgeld X man Gegenbeispiele konstruieren kann, wenn das Münzsystem nicht GREEDY-optimal. Es lässt sich in diesem Fall immer ein Gegenbeispiel für X mit $m_3 < X < m_{k-1} + m_k$ konstruieren.

5.2.4 Der Schnitt von Matroiden

Definition (Partitionsmatroid)**.** Sei E eine endliche Menge und $\Pi = (E_1, \ldots, E_k)$ eine Partition von E. Dann heißt $(E, \mathbb{F})$ mit $\mathbb{F} = \{F \subseteq E : |F \cap E_i| \leq 1 \forall E_i \in \Pi\}$ (also, keine zwei Elemente von F sind in der gleichen Menge von Π,) ein *Partitionsmatroid*.

Anmerkung. $(E, \mathbb{F})$ aus obiger Definition ist tatsächlich ein Matroid: (M1) ist klar und (M2) folgt aus der Tatsache dass für jedes $A \subseteq B \in \mathbb{F}$ gilt $A \cap E_i \subseteq B \cap E_i$. Für (M3) nehme man an, dass $A, B \in \mathbb{F}$ mit $|A| > |B|$. Somit existiert ein j sd. $|A \cap E_j| = 1$ und $|B \cap E_j| = 0$, andernfalls wäre $|B| \geq |A|$. Sei $a \in A \cap E_j$, dann gilt $B \cup \{a\} \in \mathbb{F}$. Analog kann man zeigen, dass $r(A) = |J(A)|$ mit $J(A) = \{j : E_j \cap A \neq \emptyset\}$ die Rangfunktion eines Matroids ist.

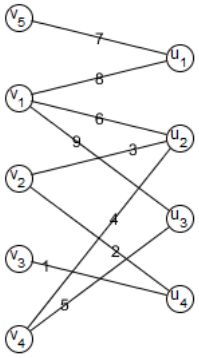

Abbildung 5.2: Beispiel Partitionsmatroide

Beispiel. Betrachte den in Abbildung 5.2 dargestellten Graphen $G = (V \dot{\cup} U, E)$ und betrachte die Partition Π_v als auch die Partition Π_u von E:

$$\Pi_v = \{E_{v_i} \subseteq E : v_i \in V, E_{v_i} = \{e : v_i \in e\}\}$$
$$\Pi_u = \{E_{u_i} \subseteq E : u_i \in U, E_{u_i} = \{e : u_i \in e\}\}.$$

Für das Beispiel ist also $\Pi_v = \{\{6, 8, 9\}, \{2, 3\}, \{1\}, \{4, 5\}, \{7\}\}$ und $\Pi_u = \{\{7, 9\}, \{3, 4, 6\}, \{5, 8\}, \{1, 2\}\}$. Definiere nun den Partitionsmatroid $(E, \mathbb{F}_v)$ bzgl. Π_v sowie den Partitionsmatroid $(E, \mathbb{F}_u)$ bzgl. Π_u.

Definition (Schnitt von Unabhängigkeitssystemen/Matroiden)**.** Seien $(E, \mathbb{F}_1), \ldots, (E, \mathbb{F}_k)$ Unabhängigkeitssysteme/Matroide. Dann ist ihr *Schnitt* definiert als $(E, \bigcap_{i=1}^{k} \mathbb{F}_i)$.

Beispiel (Matching von bipartiten Graphen)**.** Wir betrachten erneut das sogenannte *maximum-matching* Problem (siehe Kap. 2.3). Für einen bipartiten Graphen $G = (V \dot{\cup} U, E)$,

ist ein *matching* sowohl eine unabhängige Menge in $\mathbb{F}_V$ als auch in $\mathbb{F}_U$.
Wenn also $(E, \mathbb{F})$ das zugehörige Unabhängigkeitssystem des bipartiten Graphen G ist, also $\mathbb{F} = \{F \subseteq E : F \text{ ist matching von } G\}$, dann folgt $\mathbb{F} = \mathbb{F}_V \cap \mathbb{F}_U$.
Das Finden des *maximum matchings* eines bipartiten Graphen ist also äquivalent zum Finden eines Elements maximaler Kardinalität in $\mathbb{F} = \mathbb{F}_V \cap \mathbb{F}_U$.

Anmerkung. Der Schnitt von Unabhängigkeitssystemen ist wieder ein Unabhängigkeitssystem (Übungsaufgabe).

Proposition 5.14. *Jedes Unabhängigkeitssystem ist der Schnitt endlich vieler Matroide.*

Beweis. Wir zeigen zuerst, dass jeder Kreis C von $(E, \mathbb{F})$ einen Matroid $(E, \mathbb{F}_C)$ mit $\mathbb{F}_C = \{F \subseteq E : C \setminus F \neq \emptyset\}$ definiert:

- $(M1)$ und $(M2)$ sind klar.
- Wir zeigen $(M3'')$, d.h. zu zeigen ist, dass $\forall A \subseteq E$ gilt: alle maximal unabhängigen Teilmengen von A (bzgl. $\mathbb{F}_C$) haben die gleiche Kardinalität.
  Sei also $A \subseteq E$ und sei $B \subseteq A$ eine maximal unabhängige Teilmenge von A, also insbesondere $C \setminus B \neq \emptyset$. Nun unterscheiden wir zwei Fälle:
  (i) Falls $A \in \mathbb{F}_C$, so folgt $A = B$.
  (ii) Falls $A \notin \mathbb{F}_C$, folgt $C \setminus A = \emptyset$, d.h. $C \subseteq A$.
  Da $B \in \mathbb{F}_C$, gilt $|C \setminus B| \geq 1$.
  Angenommen $|C \setminus B| \geq 2$:
  $\Rightarrow \exists x, y, \in C \setminus B$ mit $x \neq y$
  $\Rightarrow y \in C \setminus (B \cup \{x\})$
  $\Rightarrow C \setminus (B \cup \{x\}) \neq \emptyset$
  $\Rightarrow B \cup \{x\} \in \mathbb{F}_C$
  ↯ Widerspruch zur maximalen Unabhängigkeit von B.
  Dieser Fall kann also nicht auftreten, d.h. für alle maximal unabhängige Teilmengen B, B' von A gilt $|C \setminus B| = 1 = |C \setminus B'|$, also insbesondere $|B| = |B'|$.

Somit ist $(E, \mathbb{F}_C)$ ein Matroid für alle Kreise C von $(E, \mathbb{F})$.

Es bleibt zu zeigen, dass $(E, \mathbb{F}) = (E, \bigcap_{C \in \mathbb{C}} \mathbb{F}_C)$, wobei $\mathbb{C}$ die Menge der Kreise von $(E, \mathbb{F})$ sei.

„$\subseteq$“: Sei $X \in \mathbb{F}$ und sei C ein beliebiger Kreis in $(E, \mathbb{F})$.
Dann gilt $C \not\subseteq X$ und damit $C \setminus X \neq \emptyset$. Also $X \in \mathbb{F}_C$ für alle $C \in \mathbb{C}$, d.h. insbesondere $X \in \cap \mathbb{F}_C$.

„$\supseteq$“: Angenommen $X \in \cap \mathbb{F}_C$, aber $X \notin \mathbb{F}$. Letzteres bedeutet, dass ein Kreis $C_X \subseteq X$ existiert mit $C_X \setminus X = \emptyset$. Daraus folgt $X \notin \mathbb{F}_{C_X}$, d.h. insbesondere $X \notin \cap \mathbb{F}_C$. ↯ Widerspruch.

□

Wir betrachten nun folgendes Problem:

Matroid-Intersektion-Problem
Gegeben: Matroide $(E, \mathbb{F}_1)$ und $(E, \mathbb{F}_2)$ (Unabhängigkeitsorakel)
Ziel: Finde eine Menge $X \in \mathbb{F}_1 \cap \mathbb{F}_2$, sodass $|X| \xrightarrow{!} \max$.

Notation: Für einen Matroid $(E, \mathbb{F})$ bezeichne im Folgenden $C(X, e)$ den eindeutig bestimmten Kreis in $X \cup \{e\}$,falls $X \cup \{e\} \notin \mathbb{F}$. Sonst setzen wir $C(X, e) = \emptyset$. Für die beiden Matroide $(E, \mathbb{F}_1)$ und $(E, \mathbb{F}_2)$ sind dies entsprechend $C_1(X, e)$ und $C_2(X, e)$.
Außerdem verwenden wir ab und zu folgende verkürzende Schreibweise:

$$X + e = X \cup \{e\},\ X - e = X \setminus \{e\}$$

$$X + Y = X \cup Y,\ X - Y = X \setminus Y.$$

Idee des folgenden Algorithmus: Wir beginnen mit $X = \emptyset$ und augmentieren X in jeder Iteration um ein Element. Dazu suchen wir ein Element $e_1 \notin X$, sodass $X \cup \{e_1\} \in \mathbb{F}_1$.

- Falls $X \cup \{e_1\} \in \mathbb{F}_2$, dann setzen wir $X \leftarrow X + e_1$.
- Falls $X \cup \{e_1\} \notin \mathbb{F}_2$:
  $\Rightarrow \exists!$ Kreis $C_2(X, e_1) \subseteq X \cup \{e_1\}$ bzgl. M_2
  $\Rightarrow$ wir wählen ein Element $e_2 \in C_2(X, e_1) \setminus \{e_1\}$
  $\Rightarrow X + e_1 - e_2 \in \mathbb{F}_2$. Außerdem $X + e_1 - e_2 \in \mathbb{F}_1$.

Im nächsten Schritt suchen wir ein Element $e_3 \notin X$, sodass $X + e_1 - e_2 + e_3 \in \mathbb{F}_2$ und fahren iterativ wie oben fort.

Dabei erhält die Hinzunahme der Elemente $e_1, e_3, e_5, \ldots$ die Unabhängigkeit in M_1, während das Entfernen der Elemente $e_2, e_4, e_6, \ldots$ die Unabhängigkeit in M_2 wieder herstellt.

Das Ziel ist es also, eine Sequenz $\mathbb{S} = (e_1, e_2, e_3, \ldots, e_s)$ zu finden, sodass die symmetrische Mengendifferenz $X \triangle \mathbb{S} \in \mathbb{F}_1 \cap \mathbb{F}_2$. Wenn s ungerade ist, so ist $X \triangle \mathbb{S}$ eine größere unabhängige Teilmenge als X von $\mathbb{F}_1 \cap \mathbb{F}_2$.

Um diese Sequenz zu finden, führen wir einen gerichteten bipartiten Hilfsgraphen $G_X = (E, A_X^{(1)} \cup A_X^{(2)})$ ein, wobei:

$$A_X^{(1)} = \{(x, y) : x \in X, y \in E \setminus X, x \in C_1(X, y) \setminus \{y\}\}$$
$$A_X^{(2)} = \{(y, x) : x \in X, y \in E \setminus X, x \in C_2(X, y) \setminus \{y\}\},$$

d.h. wenn $(x, y) \in A_X^{(1)}$, so erhält das Ersetzen von x durch y die Unabhängigkeit in M_1 und analog, wenn $(y, x) \in A_X^{(2)}$, so erhält das Ersetzen von x durch y die Unabhängigkeit in M_2.

Sei nun $Q = (y_0, x_1, y_1, \ldots, x_s, y_s)$ ein Weg in G_X, sodass $y_0, y_s \in E \setminus X$. So gilt per Konstruktion von G_X und da G_X bipartit ist, dass $x_i \in X$ und $y_i \in E \setminus X$ für alle i. Da Q ein Weg ist (also insbesondere keinen Knoten mehrfach besucht), erhalten wir

$$\underbrace{|\{x_1, \ldots, x_s\}|}_{=:\mathbb{X} \subseteq X} + 1 = \underbrace{|\{y_0, \ldots, y_s\}|}_{=:\mathbb{Y}}$$

und damit folgt für $X' =: X - \mathbb{X} + \mathbb{Y}$: $|X'| = |X| + 1$.

Im Folgenden stellen wir an solche Wege Q folgende Anforderungen:

(1) Q startet in einem Knoten aus $S_X \coloneqq \{y \in E \setminus X : X + y \in \mathbb{F}_1\}$,

(2) Q endet in einem Knoten aus $T_X \coloneqq \{y \in E \setminus X : X + y \in \mathbb{F}_2\}$,

(3) Q ist ein kürzester $S_X - T_X-$Weg (d.h. Q ist der kürzeste $a - b-$Weg über alle $a \in S_X$ und $b \in T_X$).

Wege, die diese drei Eigenschaften erfüllen, werden auch *augmentierende Wege* genannt.

Anmerkung. Falls $S_X \cap T_X \neq \emptyset$, so hat der kürzeste augmentierende Weg die Länge 0. Somit können wir ein beliebiges Element $y \in S_X \cap T_X$ nehmen um X zu erweitern.

Im Folgenden wollen wir zeigen, dass für solche augmentierende Wege $Q = (y_0, x_1, y_1, \ldots, x_s, y_s)$, gilt:

$$X' \coloneqq X \cup \{y_0, \ldots, y_s\} \setminus \{x_1, \ldots, x_s\} \in \mathbb{F}_1 \cap \mathbb{F}_2,$$

Zu diesem Zweck beginnen wir mit folgendem Hilfslemma:

Lemma 5.15. *Sei $(E, \mathbb{F})$ ein Matroid und $X \in \mathbb{F}$. Außerdem seien $x_1, \ldots, x_s \in X$ und $y_1, \ldots, y_s \in E \setminus X$ mit*

(a) $x_k \in C(X, y_k)$ für $1 \leq k \leq s$,

(b) $x_j \notin C(X, y_k)$ für $1 \leq j < k \leq s$.

Dann gilt: $X \cup \{y_1, \ldots, y_s\} \setminus \{x_1, \ldots, x_s\} \in \mathbb{F}$.

Beweis. Sei $X_r \coloneqq X \setminus \{x_1, \ldots, x_r\} \cup \{y_1, \ldots, y_r\}$ mit $r \leq s$. Wir zeigen durch vollständige Induktion, dass $X_r \in \mathbb{F}$ für alle r.

Induktionsanfang $r = 0$: $X_r = X \in \mathbb{F}$ $\surd$

Induktionsvoraussetzung: $X_{r-1} \in \mathbb{F}$.

Induktionsschritt $r-1 \to r$:

- Falls $X_{r-1} \cup \{y_r\} \in \mathbb{F}$, folgt $X_r \in \mathbb{F}$, da $X_r \subseteq X_{r-1} \cup \{y_r\} \in \mathbb{F}$.
- Falls $X_{r-1} \cup \{y_r\} \notin \mathbb{F}$, existiert ein eindeutiger Kreis $C(X_{r-1}, y_r) = C(X, y_r)$.
  Wegen Voraussetzung (b) gilt: $x_1, \ldots, x_{r-1} \notin C(X, y_r)$ und
  wegen Voraussetzung (a) gilt: $x_r \in C(X, y_r)$.
  Daraus folgt, dass $C(X, y_r) - x_r$ kein Kreis ist, also $X_{r-1} \cup \{y_r\} \setminus \{x_r\} \in \mathbb{F}$.

□

Lemma 5.16. *Seien $M_1 = (E, \mathbb{F}_1)$ und $M_2 = (E, \mathbb{F}_2)$ Matroide. Sei $X \in \mathbb{F}_1 \cap \mathbb{F}_2$. Sei $Q = (y_0, x_1, y_1, \ldots, x_s, y_s)$ ein augmentierender Weg in G_X (d.h. Q ist kürzester $S_X - T_X -$Weg mit $y_0 \in S_X$ und $y_s \in T_X$). Dann ist $X' \coloneqq X \setminus \{x_1, \ldots, x_s\} \cup \{y_0, \ldots, y_s\} \in \mathbb{F}_1 \cap \mathbb{F}_2$.*

Beweis. Sei Q ein augmentierender Weg in G_X. Wir zeigen zuerst, dass die Bedingungen (a) und (b) aus Lemma 5.15 für M_1 bzw. M_2 erfüllt sind.
Dazu setzen wir $\tilde{X} \coloneqq X \cup \{y_0\}$. Da $y_0 \in S_X$, folgt $\tilde{X} \in \mathbb{F}_1$.
Weiterhin gilt nach Konstruktion von G_X, dass $x_1, \ldots, x_s \in \tilde{X}$ und $y_1, \ldots, y_s \in E \setminus \tilde{X}$.
Da $(x_j, y_j) \in A_X^{(1)}$ und nach Definition von $A_X^{(1)}$ gilt für alle $j = 1, \ldots, s$, dass $x_j \in C_1(X, y_j) \subseteq C_1(\tilde{X}, y_j)$. und somit ist Voraussetzung (a) aus Lemma 5.15 erfüllt.
Es bleibt zu zeigen, dass auch (b) erfüllt ist. Angenommen, dies wäre nicht der Fall. Dann existiert ein $j < k \leq s$, sodass $x_j \in C_1(\tilde{X}, y_k)$.
Beachte dabei, dass

$$\underbrace{\tilde{X} \cup \{y_k\}}_{\notin \mathbb{F}_1} = \underbrace{\underbrace{X \cup \{y_0\}}_{\in \mathbb{F}_1} \cup \{y_k\}}_{\text{enthält höchstens einen Kreis}} .$$

Also $x_j \in C_1(\tilde{X}, y_k) = C_1(X, y_k)$, d.h. es existiert eine Kante $(x_j, y_k) \in A_X^{(1)}$. Daraus folgt aber, dass Q kein kürzester Weg ist, was einen Widerspruch darstellt.
Also erfüllt $\tilde{X}$ auch die Bedingung (b) aus Lemma 5.15.
Wir wenden Lemma 5.15 nun an und erhalten

$$\tilde{X} \cup \{y_1, \ldots, y_s\} \setminus \{x_1, \ldots, x_s\} \in \mathbb{F}_1,$$

also

$$X' = X \cup \{y_0\} \cup \{y_1, \ldots, y_s\} \setminus \{x_1, \ldots, x_s\} \in \mathbb{F}_1.$$

Auf ähnliche Weise kann man zeigen, dass $X' \in \mathbb{F}_2$ (Übungsaufgabe).
Insgesamt folgt damit $X' \in \mathbb{F}_1 \cap \mathbb{F}_2$. □

Proposition 5.17. *Seien $M_1 = (E, \mathbb{F}_1)$ und $M_2 = (E, \mathbb{F}_2)$ Matroide mit Rangfunktionen r_1 bzw. r_2. Dann gilt für alle $F \in \mathbb{F}_1 \cap \mathbb{F}_2$ und $W \subseteq E$, dass*

$$|F| \leq r_1(W) + r_2(E \setminus W).$$

Beweis.

- $F \in \mathbb{F}_1 \Rightarrow F \cap W \in \mathbb{F}_1 \Rightarrow |F \cap W| \leq r_1(W)$
- $F \in \mathbb{F}_2 \Rightarrow F \setminus W \in \mathbb{F}_2 \Rightarrow |F \setminus W| \leq |E \setminus W| \leq r_2(E \setminus W)$

Damit folgt $|F| = |F \setminus W| + |F \cap W| \leq r_1(W) + r_2(E \setminus W)$. □

Theorem 5.18. *Seien $M_1 = (E, \mathbb{F}_1)$ und $M_2 = (E, \mathbb{F}_2)$ Matroide und sei $X \in \mathbb{F}_1 \cap \mathbb{F}_2$. X ist kardinalitätsmaximal genau dann, wenn es keinen $S_X - T_X-$Weg in G_X gibt.*

Beweis.

„⇒“:
Angenommen, es gibt einen $S_X - T_X-$Weg in G_X. Dann existiert insbesondere auch ein kürzester $S_X - T_X-$Weg Q. Mit Lemma 5.16 folgt dann aber, dass wir ein X' konstruieren können mit $X' \in \mathbb{F}_1 \cap \mathbb{F}_2$ und $|X'| > |X|$.
↯ Widerspruch zur Maximalität von X.

„⇐“:
Sei R die Menge der von S_X aus erreichbaren Knoten, d.h.

$$R = \{z \in E \mid \exists\, S_X - z - \text{Weg in } G_X\}.$$

Da nach Voraussetzung kein $S_X - T_X-$Weg existiert, gilt $R \cap T_X = \emptyset$.
Seien nun r_1 und r_2 die Rangfunktionen von M_1 bzw. M_2.

Wir zeigen zuerst, dass $r_2(R) = |X \cap R|$.
Da $X \in \mathbb{F}_2$, folgt mit (M2), dass $X \cap R \in \mathbb{F}_2$ und wir erhalten $|X \cap R| \leq r_2(R) = \max\{|Y| : Y \in \mathbb{F}_2, Y \subseteq R\}$. Angenommen $r_2(R) > |X \cap R|$.
Dann existiert ein $Y \subseteq R$, $Y \in \mathbb{F}_2$, sodass $|Y| > |X \cap R|$.
Mit (M3) folgt: $\exists\, y \in Y \setminus (X \cap R) = Y \setminus X \cup \underbrace{Y \setminus R}_{=\emptyset} = Y \setminus X$,
sodass $(X \cap R) \cup \{y\} \in \mathbb{F}_2$. Wegen $Y \subseteq R$ gilt: $y \in R \setminus X$.
Da $y \in R$, aber $R \cap T_X = \emptyset$, folgt $y \notin T_X$. D.h. nach Definition von T_X ist $X \cup \{y\} \notin \mathbb{F}_2$.
Daraus folgt, dass ein eindeutig bestimmter Kreis $C_2(X, y)$ in $X \cup \{y\}$ existiert.
Da $X \cap R \in \mathbb{F}_2$, $\{y\} \in \mathbb{F}_2$ und $X \not\subseteq R$ folgt, dass ein $x \in X \setminus R$ existiert, sodass $x \in C_2(X, y) \setminus \{y\}$.
Nach Definition von $A_X^{(2)}$, gibt es also eine Kante (y, x) in $A_X^{(2)}$. Zusammenfassend erhalten wir also:

- Es gibt eine Kante (y, x) in $A_X^{(2)}$.
- $y \in R \Rightarrow \exists\, S_X - y-$Weg
- $x \notin R \Rightarrow \nexists\, S_X - x-$Weg

Dies ist aber ein Widerspruch, denn der $S_X - y-$Weg und die Kante (y, x) ergeben zusammen einen $S_X - x-$Weg. ↯

Auf ähnliche Weise zeigt man, dass $r_1(E \setminus R) = |X \setminus R|$ (Übungsaufgabe).

Zusammen erhalten wir

$$\begin{aligned} |X| &= |X \cap R| + |X \setminus R| \\ &= r_2(R) + r_1(E \setminus R). \end{aligned}$$

Wegen Proposition 5.17, muss X also kardinalitätsmaximal sein.

□

Als Folgerung erhalten wir die sogenannte *Min-Max-Gleichung*:

Theorem 5.19. *Seien $M_1 = (E, \mathbb{F}_1)$ und $M_2 = (E, \mathbb{F}_2)$ Matroide mit Rangfunktionen r_1 bzw. r_2. Dann gilt*

$$\max\{|X| : X \in \mathbb{F}_1 \cap \mathbb{F}_2\} = \min\{r_1(Q) + r_2(E \setminus Q) : Q \subseteq E\}$$

Nun können wir einen Algorithmus zur Lösung des Matroid-Intersektions-Problems angeben:

Algorithm 9 EDMOND's Matroid-Intersektions-Algorithmus

Input: Matroide $(E, \mathbb{F}_1)$ und $(E, \mathbb{F}_2)$, gegeben durch ein Unabhängigkeitsorakel.
Output: Menge $X \in \mathbb{F}_1 \cap \mathbb{F}_2$, sodass $|X|$ maximal ist.

1: $X \leftarrow \emptyset$
2: **for** alle $y \in E \setminus X$ und $i = 1, 2$ **do**
3: Berechne $C_i(X, y) \coloneqq \{x \in X \cup \{y\} : X \cup \{y\} \notin \mathbb{F}_i, (X \cup \{y\}) \setminus \{x\} \in \mathbb{F}_i\}$.
4: **end for**
5: Berechne S_X, T_X und G_X.
6: Bestimme einen kürzesten $S_X - T_X-$Weg Q in G_X.
7: **if** $\nexists$ Weg Q **then**
8: **return** X
9: **else**
10: $X \leftarrow X \triangle Q$ und gehe zu Schritt 2.
11: **end if**

Theorem 5.20. *EDMOND's Matroid-Intersektions-Algorithmus ist korrekt und hat Laufzeit $\mathcal{O}(|E|^3\theta)$, wobei θ die maximale Komplexität der beiden Unabhängigkeitsorakel ist.*

Beweis.

- Die Korrektheit folgt aus Lemma 5.16 und Theorem 5.18.
- Laufzeit ist Übungsaufgabe.

□

Anmerkungen.

- Es gibt schnellere Algorithmen, z.B. von Cunningham (1986) oder Garbow und Xu (1996).
- Das Matroid-Intersektions-Problem ist $\mathcal{NP}$-schwer für $k \geq 3$ Matroide.
  Beweisidee: Reduktion vom Problem Hamiltonischer Weg
  $\Rightarrow$ Definiere zwei Partitionsmatroide und einen graphischen Matroid.

$$X \in \mathbb{F}_1 \cap \mathbb{F}_2 \cap \mathbb{F}_3 \Leftrightarrow |X| = |V(G) - 1|$$

Abschließendes Beispiel – Maximum matching und Vertex Cover in bipartiten Graphen
Sei $G = (X \dot{\cup} Y, E)$ ein bipartiter Graph. Gesucht ist ein *maximum matching* von G.

$\rightarrow$ Berechne die Partitionsmatroide $(E, \mathbb{F}_X)$ und $(E, \mathbb{F}_Y)$

$\rightarrow$ Verwende den EDMONDS-Algorithmus, um ein *maximum matching* in $\mathbb{F}_X \cap \mathbb{F}_Y$ zu finden.

Natürlich ist dies nur einer von vielen Wegen, um ein *maximum matching* zu finden. Schneller ist z.B. der Hopcroft-Karp-Algorithmus mit Laufzeit $\mathcal{O}(\sqrt{n}m)$, wobei $n = |V|$ und $m = |E|$.

Im Folgenden betrachten wir nun die Frage, wie man in bipartiten Graphen ein *minimum vertex cover* mittels eines *maximum matchings* finden kann.

Sei dazu $G = (X \dot{\cup} Y, E)$ ein bipartiter Graph und sei M ein maximum matching von G. Wir definieren nun die folgenden Mengen:

- Ein M-alternierender Weg ist dabei ein Weg, in dem sich Kanten aus M und aus $E \setminus M$ abwechseln.
- $U \subseteq X$ Menge von Knoten aus X, die nicht inzident zu einer Kante aus M sind
- $T \subseteq Y$ Menge von Knoten aus Y, die von $u \in U$ über einen M-alternierenden Weg erreichbar sind.
- $S \subseteq X$ Menge von Knoten aus X, die von $u \in U$ über einen M-alternierenden Weg erreichbar sind.

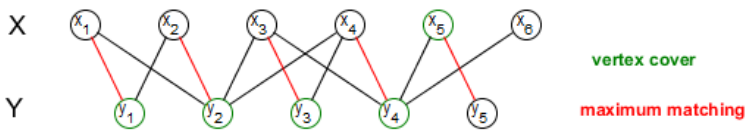

Abbildung 5.3: Zusammenhang von *maximum matching* und *vertex cover* in bipartiten Graphen

Beispiel. Betrachte den Graphen $G = (X \dot\cup Y, E)$ aus Abbildung 5.3. Hier ist $U = \{x_6\}$, $S = \{x_1, x_2, x_3, x_4, x_6\}$ und $T = \{y_1, y_2, y_3, y_4\}$. Ein *minimum vertex cover* ergibt sich z.B. als $VC = (X - S) \cup T$.

Theorem 5.21. *Sei $G = (X, \dot\cup Y, E)$ ein bipartiter Graph und sei M ein* maximum matching *von G. Dann ist $C = (X - S) \cup T$ ein minimum vertex cover von G.*

Beweis. Wir zeigen zunächst, dass C ein *vertex cover* ist.
Sei $e = (u, v) \in E$. Da G bipartit, muss einer der Knoten in X und der andere in Y. Sei $u \in X$ und $v \in Y$ sein. Angenommen C ist kein *vertex cover*, also $u, v \notin C$. Somit gilt $u \in S$ und $v \in Y - T$, d.h. . Nun unterscheiden wir 2 Fälle:

- $e \notin M$:
  Da $u \in S$, existiert ein M-alternierender Weg von $u' \in U$ zu u. Nach Definition von U sind alle zu u' inzidenten Kanten nicht in M. Daraus muss aber folgen, dass die letzte Kante auf dem M-alternierenden Weg von u' zu u eine Kante aus M ist. D.h. insbesondere gibt es also auch einen M-alternierenden Weg von u' zu v. Das ist aber ein Widerspruch zu $v \in Y - T$ ↯.

- $e \in M$:
  Da $u \in S$, folgt analog zu obigem Fall, dass ein M-alternierender Weg von $u' \in U$ zu u existiert. Da $e = (u, v) \in M$, muss es sich bei e um die letzte Kante auf dem M-alternierenden Weg von u' zu u handeln. Das bedeutet aber wieder, dass es auch einen M-alternierenden Weg von u' zu v gibt. Widerspruch zu $v \in Y - T$ ↯.

D.h. obige Annahme war falsch und C ist ein *vertex cover* von G.

Es bleibt zu zeigen, dass C auch ein *vertex cover* minimaler Kardinalität ist. Wir hatten bereits gezeigt, dass $|M| \leq |C|$ (siehe Abschnitt 2.4). Nun wollen wir zeigen, dass $|M| = |C|$.

(i) Sei $(u, v) \in M$.
  - Falls $v \in T$, so folgt $u \in S$, also $u \notin C$.
  - Falls $v \notin T$, folgt $v \notin C$.

  $\Rightarrow$ Jede Kante aus M hat also höchstens einen Endknoten in C, also $|M| \leq |C|$.

(ii) Sei $v \in C$.
  - Falls $v \in T$, dann folgt, dass es eine Kante in M gibt, die inzident zu v ist (falls nicht, wäre M noch kein optimales matching (ÜA)).
  - Falls $v \in X - S$, so ist $v \in X$ nicht erreichbar über M-alternierende Wege von einem $u \in U \subseteq X$, also insbesondere $v \notin U$ (der Weg mit leerer Kantenmenge ist erlaubt). Da U alle Knoten beinhaltet, die zu keiner Matching-Kante inzident sind, muss also v inzident zu einer Matching-Kante sein.

  $\Rightarrow$ Jeder Knoten aus C ist zu einer Kante aus M inzident.

Mit (i) und (ii) folgt, dass $|M| = |C|$. □

6 Approximationsalgorithmen

Im Allgemeinen kann jeder Algorithmus (für ein Optimierungsproblem Π), welcher eine zulässige Lösung liefert, als Approximationsalgorithmus bezeichnet werden.

Im Folgenden werden wir aber nur Approximationsalgorithmen mit polynomieller Laufzeit betrachten und deren Güte bestimmen.

Notation: Sei Π ein Optimierungsproblem und sei $I \in \Pi$ eine Instanz von Π. Dann bezeichnen wir mit $OPT(I)$ die optimale Lösung und mit $A(I)$ die Lösung des Approximationsalgorithmus. Dabei gilt für alle $I \in \Pi$:

- $A(I) \geq OPT(I)$, falls Π Minimierungsproblem
- $A(I) \leq OPT(I)$, falls Π Maximierungsproblem

6.1 Approximation mit absoluter Güte

Definition (absolute Güte)**.** Sei Π ein Optimierungsproblem und sei A ein Approximationsalgorithmus für Π. A hat *absolute Güte* k, falls für alle $I \in \Pi$ gilt:

$$| A(I) - OPT(I) | \leq k.$$

Anmerkung. Falls für k aus obiger Definition gilt

- $k = \text{const} \Rightarrow$ *konstante absolute Güte*
- $k = c\log(|I|) \Rightarrow$ *logarithmisch absolute Güte* ($|I| \hat{=}$ Eingabegröße)
- $k = |I|^c \Rightarrow$ *polynomiell absolute Güte*
- $k = 0 \Rightarrow A$ ist *exakter* Algorithmus

6.1.1 Knotenfärbung

Definition (Knotenfärbung; chromatische Zahl)**.** Sei $G = (V, E)$ ein Graph. Eine *Knotenfärbung* von G ist eine Abbildung $\gamma : V(G) \to C$ von der Knotenmenge $V(G)$ in eine Menge C von Farben, sodass für alle $(u, v) \in E$ gilt: $\gamma(u) \neq \gamma(v)$ (Eine solche Knotenfärbung existiert immer. Setze dazu $C = V$).

Eine Knotenfärbung γ, die höchstens k Farben verwendet, heißt *k-Färbung* von G, d.h. $|C| \leq k$.
Das kleinste k, für welches eine Knotenfärbung γ von G existiert, heißt *chromatische Zahl* $\chi(G)$.

Dies führt uns auf das Minimum-Knotenfärbung-Problem:

Minimum-Knotenfärbung-Problem
Gegeben: Graph $G = (V, E)$
Ziel: Bestimme die chromatische Nummer $\chi(G)$ (und die entsprechende Knotenfärbung $\gamma : V \to C = \{1, \ldots, \chi(G)\}$ von G).

Beispiel (G bipartit)**.** Sei $G = (V, E)$ bipartit. Dann gilt $\chi(G) \leq 2$.

Für allgemeine Graphen gilt der folgende Satz:

Theorem (Brook, 1941)**.** *Für alle Graphen $G = (V, E)$ gilt*

$$\chi(G) \leq \Delta(G) + 1,$$

wobei $\Delta(G)$ der maximale Grad eine Knotens in G ist.

Beweis. Induktion über die Knotenanzahl $|V|$:

Induktionsanfang:
Sei $|V| = 1$. Dann gilt $\chi(G) = 1$ und $\Delta(G) = 0$, d.h. insbesondere $\chi(G) \leq \Delta(G) + 1$.

Induktionsannahme:
Die Behauptung $\chi(G) \leq \Delta(G) + 1$ gilt für alle Graphen mit $|V| < k$, $k \in \mathbb{N}$.

Induktionsschritt:
Sei $G = (V, E)$ ein Graph mit $|V| = k$ und $v \in V$ ein beliebiger Knoten.
Dann können wir den Graphen $G - v$ betrachten, der entsteht, wenn v und alle zu v inzidenten Kanten aus G gelöscht werden. Offensichtlich gilt $|V(G - v)| < k$ und nach Induktionsannahme gilt $\chi(G-v) \leq \Delta(G-v)+1$, d.h. es gibt eine $\Delta(G-v)+1$-Färbung für $G - v$.
Wir wissen außerdem, dass der Knoten v höchstens $\Delta(G)$ Nachbarn in G hat, d.h. $\Delta(G - v) \leq \Delta(G)$. Nun können wir zwei Fälle unterscheiden:

(i) $\Delta(G - v) = \Delta(G)$:
In $G - v$ werden höchstens $\Delta(G - v) + 1 = \Delta(G) + 1$ Farben benutzt. Selbst wenn alle $\Delta(G)$ Nachbarn von v unterschiedliche Farben haben, bleibt eine Farbe für v übrig, d.h. $\chi(G) \leq \Delta(G) + 1$.

(ii) $\Delta(G - v) < \Delta(G)$:
$\Leftrightarrow \Delta(G - v) + 1 \leq \Delta(G)$.

In $G - v$ werden höchstens $\Delta(G)$ Farben benutzt. Aber selbst wenn alle $\Delta(G)$ Nachbarn von v unterschiedlich gefärbt sind und wir für v eine neue Farbe einführen müssen, gilt $\chi(G) \leq \Delta(G) + 1$.

□

Minimum-Knotenfärbung-Problem als Entscheidungsproblem

Das Minimum-Knotenfärbungs-Problem lässt sich wie folgt als Entscheidungsproblem formulieren:

Min-V-col
Gegeben: Graph $G = (V, E)$, natürliche Zahl $k \in \mathbb{N}$
Frage: Gilt $\chi(G) \leq k$?

Theorem. *Das Entscheidungsproblem Min-V-col ist $\mathcal{NP}$-vollständig für $k = 3$.*

Beweisidee.

- Min-V-col $\in \mathcal{NP}$ $\surd$
- Min-V-col ist $\mathcal{NP}$-schwer:
  Um die $\mathcal{NP}$-Schwere zu zeigen, nutzen wir folgende Reduktion:

$$3SAT \leq_p \textit{Min-V-col}(k = 3)$$

  Sei $\varphi \in 3SAT$, $\varphi = c_1 \wedge \ldots \wedge c_m$, $|c_i| = 3$ mit booleschen Variablen $x_1, \ldots, x_n$.
  Die Idee ist nun, einen *Skelettgraph* für die booleschen Variablen zu bauen, sowie einen *gadget graph* für jede Klausel $c_i = (l_i^1 \vee l_i^2 \vee l_i^3)$ (siehe Tafel).
  Damit kann man nun zeigen (Übungsaufgabe), dass

$$\varphi \text{ erfüllbar } \Leftrightarrow G_\varphi \text{ ist 3-färbbar}$$

□

Korollar. *Min-V-col ist $\mathcal{NP}$-vollständig für $k \geq 3$.*

Beweis. Übungsaufgabe. □

Wir geben nun einen Approximationsalgorithmus zur Lösung des Minimum-Knotenfärbung-Problems an:

Theorem.

(1) Approx_Knotenfärbung liefert eine Knotenfärbung γ mit höchstens $\Delta(G) + 1$ Farben.

(2) Approx_Knotenfärbung hat eine absolute Güte von $\Delta(G) - 1$.

Algorithm 10 Approx_Knotenfärbung(Graph $G = (\{u_1, \ldots, u_n\}, E)$ (zusammenhängend), $n > 1$)

1: Setze $\gamma(u_i) = \infty \, \forall \, 1 \leq i \leq n$
2: **for** $i = 1, \ldots, n$ **do**
3: $\gamma(u_i)$ = kleinste Farbe, die keiner der schon gefärbten Nachbarn von u_i hat (wobei $\gamma(u_1) := 0$).
4: **end for**
5: **return** Knotenfärbung γ

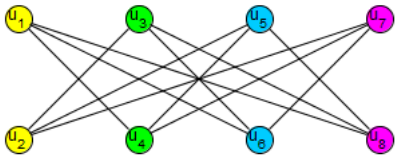

Abbildung 6.1: Knotenfärbung mittels Approx_Knotenfärbung: $A(I) = 4$ (optimal wäre $OPT(I) = 2$)

Beweis.

(1) Wenn der Algorithmus bei Knoten u_i ankommt, dann können nicht alle Farben $\{1, \ldots, \Delta(G)+1\}$ an die Nachbarn von u_i vergeben sein (denn u_i hat höchstens $\Delta(G)$ viele Nachbarn). Also existiert eine kleinste (Farbe in $\{1, \ldots, \Delta(G) + 1\}$, die für u_i verwendet werden kann.

(2) Betrachte die optimale Lösung $OPT(I)$ und die Lösung von Approx_Knotenfärbung $A(I)$. Dann gilt:

$$2 \leq OPT(I) \leq A(I) \leq \Delta(G) + 1.$$

Daraus folgt

$$A(I) - OPT(I) \leq \Delta(G) + 1 - 2 = \Delta(G) - 1$$

□

Anmerkung. Die Schranke aus Teil (2) des obigen Satzes ist scharf, d.h. es gibt Instanzen mit $A(I) - OPT(I) = \Delta(G) - 1$. Betrachte dazu z.B. den bipartiten Graphen aus Abbildung 6.1. Hier erhalten wir $OPT(I) = 2$, $\Delta(G) = 3$, $A(I) = 4$ und damit $A(I) - OPT(I) = \Delta(G) - 1 = 2$.

Spezialfall: Planare (ungerichtete) Graphen

Definition (planare Einbettung)**.** Eine *planare Einbettung* eines Graphen ist eine „Zeichnung“ in der 2-dim. Ebene $\mathbb{R}^2$, sodass sich verschiedene Kanten nur in gemeinsamen Endpunkten schneiden (formal: Kanten $\hat{=}$ Jordankurven).
Ein Graph $G = (V, E)$ heißt *planar*, wenn es eine planare Einbettung von G gibt.

Beispiele (Planare und nicht planare Graphen)**.**

- Planare Graphen: K_4, Q_3
- Nicht planare Graphen: K_5, $K_{3,3}$

Anmerkung. Jede planare Einbettung von G zerlegt die Ebene in zusammenhängende Bereiche, sogenannte *Flächen* oder *Gebiete*.
Das Problem der planaren Einbettung von Graphen und deren Färbung geht zurück auf das bekannte 4-Farben-Problem (Erläuterungen siehe Tafel)

Theorem (Eulersche Polyederformel)**.** *Sei $G = (V, E)$ ein zusammenhängender, planarer Graph mit Flächenmenge F. Dann gilt:*

$$|V| + |F| - |E| = 2$$

Beweis. Induktion über $|F|$:

Induktionsanfang:
Sei $|F| = 1$. Dann ist G ein Baum, d.h. $|E| = |V|-1$. Daraus folgt aber $|V|+|F|-|E| = 2$.

Induktionsannahme:
Die Behauptung gilt für alle Graphen mit $\leq |F| - 1$ Flächen.

Induktionsschritt:
Sei $G = (V, E)$ planar und $|F| \geq 2$.
Da G zusammenhängend und $|F| \geq 2$, ist G kein Baum, d.h. es existiert ein Kreis in G. Nun löschen wir eine beliebige Kante (v_i, v_{i+1}) aus diesem Kreis und erhalten so einen Graphen G' mit $|E|-1$ Kanten und $|F|-1$ Flächen (Durch „Öffnen“ des Kreises, wurden zwei Flächen vereinigt).
Nach Induktionsannahme gilt für G':

$$\begin{aligned} &|V| + (|F| - 1) - (|E| - 1) = 2 \\ \Leftrightarrow\ &|V| + |F| - |E| = 2. \end{aligned}$$

□

Anmerkungen.

- Wenn G ein planarer, nicht-zusammenhängender Graph mit k Zusammenhangskomponenten ist, so gilt:

$$|V| - |E| + |F| = k + 1 \text{ (Übungsaufgabe)}$$

- Jede planare Einbettung eines planaren Graphen G hat die gleiche Anzahl an Flächen.

Definition (Länge einer Fläche)**.** Sei $G = (V, E)$ planar und f eine Fläche von G. Dann ist die *Länge von f* definiert als

$l(f)$ = Anzahl der Kanten auf dem Rand, der f umschließt, wobei Kanten, die „beide“ Seiten auf f haben, doppelt gezählt werden.

Anmerkung. Für jede planare Einbettung von G gilt (Übungsaufgabe):

$$\sum_{f \in F} l(f) = 2 \cdot |E|$$

Theorem (Obere Schranke für $|E|$)**.** *Sei $G = (V, E)$ planar und $|V| \geq 3$. Dann gilt:*

$$|E| \leq 3|V| - 6$$

Beweis. oBdA sei G zusammenhängend (Falls nicht, konstruiere einen neuen Graphen G' durch Einfügen von Kanten in G, die die Zusammenhangskomponenten verbinden. Wenn die Aussage nun für G' gilt, gilt sie auch für G, da die Knotenanzahl in G unverändert und die Kantenanzahl geringer ist (im Vergleich zu G')).
Da G zusammenhängend ist und mindestens drei Knoten enthält, gilt $l(f) \geq 3$ für alle $f \in F$, wobei F die Flächenmenge von G ist.

$$\Rightarrow 2|E| = \sum_{f \in F} l(f) \geq 3|F|$$

Mit der Eulerschen Polyederformel folgt nun:

$$3|F| = 6 + 3|E| - 3|V|,$$

also insbesondere

$$|E| \leq 3|V| - 6.$$

□

Beispiel. Für den Graphen K_5 gilt die Abschätzung nicht: $|E| = 10$, $|V| = 5$ und damit $|E| > 3|V| - 6$. Somit ist K_5 nicht planar.

Lemma. *Sei $G = (V, E)$ planar. Dann enthält G mindestens einen Knoten $v \in V$ mit $deg(v) \leq 5$.*

Beweis. Angenommen für alle $v \in V$ gilt $deg(v) \geq 6$. Dann folgt: $2|E| = \sum_{v \in V} deg(v) \geq 6|V|$ und somit $|E| \geq 3|V| > 3|V| - 6$ ↯Widerspruch. □

Lemma. *Sei $G = (V, E)$ planar und zusammenhängend. Dann kann G mit höchstens 5 Farben gefärbt werden (d.h. G ist 5-färbbar).*

Beweis. Induktion über $|V|$:

Trivialfall:

Für jeden planaren Graphen mit maximal 5 Knoten ist die Behauptung wahr.

Induktionsvoraussetzung:

Zusammenhängende planare Graphen mit $|V| = n - 1$ sind 5-färbbar.

Induktionsschritt ($n - 1 \to n$):

Sei nun G ein zusammenhängender planarer Graph mit $|V| = n > 5$. Nach Lemma 6.1.1 existiert in G ein Knoten $u \in V$ mit $deg(u) \leq 5$. Wir löschen u aus G und erhalten einen Graphen $G' = G - u$ mit $n - 1$ Knoten, der nach Induktionsvoraussetzung 5-färbbar ist. Nun unterscheiden wir verschiedene Fälle:

- Fall 1: $deg(u) \leq 4$. Dann können wir u eine Farbe zuweisen, die sich von der seiner Nachbarn unterscheidet.
- Fall 2: $deg(u) = 5$, aber die Nachbarknoten sind nur mit vier Farben gefärbt. Dann können wir u die fünfte Farbe zuweisen.
- Fall 3: $deg(u) = 5$ und die Nachbarn von u sind mit fünf Farben gefärbt. Wir bezeichnen die Nachbarknoten von u mit $v_1, \ldots, v_5$ entsprechend der Ordnung bzgl. ihrer Einbettung. OBdA habe Knoten v_i die Farbe i, $1 \leq i \leq 5$. Für zwei verschiedene Farben i und j definieren wir $G_{i,j}$ als den Teilgraphen von $G - u$, der nur aus den mit i oder j gefärbten Knoten und allen Kanten zwischen diesen Knoten besteht. Nun betrachten wir $G_{1,3}$:
  - 1. Fall: v_1 und v_3 gehören nicht zur gleichen Zusammenhangskomponente, d.h. es gibt keinen Weg von v_1 zu v_3. Insbesondere sind v_1 und v_3 auch nicht adjazent. Dann färben wir in der Zusammenhangskomponente, die v_3 beinhaltet, alle Knoten mit Farbe 1 oder 3 um: Aus 1 wird 3 und aus 3 wird 1. v_1 ist von dieser Umfärbung nicht betroffen, da dieser Knoten in einer anderen Zusammenhangskomponente liegt. Da nun also v_1 und v_3 beide die Farbe 1 tragen, kann der Knoten u mit der Farbe 3 gefärbt werden und wir erhalten eine zulässige 5-Färbung von G.
  - 2. Fall: v_1 und v_3 liegen in derselben Zusammenhangskomponente, d.h. es gibt einen Weg P_1 von v_1 zu v_3. Nun betrachten wir zusätzlich $G_{2,4}$:
    - Falls v_2 und v_4 zu unterschiedlichen Zusammenhangskomponenten gehören, färben wir analog zu oben alle Knoten der Farben 2 und 4 um, die zur selben Zusammenhangskomponente gehören wie v_4. Dadurch tragen dann die Knoten v_2 und v_4 beide die Farbe 2, sodass wir u mit der Farbe 4 färben können und eine zulässige 5-Färbung erhalten.
    - Falls v_2 und v_4 in derselben Zusammenhangskomponente liegen, so gibt es einen Weg P_2 von v_2 zu v_4. Aufgrund der Planarität von G und der Anordnung der Knoten $v_1, \ldots, v_5$ um u, muss es dann aber einen Knoten x geben, an dem sich die Wege P_1 und P_2 schneiden. Da dieser auf P_1 liegt, hat er die Farbe 1 oder 3. Da x aber auch auf P_2 liegt, hat er auch die Farbe 2 oder 4. Widerspruch. ↯

□

Folgende zwei Theorem werden ohne Beweis gegeben.

Theorem (4-Farben-Satz)**.** *Jeder planare Graph ist 4-färbbar.*

Theorem. *Das Entscheidungsproblem, ob ein gegebener planarer Graph 3-färbbar ist oder nicht ist $\mathcal{NP}$-vollständig.*

Algorithm 11 Approx_V_col_planar (Graph $G = (V, E)$ zusammenhängend und planar)

```
1: if G ist 2-färbbar then
2:     Färbe G mit 2 Farben.
3: else
4:     Färbe G mit (höchstens) 5 Farben.
5: end if
```

Theorem. *Approx_V_col_planar liefert eine Knotenfärbung mit höchstens 5 Farben und hat absolute Güte 2.*

Beweis.

- Korrektheit $\surd$
- Güte:
  - Falls G 2-färbbar: $A(G) = 2$ und $OPT(G) = 2$, d.h. der Algorithmus ist exakt.
  - Falls G nicht 2-färbbar: $A(G) \leq 5$ und $OPT(G) \geq 3 \rightarrow$ absolute Güte ist 2.

□

6.2 Nichtapproximierbarkeit mit absoluter Güte

Ziel: Wir wollen eine Beweisstrategie herleiten, um zu zeigen, dass ein Problem Π nicht approximierbar mit absoluter Güte ist. Dazu betrachten wir zunächst zwei Beispiele: das CLIQUE-Problem und das Vertex-Cover-Problem.

Lemma. *Wenn $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen Approximationsalgorithmus (mit polynomieller Laufzeit) mit absoluter Güte für das Problem CLIQUE.*

Beweis. Sei $G = (V, E)$ ein Graph und $k \in \mathbb{N}$ beliebig. Wir konstruieren einen Graphen G^k wie folgt:
G^k besteht aus k disjunkten Kopien (V_i, E_i) von G $(V_i = V, E_i = E)$, deren Knoten untereinander alle paarweise verbunden sind (d.h. G^k ist der *join* von k Kopien von G).
Weiter bezeichnen wir mit $\omega(G)$ die Anzahl der Knoten einer größten Clique in G. Es ist klar, dass $\omega(G) = s \Leftrightarrow w(G^k) = ks$.
Nun nehmen wir an, dass es einen Polynomialzeitalgorithmus mit absoluter Güte k für das

Problem CLIQUE gibt. Diesen Algorithmus wenden wir auf G^{k+1} an und erhalten eine Clique der Größe $|A(G^{k+1})|$. Also gilt:

$$\begin{aligned} & \omega(G^{k+1}) - A(G^{k+1}) \leq k \\ \Leftrightarrow\ & (k+1)\omega(G) - A(G^{k+1}) \leq k \\ \Leftrightarrow\ & 0 \leq \omega(G) - \frac{A(G^{k+1})}{k+1} \leq \frac{k}{k+1} < 1\,(*) \end{aligned}$$

Sei C' eine Clique der Größe $A(G^{k+1})$, die wir in G^{k+1} mit dem Algorithmus A gefunden haben. Wir zeigen nun, dass man, gegeben C', eine optimale Clique in G findet und zwar in polynomieller Zeit. Dazu verwenden wir folgenden Algorithmus:

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

1: Konstruiere G^{k+1}.
2: Wende A auf G^{k+1} an und finde so Clique C' in G^{k+1}
3: Setze $C_i = C' \cap V_i$, $1 \leq i \leq k+1$
4: Setze C auf ein C_i mit maximaler Knotenanzahl
5: Return C

Noch zu zeigen ist, dass C eine größte Clique in G ist. Zur vereinfachten Schreibweise werden wir $A(G^{k+1})$ statt $|A(G^{k+1})|$ schreiben.
Wir zeigen zunächst $|C| \geq \frac{A(G^{k+1})}{k+1}$. Angenommen dies wäre nicht der Fall, d.h.

$$|C| < \frac{A(G^{k+1})}{k+1} \Leftrightarrow |C' \cap V_i| < \frac{A(G^{k+1})}{k+1}\, \forall\, 1 \leq i \leq k+1.$$

Es gilt aber

$$|C'| = \sum_{i=1}^{k+1} |C' \cap V_i| < \sum_{i=1}^{k+1} \frac{A(G^{k+1})}{k+1} = A(G^{k+1})\, \lightning$$

Also gilt $|C| \geq \frac{A(G^{k+1})}{k+1}$ und wegen $(*)$ auch $\omega(G) - \frac{A(G^{k+1})}{k+1} \leq \frac{k}{k+1} < 1$.
Insgesamt folgt also $0 \leq \omega(G) - |C| < 1$.
Da $\omega(G)$ und $|C|$ natürliche Zahlen sind, folgt $\omega(G) = |C|$. Also ist C eine größte Clique in G und damit $\mathcal{P} = \mathcal{NP}$ $\lightning$ □

Lemma. *Wenn $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen Approximationsalgorithmus (mit polynomieller Laufzeit) mit absoluter Güte für das Rucksack-Problem* Π
(Anmerkung: Das Rucksack-Problem (Entscheidungsproblem) ist $\mathcal{NP}$-vollständig).

Beweis. Sei $I \in \Pi$ eine Instanz des Rucksack-Problems, d.h. wir haben n Gegenstände $1, \ldots, n$ mit Gewicht w_i und Wert b_i, $1 \leq i \leq n$ und eine Kapazität C (vgl. Problem 0.3). Nun nehmen wir an, dass ein Approximationsalgorithmus mit polynomieller Laufzeit und absoluter Güte k für dieses Problem existiert.
Wir konstruieren eine neue Instanz I' des Problems mit Gegenständen $1, \ldots, n$, Gewichten w_i, $1 \leq i \leq n$, Werten $b'_i = (k+1)b_i$, $1 \leq i \leq n$ und Kapazität C.

Es ist klar, dass jede zulässige Lösung von I auch eine zulässige Lösung von I' ist (da nur die Werte der Gegenstände verändert, während Gewichte und Kapazität unverändert).
Sei $S \subseteq \{1, \ldots, n\}$ eine zulässige Lösung. Dann gilt:

- alte Instanz I: $f_I(S) = \sum_{s \in S} b_s$
- neue Instanz I': $f_{I'}(S) = \sum_{s \in S} b'_s = \sum_{s \in S} (k+1) b_s = (k+1) f_I(S)$

Nun wenden wir den Algorithmus A auf I' an und erhalten:

$$\begin{aligned} & OPT(I') - A(I') \leq k \\ \Leftrightarrow\, & (k+1)OPT(I) - (k+1)A(I) \leq k \\ \Leftrightarrow\, & 0 \leq OPT(I) - A(I) < 1 \end{aligned}$$

Das $\mathcal{NP}$-vollständige Rucksack-Problem kann also in polynomieller Zeit exakt gelöst werden und damit $\mathcal{P} = \mathcal{NP}$. ↯ □

Obige Beispiele/Beweise enthalten typische Vorgehensweisen, um zu zeigen, dass ein Optimierungsproblem Π „schlecht“ approximierbar (d.h. nicht mit absoluter Güte approximierbar) ist. Wir haben gezeigt, dass das Finden der exakten Lösung nicht schwieriger ist als das Problem mit absoluter Güte zu approximieren.

Oft liegt ein Optimierungsproblem Π in folgender Form vor:
Finde $X^* \subseteq X$ zu gegebener Kostenfunktion $c : X \to \mathbb{Z}$, sodass $c(X^*) = \sum_{x \in X^*} c(x) \overset{!}{\to}$ min / max. Dann gilt:

Π ist approximierbar mit absoluter Güte $\Leftrightarrow$ Π kann optimal in polynomieller Zeit gelöst werden

Beweisskizze. Sei I eine Instanz von Π (mit „Grundmenge“ X und Kostenfunktion c). Angenommen, es existiert ein Polynomialzeitalgorithmus mit absoluter Güte k, d.h.

$$|A(I) - OPT(I)| \leq k \,\forall\, I \in \Pi$$

Dann kann Π in polynomieller Zeit exakt gelöst werden:
Baue eine Instanz I' von Π mit Grundmenge X und Kostenfunktion c', wobei $c'(x) = (k+1)c(x)$ für alle $x \in X$. Sei X^* eine optimale Lösung von I', die mittels des Approximationsalgorithmus A gefunden wird und $|A(I') - OPT(I')| \leq k$. Dann folgt

$$\begin{aligned} & |A(I') - OPT(I')| \leq k \\ \Leftrightarrow\, & |(k+1)c(X^*) - (k+1)OPT(I')| \leq k \end{aligned}$$

Also $A(I) = c(X^*) = OPT(I)$, d.h. I kann in polynomieller Zeit exakt gelöst werden. □

6.3 Approximation mit relativer Güte

Im Folgenden lockern wir den Begriff Approximation und betrachten Algorithmen mit relativer Güte, d.h. Approximationsalgorithmen welche eine optimale Lösung immerhin noch bis auf einen bestimmten Faktor approximieren.

Definition (relative Güte; k-Approximationsalgorithmus)**.** Sei Π ein Optimierungsproblem und sei A ein Approximationsalgorithmus. A hat *relative Güte* k (≥ 1), falls für alle $I \in \Pi$ gilt:

$$\frac{1}{k} \cdot OPT(I) \leq A(I) \leq k \cdot OPT(I).$$

Ein Approximationsalgorithmus mit relativer Güte k heißt *k-Approximationsalgorithmus.*

Anmerkung. Für Minimierungsprobleme ist die linke Seite der Ungleichung für $k \geq 1$ immer erfüllt, daher genügt es

$$\frac{A(I)}{OPT(I)} \leq k$$

zu fordern. Analog ist für Maximierungsprobleme die rechte Seite der Ungleichung für $k \geq 1$ immer erfüllt, d.h. hier genügt es,

$$\frac{OPT(I)}{A(I)} \leq k$$

zu fordern.

6.3.1 Minimum-Vertex-Cover-Problem

Wir betrachten erneut den Algorithmus Approx_VC (siehe auch Alg. 1).

Algorithm 12 Approx_VC ($G = (V, E)$)

1: $C \leftarrow \emptyset$
2: $E' \leftarrow E$
3: **while** $E' \neq \emptyset$ **do**
4: Wähle beliebige Kante $(uw) \in E'$.
5: $C \leftarrow C \cup \{u, w\}$
6: Entferne alle Kanten aus E', die inzident zu u oder w sind.
7: **end while**
8: **return** C als VC

Theorem. *Es gilt:*

(1) Approx_ VC liefert ein vertex cover von G.

(2) Approx_ VC hat relative Güte 2, ist also ein 2-Approximationsalgorithmus.

Beweis. Siehe Abschnitt 2.3. □

Anmerkung. Es ist bisher kein Approximationsalgorithmus für das Vertex-Cover-Problem mit relativer Güte kleiner als 2 bekannt. Es konnte aber gezeigt werden:

Theorem (Dinur,Safran 2005). *Wenn $\mathcal{P} \neq \mathcal{NP}$, dann existiert kein Approximationsalgorithmus mit relativer Güte von 1,3606 für das Vertex-Cover-Problem.*

6.3.2 Das MAX-CUT-Problem

Definition (Schnitt (Cut) eines Graphen). Sei $G = (V, E)$ ein Graph und $c : E \to \mathbb{R}^+$ eine Gewichtsfunktion. Wir definieren eine Partition der Knoten in zwei Teilmengen, $C_V \subseteq V$ und $V \setminus C_V =: \overline{C_V}$, wobei C_V als *Cut* bezeichnet wird. Das Gewicht eines Cuts definieren wir als $f(C_V) := \sum_{e \in C_E} c(e)$, wobei $C_E = \{(u, v) \in E \mid u \in C_V, v \in \overline{C_V}\}$ die Menge aller Kanten ist, die einen Endpunkt in C_V und einen Endpunkt in $\overline{C_V}$ haben.

MAX-CUT-Problem
Gegeben: Ungerichteter Graph $G = (V, E)$, Kostenfunktion $c : E \to \mathbb{R}^+$
Ziel: Finde einen Cut C_V mit maximalem Gewicht.

Anmerkung. Wenn $G = (V, E)$ zusammenhängend ist und die nicht-leere Menge $C_V \neq V$ ein Cut von G ist, so ist der Graph $G' := (V, E \setminus C_E)$ nicht zusammenhängend.

Theorem. *Das Entscheidungsproblem zu MAX-CUT ist $\mathcal{NP}$-vollständig (ohne Beweis).*

Wir betrachten nun einen Approximationsalgorithmus für das MAX-CUT-Problem. Zur Vereinfachten Notation verwenden wir die Kostenfunktion $c : E \to \{1\}$. Der Algorithmus sowie folgende Beweise können aber leicht auf den Fall $c : E \to \mathbb{R}^+$ erweitert werden.

Algorithm 13 Approx_CUT $(G = (V = \{1, \ldots, n\}, E), c : E \to \{1\}))$

```
1:  C_V ← {1}
2:  C̄_V ← ∅
3:  for i = 2, ..., n do
4:      if |N(i) ∩ C_V| ≤ |N(i) ∩ C̄_V| then        ▷ N(i) = {v ∈ V | (v, i) ∈ E}
5:          C_V ← C_V ∪ {i}
6:      else
7:          C̄_V ← C̄_V ∪ {i}
8:      end if
9:  end for
10: return C_V, C̄_V
```

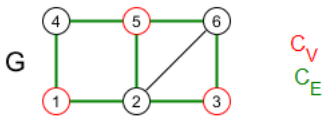

Abbildung 6.2: Beispiel zu Approx_CUT

Beispiel. Betrachte den Graphen $G = (V, E)$ aus Abbildung 6.2. Der Algorithmus Approx_CUT führt nun folgende Schritte aus (siehe Tabelle 6.1):

Tabelle 6.1: Durchlauf des Algorithmus Approx_CUT für $G = (V, E)$ aus Abbildung 6.2

i	$N(i)$	$N(i) \cap C_V$	$N(i) \cap \overline{C_V}$	C_V (alt)	C_V (neu)	$\overline{C_V}$ (alt)	$\overline{C_V}$(neu)
1	-	-	-	-	$\{1\}$	-	$\emptyset$
2	$\{1,3,5,6\}$	$\{1\}$	$\emptyset$	$\{1\}$	$\{1\}$	$\emptyset$	$\{2\}$
3	$\{2,6\}$	$\emptyset$	$\{2\}$	$\{1\}$	$\{1,3\}$	$\{2\}$	$\{2\}$
4	$\{1,5\}$	$\{1\}$	$\emptyset$	$\{1,3\}$	$\{1,3\}$	$\{2\}$	$\{2,4\}$
5	$\{2,4,6\}$	$\emptyset$	$\{2,4\}$	$\{1,3\}$	$\{1,3,5\}$	$\{2,4\}$	$\{2,4\}$
6	$\{2,3,5\}$	$\{3,5\}$	$\{2\}$	$\{1,3,5\}$	$\{1,3,5\}$	$\{2,4\}$	$\{2,4,6\}$

Theorem. *Approx_CUT*

(1) liefert eine zulässige Lösung $C_V \subseteq V$;

(2) hat eine relative Güte von zwei.

Beweis.

(1) Offensichtlich gilt $C_V \subseteq V$.

(2) Seien $C_i, \overline{C_i}$ die Mengen C_V bzw. $\overline{C_V}$, die der Algorithmus in Durchlauf i bestimmt. Sei G_i der durch $C_i \cup \overline{C_i}$ induzierte Teilgraph und $E(C_i, \overline{C_i}) = \{(x, y) \in E \mid x \in C_i, y \in \overline{C_i}\}$. Wir zeigen zunächst induktiv folgende Abschätzung:

$$|E(C_i, \overline{C_i})| \geq \frac{1}{2}|E(G_i)|$$

Induktionsanfang $i = 1$: $C_i = \{1\}$, $\overline{C_i} = \emptyset$, also

$$|E(C_i, \overline{C_i})| = 0 \geq \frac{1}{2}|E(G_i) = 0$$

Induktionsvoraussetzung: Die Annahme gelte für $i < n$

Induktionsschritt: Wir unterscheiden die zwei Fälle der IF-Abfrage im Alg. Approx_CUT.

- Fall 1: $|N(i+1) \cap C_i| \leq |N(i+1) \cap \overline{C_i}|$
  Dann können wir folgende Abschätzung aus der Mengentheorie verwenden: Wenn A und B zwei Mengen sind mit $|A| \leq |B|$, so gilt $|B| \geq \frac{1}{2}|A \cup B|$ (ÜA)

Also gilt:

$$\begin{aligned}|N(i+1) \cap \overline{C_i}| &\geq \frac{1}{2}|(N(i+1) \cap C_i) \cup (N(i+1) \cap \overline{C_i})| \\ &= \frac{1}{2}|N(i+1) \cap (C_i \cup \overline{C_i})| \ (*)\end{aligned}$$

Da dies den ersten Fall der IF-Abfrage darstellt, gilt

$$\begin{aligned}C_{i+1} &= C_i \cup \{i+1\} \\ \overline{C_{i+1}} &= \overline{C_i}\end{aligned}$$

Somit erhalten wir

$$\begin{aligned}|E(C_{i+1}, \overline{C_{i+1}})| &= \underbrace{|E(C_i, \overline{C_i})|}_{\geq \frac{1}{2}|E(G_i)|} + \underbrace{|N(i+1) \cap \overline{C_i}|}_{\geq \frac{1}{2}|N(i+1) \cap (C_i \cup \overline{C_i})| \ (*)} \\ &\geq \frac{1}{2}|E(G_i) + \frac{1}{2}|N(i+1) \cap (C_i \cup \overline{C_i})| \\ &= \frac{1}{2}|E(G_{i+1})|\end{aligned}$$

- Fall 2: $|N(i+1) \cap C_i| > |N(i+1) \cap \overline{C_i}|$
  analog

Also gilt für $i = n$:

$$|E(C_V, \overline{C_V})| \geq \frac{1}{2}E(G)$$

Damit folgt

$$\frac{OPT(G)}{Approx\_CUT(G)} = \frac{OPT(G)}{|E(C_V, \overline{C_V})|} \leq \frac{|E(G)|}{|E(C_V, \overline{C_V})|} \leq 2.$$

□

Anmerkungen.

- Für das MAX-CUT-Problem sind Approximationsalgorithmen mit relativer Güte von 1,139 bekannt.
-

  Theorem (Håstad). *Wenn $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen Approximationsalgorithmus mit relativer Güte von $\frac{17}{16} = 1,062$ für das MAX-CUT-Problem.*

6.4 Nichtapproximierbarkeit mit relativer Güte

Wir betrachten erneut das Problem des Handlungsreisenden (TSP; siehe Probl. 0.2), d.h. gegeben seien ein vollständiger Graph K_n und eine Kostenfunktion $c : E \to \mathbb{R}^+$. Gesucht ist

ein kürzester Rundweg, sodass alle n Knoten (Städte) genau einmal besucht werden.

Theorem. *Wenn $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen Polynomialzeit-Approximationsalgorithmus mit relativer Güte für das TSP.*

Beweis. Wir nehmen an, dass es einen Polynomialzeit-Approximationsalgorithmus mit relativer Güte k für das TSP gibt. Nun betrachten wir eine Instanz I des Problems *Hamiltonischer Kreis* (also ungerichteter Graph $G = (V, E)$) und konstruieren daraus eine Instanz I' für das TSP wie folgt:
Ungerichteter (vollständiger) Graph $H = (V, \binom{V}{2})$,
Kostenfunktion $c : E(H) \to \mathbb{R}^+$ mit $c(i,j) \begin{cases} 1, & (i,j) \in E(G) \\ k \cdot |V|, & (i,j) \notin E(G) \end{cases}$
Nun zeigen wir: G enthält einen hamiltonischen Kreis $\Leftrightarrow A(I) \leq k \cdot |V|$.

„$\Rightarrow$“: Wenn G einen hamiltonischen Kreis enthält, so ist dieser Kreis eine optimale Rundtour in H mit Gewichtsfunktion c, d.h. $OPT(I) = |V|$. Da A relative Güte k hat, folgt

$$A(I) \leq k \cdot OPT(I) = k \cdot |V|.$$

„$\Leftarrow$“: Angenommen G besitzt keinen hamiltonischen Kreis. Dann enthält jede Rundtour in H mit Gewichtsfunktion c eine Kante, die nicht in G ist, also eine Kante mit Gewicht $k \cdot |V|$. Daraus folgt

$$A(I) \geq (|V| - 1) + k \cdot |V| > k \cdot |V|$$

Da Algorithmus A also das $\mathcal{NP}$-vollständige Problem *Hamiltonischer Kreis* in polynomieller Zeit löst, folgt $\mathcal{P} = \mathcal{NP}$. ↯ □

Anmerkung. Für den Spezialfall, dass die Gewichtsfunktion c die Dreiecksungleichung erfüllt ($\triangle$-TSP), gibt es Approximationsalgorithmen mit einer relativen Güte von $1,5$, aber keine mit absoluter Güte.
Schränkt man das $\triangle$-TSP weiter ein, sodass die Knoten aus V in den $\mathbb{R}^2$ eingebettet sind und c die euklidische Distanz zwischen Knoten ist, erhält man das sogenannte *Euklid-TSP*, welches mit zunehmender Laufzeit des Algorithmus beliebig gut approximiert werden kann.

Verallgemeinerungen Seien nun ein (Minimierungs)problem Π und eine Konstante $k \in \mathbb{N}^+$ gegeben. Dann können wir das folgende Entscheidungsproblem $k - \Pi$ betrachten:

k − Π
Gegeben: Instanz I von Π
Frage: Ist $OPT(I) \leq k$?

Beachte, dass k dabei eine Konstante und *nicht* Teil der Eingabe des Problems Π ist.
Der folgende Satz gibt den Zusammenhang zwischen der Zeitkomplexität von $k - \Pi$ und der Approximierbarkeit des zugehörigen Optimierungsproblems Π an.

Theorem. *Sei Π ein Minimierungsproblem. Falls $\mathcal{P} \neq \mathcal{NP}$ und falls für eine Konstante $k \in \mathbb{N}^+$ das Entscheidungsproblem $k - \Pi$ $\mathcal{NP}$-vollständig ist, so existiert kein Polynomialzeit-Approximationsalgorithmus für Π mit relativer Güte $< 1 + \frac{1}{k}$.*

Beweis. Wir nehmen an, dass es einen Polynomialzeit-Approximationsalgorithmus A für Π mit relativer Güte $l < 1 + \frac{1}{k}$ gibt. Dann kann man das Entscheidungsproblem $k - \Pi$ mit A wie folgt lösen:

$$A(I) \leq k \Leftrightarrow OPT(I) \leq k,$$

da:

„$\Rightarrow$“: Da $A(I) \leq k$ und Π Minimierungsproblem, folgt direkt $OPT(I) \leq A(I) \leq k$.

„$\Leftarrow$“: Angenommen $A(I) > k$. Dann folgt $k+1 \leq A(I) \leq l \cdot OPT(I) < (1+\frac{1}{k})OPT(I)$, d.h. $OPT(I) > k$.

Das $\mathcal{NP}$-vollständige Entscheidungsproblem kann also mit A in Polynomialzeit gelöst werden, d.h. $\mathcal{P} = \mathcal{NP}$. $\lightning$ □

Beispiel. Das Entscheidungsproblem *k-Min-V-col* ist $\mathcal{NP}$-vollständig für $k = 3$ (siehe 6.1.1). Also gilt für *Min-V-col* unter der Annahme $\mathcal{P} \neq \mathcal{NP}$, dass es keine Approximationsalgorithmen mit relativer Güte $< \frac{4}{3}$ gibt.

Anmerkung. Analog kann man letztere Resultate auch für Maximierungsprobleme zeigen.

6.5 Approximationsschemata

Approximationsalgorithmen mit absoluter oder relativer Güte liefern Lösungen, deren Abweichung von der optimalen Lösung gewissermaßen „fest“ ist.

Im Folgenden wollen wir, dass die Genauigkeit mit wachsender Laufzeit steigt, d.h. die relative Güte soll gegen eins konvergieren, wenn die Laufzeit gegen unendlich geht. Dazu betrachten wir sogenannte Approximationsschemata.

Definition (Polynomielles Approximationsschema)**.** Sei Π ein Optimierungsproblem. Ein Algorithmus A, der als Eingabe eine Instanz $I \in \Pi$ und eine Fehlerschranke $\epsilon > 0$ erhält, ist ein *polynomielles Approximationsschema* für Π, wenn A für alle ϵ (fest) ein $|I|^{\mathcal{O}(1)}$-Zeit Algorithmus mit relativer Güte $1 + \epsilon$ ist. A wird dann auch als *PAS/PTAS* (polynomial-time approximation scheme) bezeichnet.

Anmerkung. Die Laufzeit kann für verschiedene ϵ variieren. Insbesondere kann sie mit kleiner werdendem ϵ sehr schnell wachsen, z.B. falls A eine Laufzeit von $\mathcal{O}(|I|^{\frac{1}{\epsilon}})$ hat. Um dies zu vermeiden, sucht man auch nach sogenannten vollen polynomiellen Approximationsschemata.

Definition (Volles polynomielles Approximationsschema)**.** Ein polynomielles Approximationsschema heißt *volles polynomielles Approximationsschema*, falls seine Laufzeit sowohl polynomiell in $|I|$ als auch in $\frac{1}{\epsilon}$ ist. Volle polynomielles Approximationsschemata werden auch als *FPAS/FTPAS*(fully polynomial-time approximation scheme) bezeichnet.

Tabelle 6.2: Beispiele für PAS und FPAS

Laufzeit	PAS	FPAS
$\mathcal{O}(n^{\frac{1}{\epsilon}})$	$\surd$	-
$\mathcal{O}\left(\left(\frac{1}{\epsilon}\right)^2 n^5\right)$	$\surd$	$\surd$
$\mathcal{O}\left(\left(\frac{1}{\epsilon}^n\right)\right)$	-	-
$\mathcal{O}(4^{\frac{1}{\epsilon}} n^2)$	$\surd$	-

Approximationsschemata sind oft schwieriger zu formulieren als Approximationsalgorithmen.

Wir hatten bereits gezeigt, dass es unter der Annahme $\mathcal{P} \neq \mathcal{NP}$, keinen Approximationsalgorithmus mit relativer Güte $< \frac{4}{3}$ für das Problem Min-V-col gibt. Daraus folgt, dass auch kein PAS oder FPAS für Min-V-col existiert.

Allgemeiner haben wir gezeigt, dass für $\mathcal{P} \neq \mathcal{NP}$ und $k - \Pi$ $\mathcal{NP}$-vollständig, kein Approximationsalgorithmus mit relativer Güte $< 1 + \frac{1}{k}$ für Π existiert. Das bedeutet, dass auch kein PAS oder FPAS für Π existiert.

Weitere Komplexitätsklassen:

- APX: Optimierungsprobleme, die mit konstanter relativer Güte approximiert werden können
- ABS: Optimierungsprobleme, die mit absoluter relativer Güte approximiert werden können
- PAS: Optimierungsprobleme, für die ein polynomielles Approximationsschema existiert
- FPAS: Optimierungsprobleme, für die ein volles polynomielles Approximationsschema existiert
- PO: Optimierungsprobleme, die optimal in Polynomialzeit lösbar sind
- NPO: Optimierungsprobleme Π, sodass
  (1) Die Frage, ob eine Instanz $I \in \Pi$, kann in Polynomialzeit entschieden werden.
  (2) $\exists$ Polynom p, sodass für alle $I \in \Pi$:
    (a) Für alle zulässigen Lösungen σ gilt: $|\sigma| \leq p(|I|)$
    (b) Für alle t mit $|t| \leq p(|I|)$ kann in Polynomialzeit entschieden werden, ob t eine zulässige Lösung ist.

  Dabei gelten folgende Beziehungen:

$$\text{PO} \subseteq \text{FPAS} \subseteq \text{PAS} \subseteq \text{APX} \subseteq \text{NPO}$$

$$\text{PO} \subseteq \text{ABS} \subseteq \text{ABX}$$

6.5.1 Max-Simple-Knapsack

Wir betrachten nun das sogenannte *Max-Simple-Knapsack Problem*, das eine Vereinfachung des ursprünglichen Knapsack Problems darstellt.

Max-Simple-Knapsack
Gegeben: Gegenstände $W = \{1, \ldots, n\}$ mit Gewicht $w_i \in \mathbb{N}$ für $i = 1, \ldots, n$; Kapazität C
Ziel: Finde eine Teilmenge $M \subseteq W$ mit $f_I(M) = \sum_{i \in M} w_i \xrightarrow{!} \max$ und $f_I(M) \leq C$.

Approximation mit relativer Güte

Wir betrachten zunächst einen GREEDY-Algorithmus für das Max-Simple-Knapsack Problem.

Algorithm 14 GREEDY_SK $(W, \{w_i\}, C)$

```
1: Sortiere die Gewichte so, dass oBdA w_1 ≥ w_2 ≥ ... ≥ w_n
2: M ← ∅
3: cost ← 0
4: for i = 1, ..., n do
5:     if cost + w_i ≤ C then
6:         M ← M ∪ {i}
7:         cost ← cost + w_i
8:     end if
9: end for
10: return M, cost
```

Theorem. *GREEDY_SK*

(1) liefert eine zulässige Lösung;

(2) hat relative Güte 2.

Beweis.

(1) klar

(2) Sei $I = (W, \{w_i\}, C)$ eine Instanz von Max-Simple-Knapsack,sodass die Gewichte oBdA wie folgt sortiert sind: $C \geq w_1 \geq w_2 \geq \ldots \geq w_n$. Sei $M \subseteq W$ die von GREEDY_SK gefundene zulässige Lösung für I. Sei weiter $j+1$ der kleinste Index, sodass $j+1 \notin M$, d.h. $1, \ldots, j \in M$. Da $w_i \leq C$ für alle i, ist $M \neq \emptyset$, also insbesondere $j > 1$ (der Gegenstand 1 ist also für alle Instanzen $I \in \Pi$ in M). Nun unterscheiden wir zwei Fälle:

Tabelle 6.3: Durchlauf von GREEDY_SK für $W = \{1, \ldots, 7\}, C = 25, \{w_1, \ldots, w_7\} = \{14, 14, 13, 12, 6, 6, 1\}$:

i	M	$cost$
1	$\{1\}$	14
2	$\{1\}$	14
3	$\{1\}$	14
4	$\{1\}$	14
5	$\{1, 5\}$	20
6	$\{1, 5\}$	20
7	$\{1, 5, 7\}$	21

- $j = 1$:
  2 ist der erste Gegenstand, der nicht in M liegt, also muss gelten $w_1 + w_2 > C$. Da $w_1 \geq w_2$, folgt $A(I) \geq w_1 > \frac{C}{2}$, also

$$\frac{OPT(I)}{A(I)} \leq \frac{C}{\frac{C}{2}} = 2$$

- $j \geq 2$:
  Dann gilt $A(I) + w_{j+1} > C \geq OPT(I)$. Da außerdem $w_1 \geq w_2 \geq \ldots \geq w_n$ gilt, folgt

$$w_{j+1} \leq w_j = \frac{jw_j}{j} \leq \frac{w_1 + w_2 + \ldots + w_j}{j} \leq \frac{C}{j} \tag{6.1}$$

  Also, $A(I) > C - w_{j+1} \overset{6.1}{\geq} C - \frac{C}{j} \geq \frac{C}{2}$. Damit folgt

$$\frac{OPT(I)}{A(I)} \leq \frac{C}{\frac{C}{2}} = 2$$

□

PAS für Max-Simple-Knapsack

oBdA seinen im Folgenden die Gewichte so gewählt, dass für alle Instanzen $I \in \Pi$ gilt:

$$C \geq w_1 \geq w_2 \geq \ldots \geq w_n.$$

Sei $M_{Opt} = \{i_1, \ldots, i_r\}$ mit $w_{i_1} \geq \ldots w_{i_r}$ eine optimale Lösung für $I \in \Pi$. Sei weiterhin M eine beliebige zulässige Lösung von I, welche (neben anderen Elementen) die ersten j Elemente

$w_{i_1}, \ldots, w_{i_j}$ aus M_{Opt} enthält mit

$$f_I(M) + w_{i_{j+1}} = \sum_{k=1}^{j} w_{i_k} + \sum_{l \in M \setminus \{i_1,\ldots,i_j\}} w_l + w_{i_{j+1}} > C$$

Dann gilt für die Differenz von $f(M_{Opt})$ und $f(M)$:

$$f(M_{Opt}) - f(M) \leq C - f(M) < w_{i_{j+1}} \overset{6.1}{\leq} \frac{C}{i_{j+1} - 1} \leq \frac{C}{j}.$$

Um ein solches M zu erhalten, genügt es, alle Teilmengen $M \subseteq \{1, \ldots, n\}$ mit Kardinalität $\leq j$ zu betrachten und diese dann mittels GREEDY_SK zu erweitern. Klar, mit zunehmenden j verbessert sich die Lösung, während die Laufzeit schlechter wird. Dies führt uns auf das Approximationsschema SK_Schema für das Max-Simple-Knapsack Problem.

Algorithm 15 SK_Schema $(W = \{1, \ldots, n\}, \{w_i\}, C, \epsilon)$

1: Sortiere die Gewichte so, dass oBdA $C \geq w_1 \geq w_2 \geq \ldots \geq w_n$
2: $K_\epsilon \leftarrow \lceil \frac{1}{\epsilon} \rceil$
3: Für alle $M \subseteq \{1, \ldots, n\}$ mit $|M| \leq K_\epsilon$ und $\sum_{i \in M} w_i \leq C$, erweitere M mittels GREEDY_SK zu M^*.
4: **return** M^* mit maximalem Wert f_I

Theorem. *SK_Schema ist ein polynomielles Approximationsschema für Max-Simple-Knapsack mit Laufzeit $\mathcal{O}(n^{\lceil \frac{1}{\epsilon} \rceil + 1})$.*

Beweis.

- Laufzeit:
  Wir nehmen an, dass die Gewichte bereits sortiert sind (sonst Laufzeit für das Sortieren: $\mathcal{O}(n \log n)$). Die Bestimmung der Teilmengen M mit $|M| \leq K_\epsilon$ aus Schritt 3 liegt in $\mathcal{O}(n^{K_\epsilon})$, da:

$$\begin{aligned}
\text{Anzahl an Teilmengen } M \text{ mit } |M| \leq K_\epsilon &= \sum_{0 \leq i \leq K_\epsilon} \binom{n}{i} \\
&= \sum_{0 \leq i \leq K_\epsilon} \frac{n!}{(n-i)!i!} \\
&\leq \sum_{0 \leq i \leq K_\epsilon} (n-i+1)(n-i+2)\ldots n \\
&\leq \sum_{0 \leq i \leq K_\epsilon} n^i \in \mathcal{O}(n^{K_\epsilon})
\end{aligned}$$

  Wenn die Gewichte schon sortiert sind, hat GREEDY_SK eine Laufzeit von $\mathcal{O}(n)$, d.h. wir erhalten eine Gesamtlaufzeit von $\mathcal{O}(n^{K_\epsilon} n) = \mathcal{O}(n^{K_\epsilon + 1}) = \mathcal{O}(n^{\lceil \frac{1}{\epsilon} \rceil + 1})$.

- Relative Güte:
  Sei I eine Instanz mit Items $\{1, \ldots, n\}$ und Gewichten $C \geq w_1 \geq \ldots \geq w_n$. Weiter sei

$M_{Opt} = \{i_1, \ldots, i_p\} \subseteq \{1, \ldots, n\}$ mit $w_{i_1} \geq w_{i_2} \geq \ldots \geq w_{i_p}$ eine optimale Lösung für I, d.h. $f(M_{Opt}) = OPT(I)$. Nun unterscheiden wir zwei Fälle:

(i) $p \leq K_\epsilon$:
M_{Opt} wird im 3. Schritt von SK_Schema betrachtet/gefunden (und wird mittels GREEDY_SK nicht mehr verändert), d.h. $A(I) = OPT(I)$.

(ii) $p > K_\epsilon$:
$M = \{i_1, \ldots, i_{K_\epsilon}\}$ ist eine der Teilmengen, die in Schritt 3 von SK_Schema betrachtet werden. Diese wird dann mittels GREEDY_SK zu einer Menge M^* erweitert. Nun unterscheiden wir erneut zwei Fälle:

- $M^* = M_{Opt}$: $\Rightarrow A(I) = OPT(I)$.
- $M^* \neq M_{Opt}$:
  M_{Opt} sieht folgendermaßen aus: $M_{Opt} = \{i_1, \ldots, i_{K_\epsilon}, i_{K_\epsilon+1}, \ldots, i_p\}$, wobei $i_1 < i_2 < \ldots < i_p$. Insbesondere existiert ein Element $i_q \in M_{Opt} \setminus M^*$, sodass $i_q > i_{K_\epsilon} \geq K_\epsilon$.
  Es gilt:
  $$f(M^*) + w_{i_q} > C \geq f(M_{Opt}) \tag{6.2}$$
  Da aber $i_1, \ldots, i_{K_\epsilon}$ und $i_q \in M_{Opt}$, gilt:
  $$w_{i_q} = \frac{(K_\epsilon + 1)w_{i_q}}{K_\epsilon + 1} \leq \frac{w_{i_1} + w_{i_2} + \ldots + w_{i_q}}{K_\epsilon + 1} \leq \frac{f(M_{Opt})}{K_\epsilon + 1} \tag{6.3}$$
  Daraus folgt:
  $$\frac{OPT(I)}{A(I)} = \frac{f(M_{Opt})}{f(M^*)} \overset{6.2}{<} \frac{f(M_{Opt})}{f(M_{Opt}) - w_{i_q}} \overset{6.3}{\leq} \frac{f(M_{Opt})}{f(M_{Opt}) - \frac{f(M_{Opt})}{K_\epsilon+1}} = 1 + \frac{1}{K_\epsilon} \leq 1 + \epsilon.$$

□

Anmerkungen.

- Für das "Standard" Max-Knapsack Problem gibt es PAS-Algorithmen mit Laufzeit $O(n^{\frac{1}{\epsilon}+2})$ und sogar FPAS-Algorithmen mit Laufzeit $\mathcal{O}(n^3 \frac{1}{\epsilon})$.
- Aus letzterem Resultat können wir folgern, dass die Existenz eines FPAS nicht bedeutet, dass es keinen Approximationalgorithmus mit absoluter Güte gibt.

7 Fest-Parameter-berechenbare Algorithmen (fixed-parameter-tractable; FPT)

Die Untersuchung bzw. das Design von FPT-Algorithmen ist ein relativ junges Teilgebiet der Informatik und wird seit den 1990er Jahren zur Handhabung $\mathcal{NP}$-schwerer Probleme eingesetzt.

Hauptidee: Isolierung von Parametern als Teil des Problems, auf denen das exponentielle Verhalten beschränkt werden kann $\Rightarrow$ für „kleine“ Parameter kann man das Problem „effizient“ lösen.

Definition (parametrisiertes Problem)**.** Ein *parametrisiertes Problem* ist ein Paar (Π, k), sodass Π ein Entscheidungsproblem mit Instanzmenge I_Π ist und $k : I_\Pi \to \mathbb{N}, I \in I_\Pi \mapsto k(I)$ eine (in Polynomialzeit) berechenbare Funktion (die sogenannte Parametrisierung). Ein parametrisiertes Problem wird auch als *para*-Π bezeichnet.

Beispiel (para-VC)**.**

Eingabe: Graph $G = (V, E)$, Parameter $k \in \mathbb{N}$

Frage: Existiert ein vertex cover $V' \subseteq V$ von G, sodass $|V| \leq k$?

Allgemein stellt sich die Frage, welche para-Π Probleme „effizient“ für „kleine“ k gelöst werden können.

Definition (FPT-Algorithmus, fixed-parameter-tractable, Klasse FPT)**.** Sei (Π, k) ein parametrisiertes Problem, I_Π die Instanzmenge von Π und $k : I_\Pi \to \mathbb{N}$ die Parametrisierung.

(1) Ein Algorithmus A heißt *FPT-Algorithmus* für (Π, k), wenn A bei Eingabe von $I \in I_\Pi$ in Laufzeit $f(k)p(|I|) \leq f(k)|I|^c$ entscheiden kann, ob I JA-Instanz ist oder nicht ($\forall I \in I_\Pi$), wobei $f(k)$ eine beliebige berechenbare Funktion (unabhängig von $|I|$) und c eine Konstante ist.

(2) (Π, k) heißt *fixed-parameter-tractable*, wenn es einen FPT-Algorithmus für (Π, k) gibt.

(3) Die *Klasse FPT* ist die Menge aller parametrisierten Probleme (Π, k), die fixed-parameter-tractable sind.

Lemma. $(\Pi, k) \in FPT \Leftrightarrow$ *es existieren eine berechenbare Funktion* $f : \mathbb{N} \to \mathbb{R}$*, ein Polynom* p *und ein Algorithmus* A*, der* Π *für eine beliebige Instanz* I *in* $\mathcal{O}(f(k(I)) + p(|I|))$ *löst (ohne Beweis).*

Anmerkung. Wenn ein Optimierungsproblem ein FPAS besitzt, so ist die parametrisierte Version des Problems in FPT (ÜA).

Grundtechniken: Neben anderen, gibt es zwei wichtige Grundtechniken um festzustellen, ob ein Problem FPT ist, bzw. zur Entwicklung von FPT-Algorithmen:

(1) Problemkernreduzierung:
Dabei wird eine Instanz durch Anwendung verschiedener Regeln auf einen (schweren) *Problemkern* reduziert.

(2) Tiefenbeschränkte Suchbäume:
Dabei wird eine „erschöpfende" Suche (d.h. volle Enumeration aller zulässigen Lösungen) in einem geeigneten Suchbaum mit beschränkter Tiefe durchgeführt.

Schritt (2) wird oft nach Schritt (1) ausgeführt, also nachdem das Problem auf ein „kleineres" Problem reduziert wurde.

7.1 Reduzierung auf den Problemkern

Idee: Gegeben sei eine Instanz $I \in I_\Pi$ für ein parametrisiertes Problem (Π, k). Reduziere I in Polynomialzeit auf eine äquivalente Instanz I', sodass die Größe von I' allein durch eine Funktion in Abhängigkeit vom Parameter k beschränkt werden kann.

Definition (Problemkern-Reduktion)**.** Sei (Π, k) ein parametrisiertes Problem mit Instanzmenge I_Π und Parametrisierung $k : I_\Pi \to \mathbb{N}$. Eine in Polynomialzeit berechenbare Funktion $f : I_\Pi \times \mathbb{N} \to I_\Pi \times \mathbb{N}$ heißt *Problemkern-Reduktion für* (Π, k) *(kernelization)*, wenn f bei Eingabe $(I, k(I))$ eine Ausgabe $(I', k(I'))$ berechnet, sodass folgende drei Eigenschaften erfüllt sind:

(1) Für $I \in I_\Pi$ ist $(I, k(I))$ eine JA-Instanz von $\Pi \Leftrightarrow (I', k(I'))$ ist JA-Instanz von Π.

(2) Es existiert eine Funktion $f' : \mathbb{N} \to \mathbb{N}$, sodass $|I'| \leq f'(k(|I|))$

(3) Es gilt $k(I') \leq k(I)$.

$(I', k(I'))$ heißt dann *(Problem)Kern* von (Π, k) und $f'(k(I))$ heißt *Größe des Kerns.*

7.1.1 Beispiel: Vertex Cover

Lemma 7.1. *Sei* $G = (V, E)$ *ein Graph und* $v \in V$ *ein Knoten mit* $deg(v) = 0$*. Dann gilt:* G *hat VC der Größe* $k \Leftrightarrow G - v$ *hat VC der Größe* k*.*

Lemma 7.2 (Problemkernreduktion von Buss)**.** *Sei (Π, k) das parametrisierte VC (para-VC) und $(G = (V, E), k)$ eine Instanz. Weiterhin sei $v \in V$ ein Knoten mit $deg(v) > k$. Dann gilt: G hat VC der Größe $k \Leftrightarrow G - v$ hat VC der Größe $k - 1$ (Beweis ÜA).*

Lemma 7.3. *Sei $G = (V, E)$ ein Graph (ohne Knoten mit Grad 0) und sei $C \subseteq V$ ein vertex cover, sodass $|C| \leq k$. Für den Maximalgrad von G gelte $\Delta(G) \leq d$. Dann hat G höchstens dk Kanten und $k(d+1)$ Knoten.*

Beweis.

- $|E| \leq k\Delta(G) \leq kd$
- $|V| \leq |C| + dk \leq k + dk \leq k(d+1)$

□

Lemma 7.4. *Das Problem para-VC hat einen Problemkern der Größe $\leq k(k+1)$ für einen Parameter k.*

Beweis. Sei (G, k) eine Instanz von para-VC, wobei $G = (V, E)$ ein Graph und $k \in \mathbb{N}$ ein Parameter sind. oBdA nehmen wir an, dass in G kein Knoten $v \in V$ mit $deg(v) = 0$ oder $deg(v) > k$ existiert (ansonsten modifizieren wir G gemäß Lemma 7.1 und Lemma 7.2). Nun unterscheiden wir zwei Fälle:

- $|V| > k(k+1)$: Mit Lemma 7.3 folgt, dass kein VC der Größe $\leq k$ in G existiert.
- $|V| \leq k(k+1)$: Problemkern der Größe $k(k+1)$ gefunden.

□

Anmerkung. In anderen Worten bedeutet die letzten Resultate, dass es genügt, für einen gegeben Graphen $G = (V, E)$ und einen Parameter k, den Graphen G zuerst auf seinen Problemkern G' der Größe $\leq k(k+1)$ mittels Lemma 7.1 und 7.2 zu reduzieren. Dabei merken wir uns die Knoten, die in jedem VC von G enthalten (Lemma 7.2) sein müssen. Um nun das gesamte VC von G der Kardinalität $\leq k$ zu finden, genügt es dann in $G' = (V', E')$ z.B. mittels vollständiger Enummerierung der Lösungen ein entsprechendes VC der Kardinalität $k' \leq k$ von G' zu finden (k' ergibt sich aus dem Reduktionsschritt Lemma 7.2). Wichtig, es gilt $|V'| \leq k(k+1)$.

Beispiel. Betrachte den Graph $G = (V, E)$ aus Abb. 7.1 und $k = 4$. Dabei soll gelten $deg(1) = deg(2) = deg(3) = L$, $L > 4$. (angedeutet durch die gestrichelten Linien). Ein naiver Ansatz wäre, für alle 4-elementigen Teilmengen von V zu testen, ob sie ein VC bilden. Dies führt jedoch auf $\binom{|V|}{4}$ Fälle, welches für mittel-große V nicht mehr effizient ist. Daher reduzieren wir das Problem auf seinen Problemkern:

- Wegen Lemma 7.2 muss $C' := \{1, 2, 3\}$ in jedem VC von G enthalten sein.
  Wir löschen diese Knoten aus G und erhalten einen Graph $G - C'$. Nun gilt: G hat VC der Größe $k \Leftrightarrow G - C'$ hat VC der Größe $k - |C'|$.

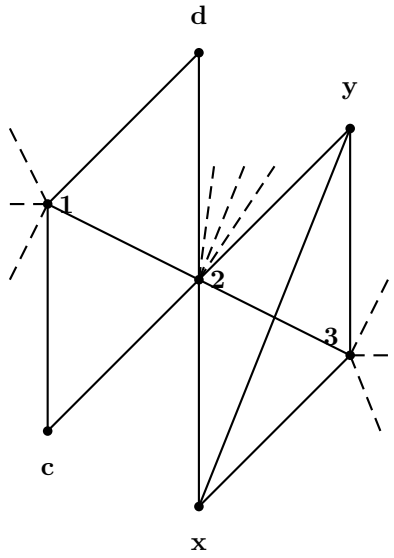

Abbildung 7.1: Problemkernreduktion VC

- Wegen Lemma 7.1, müssen isolierte Knoten in $G - C'$ nicht betrachtet werden. Nach Löschen dieser Knoten erhalten wir einen Graphen G'', der nur noch aus der Kante (x, y) besteht. Da G'' offensichtlich ein VC der Größe 1 hat, folgt, dass G ein VC der Größe 4 hat.

Anmerkung. Es gibt auch „unschöne“ Graphen, für die eine Problemkernreduktion nichts bringt. Ein Beispiel hierfür sind reguläre Graphen (d.h. alle Knoten haben den gleichen Grad).

Theorem 7.5. *FPT_ VC2 entscheidet für jeden Graphen $G = (V, E)$ und Parameter k in Zeit $\mathcal{O}(k|V| + 2^k k^2)$, ob G ein vertex cover der Größe $\leq k$ besitzt.*

Beweis.

- Korrektheit: folgt aus Lemma 7.1, Lemma 7.2 und Lemma 7.3.
- Laufzeit:
  Die Schritte in Zeilen 1-10 gehen bei geigneter in Datenstrucktur in $\mathcal{O}(|V|k)$ Zeit .
  G' hat dann höchstens $k'(k+1)$ Knoten/Kanten, was auf eine Laufzeit von $\mathcal{O}(k^2)$ führt.
  Die Suchbaumhöhe beträgt $2^{k'}$, wobei $k' \leq k$, d.h. wir erhalten $\mathcal{O}(2^k)$.
  Insgesamt ergibt sich also eine Laufzeit von $\mathcal{O}(|V|k + k^2 2^k)$.

□

Besser als ein quadratischer Problemkern wäre ein linearer. Zu diesem Zweck betrachten wir jetzt das folgende Theorem.

Theorem 7.6 (Nemhauser-Trotter (NT-Theorem))**.** *Für einen Graphen $G = (V, E)$ mit $|V| = n$ und $|E| = m$ können zwei disjunkte Mengen $C_0, V_0 \subseteq V$ in $\mathcal{O}(\sqrt{n}m)$ berechnet werden, sodass folgende Eigenschaften gelten:*

Algorithm 16 FPT_VC2 ($G = (V, E)$, Parameter $k \in \mathbb{N}$)

```
1:  C' ← {v ∈ V | deg(v) > k in G}                                  ▷ Lemma 7.2
2:  if |C'| > k then
3:      return False
4:  else
5:      k' ← k − |C|
6:      G' ← G − C'
7:      Lösche alle isolierten Knoten aus G', d.h. G' ← G' \ { isolierte Knoten}  ▷ Lemma 7.1
8:      if |V(G')| > k'(k + 1) then
9:          return False                                            ▷ Lemma 7.3
10:     else
11:         Finde VC mittels Suchbaum auf G' mit Parameter k'
12:         if VC existiert then
13:             return True
14:         else
15:             return False
16:         end if
17:     end if
18: end if
```

(a) Wenn $D \subseteq V_0$ ein vertex cover von $\langle V_0 \rangle$ ist, dann ist $C = D \cup V_0$ ein vertex cover von G.

(b) Es existiert ein minimum vertex cover C^* von G, sodass $C_0 \subseteq C^*$.

(c) $\langle V_0 \rangle$ besitzt ein minimum vertex cover der Größe $\geq \frac{|V_0|}{2}$.

Anmerkung. Theorem 7.6 liefert eine Menge $C_0 \subseteq V$, welche Teilemenge eines optimalen VC von G ist. Die Eigenschaft (c) besagt, dass der „verbleibende Restgraph“ $\langle V_0 \rangle$ von G höchstens doppelt so viele Knoten enthält wie sein optimales VC groß ist. Da das optimale VC in para-VC durch k beschränkt ist und das NT-Theorem die Menge V_0 liefert, erhalten wir einen Problemkern mit maximal $2k$ Knoten (das VC von $\langle V_0 \rangle$ muss man dann noch finden). Wir erhalten damit folgendes Proposition:

Proposition. *Das Problem para-VC kann (mit vorausgehender Buss-Reduktion) in Zeit $\mathcal{O}(\sqrt{n}m + 2^k k)$ gelöst werden.*

Beweis des NT-Theorems. Sei $G = (V, E)$ ein Graph. Für eine gegebene Menge X sei desweiteren die Menge $X' \coloneqq \{x' \mid x \in X\}$ die Kopie von X. Wir betrachten folgenden Algorithmus, der G als Eingabe erhält:

1: Konstruiere einen bipartiten Graphen $B = (V \cup V', E_B)$ mit $E_B = \{(x'y), (xy') \mid (xy) \in E\}$.
2: Berechne ein optimales VC C_B von B (z.B. mittels maximum-matching-Algorithmus in $\mathcal{O}(\sqrt{n}m)$).
3: Setze $C_0 \coloneqq \{x \mid x \in C_B \text{ und } x' \in C_B\}$ und $V_0 \coloneqq \{x \mid x \in C_B \text{ oder } x' \in C_B\}$.

Per Konstruktion gilt $C_B = V_0 \dot\cup C_0 \dot\cup C_0'$. Wir zeigen nun, dass die Eigenschaften (a)-(c) für

C_0 und V_0 erfüllt sind. Dazu definieren wir zusätzlich die Menge $I_0 := \{x \mid x \notin C_B \text{ und } x' \notin C_B\} = V \setminus (V_0 \cup C_0)$.

(a) Sei $(x, y) \in E$. Zu zeigen ist, dass $x \in C$ oder $y \in C$ mit $C = C_0 \cup D$, wobei D ein VC von $\langle V_0 \rangle$ ist. Dazu unterscheiden wir vier Fälle:

(i) $x \in I_0 \Rightarrow x, x' \notin C_B \overset{C_B \text{ VC von } B}{\Rightarrow} y, y' \in C_B \Rightarrow y \in C_0 \Rightarrow y \in C = C_0 \cup D$.

(ii) $y \in I_0$ analog.

(iii) $x \in C_0$ oder $y \in C_0 \Rightarrow x \in C$ oder $y \in C$.

(iv) $x, y \in V_0 \Rightarrow x \in D$ oder $y \in D$.

(b) Zu zeigen: $C_0 \subseteq C^*$ für ein optimales VC C^* von G.
Wir setzen $S := C^*$ und definieren weiter $S_V := S \cap V_0$, $S_C := S \cap C_0$, $S_I := S \cap I_0$, $\overline{S_I} := I_0 \setminus S_I$. Dabei gilt $S = S_V \dot\cup S_C \dot\cup S_I$ und damit $|S| = |S_V| + |S_C| + |S_I|$.
Wir zeigen zuerst, dass $C_{B_1} := (V \setminus \overline{S_I}) \cup S'_C$ ein VC von B ist (wobei S'_C die Kopie von S_C ist).
Sei also $(x, y') \in E_B$. Zu zeigen ist nun, dass $x \in C_{B_1}$ oder $y' \in C_{B_1}$.

- $x \notin \overline{S_I} \Rightarrow x \in C_{B_1}$
- $x \in \overline{S_I} \Rightarrow x \in I_0 \overset{\text{wie in (a)}}{\Rightarrow} y \in C_0$. Andererseits folgt aus $x \in \overline{S_I}$ aber auch $x \notin S_I$. Da also $x \in I_0$ und $x \notin S_I$, gilt $x \notin S = C^*$.
  Da C^* VC von G und $x \notin S = C^*$, folgt $y \in S$. Da aber auch $y \in C_0$, folgt $y \in S \cap C_0 = S_C$. Also erhalten wir insgesamt $y' \in S'_C$ und damit insbesondere $y' \in C_{B_1}$.

Bleibt zu zeigen, dass C_0 eine Teilmenge von $S = C^*$ ist.
Wir zeigen zuerst

$$|C_0| \leq |S \setminus S_V| \tag{7.1}$$

Dazu betrachten wir folgende Ungleichung

$$\begin{aligned} |V_0| + 2|C_0| &= |V_0 \dot\cup C_0 \dot\cup C'_0| = |C_B| \\ &\leq |C_{B_1}| = |V \setminus \overline{S_I}| + |S'_C| \\ &= |(V_0 \dot\cup C_0 \dot\cup C'_0) \setminus (I_0 \setminus S_I)| + |S'_C| \\ &= |V_0| + |C_0| + |S_I| + |S_C| \end{aligned}$$

Umstellen ergibt

$$|C_0| \leq |S_I| + |S_C| = |S_I| + |S_C| + |S_V| - |S_V| = |S \setminus S_V|$$

Aus Gleichung 7.1 folgt nun

$$|C_0 \dot\cup S_V| = |C_0| + |S_V| \leq |S \setminus S_V| + |S_V| = |S| = |C^*|$$

Da S_V ein VC von $\langle V_0 \rangle$ ist, folgt mit (a), dass $C_0 \cup S_V$ ein VC von G ist.
Da $|C_0 \dot\cup S_V| \leq |C^*|$ und C^* ein optimales VC von G, folgt insgesamt, dass $C_0 \dot\cup S_V$ ein optimales VC von G ist, welches C_0 beinhaltet.

(c) Sei S_0 ein (beliebiges) optimales VC von $\langle V_0 \rangle$. Wegen (a) ist $S_0 \cup C_0$ ein VC von G. Außerdem gilt per Konstruktion von B, dass $C_0 \cup C_0' \cup S_0 \cup S_0'$ ein VC von B ist. Also gilt

$$|V_0| + 2|C_0| = |C_B| \leq |C_0 \cup C_0' \cup S_0 \cup S_0'| = 2|C_0| + 2|S_0|$$

Daraus folgt $|V_0| \leq 2|S_0|$.

□

7.2 Tiefenbeschränkte Suchbäume

7.2.1 Einführung

Im Folgenden sei $T = (V, E)$ ein Baum. T heißt *gewurzelt*, falls es einen Knoten v mit $deg(v) > 1$ gibt, der als Wurzel ρ gekennzeichnet ist. Knoten in T mit Grad 1 heißen *Blätter* und wir bezeichnen mit $L \subseteq V$ die Menge der Blätter.

Übungsaufgabe: Wenn jeder Knoten $v \in V \setminus L$, der kein Blatt ist, mindestens 2 Kinder hat, so gilt:

$$|V| \leq 2|L| - 1.$$

Dies liefert uns $\mathcal{O}(|V|) = \mathcal{O}(|L|)$.

Sei nun A ein rekursiver Algorithmus, sodass jeder einzelne Aufruf von A Komplexität $\mathcal{O}(f(n))$ hat. Dann hat A insgesamt eine Komplexität von $\mathcal{O}(|L| \cdot f(n))$, wobei L die Menge der Blätter des resultierenden Rekursionsbaums ist.

Branching-Algorithmus: Wir betrachten jetzt einen einfachen Branching-Algorithmus für ein parametrisiertes Problem (Π, k), sodass nach dessen Ausführung p kleinere Probleme gelöst werden müssen, in denen die Parameter von oben durch $k - k_1, k - k_2, \ldots, k - k_p$ beschränkt sind (siehe Abb. 7.2).

Der Vektor $(k_1, \ldots, k_p)$ heißt *Verzweigungsvektor/branching-Vektor* und für die Anzahl der Blätter $L(T_k)$ von T_k gilt

$$L(T_k) \geq \sum_{i=1}^{p} L(T_{k-k_i}).$$

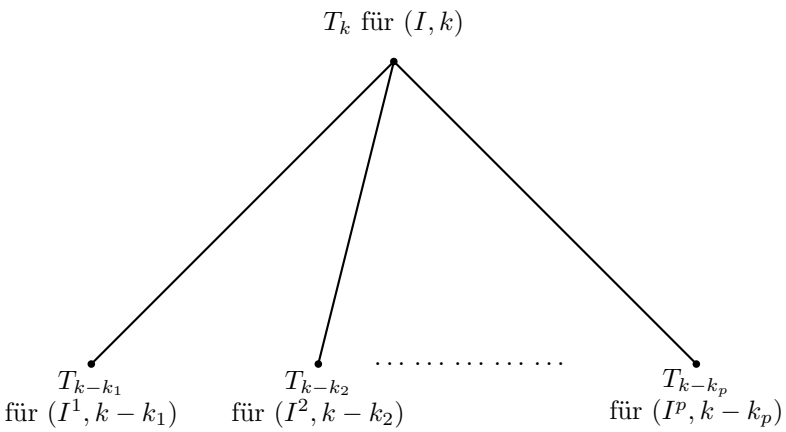

Abbildung 7.2: Verzweigungsbaum für das Problem (Π, k)

Sei $L(T_k) = \alpha^k$ mit einer Konstanten $\alpha \geq 1$ $(k_1, \ldots, k_p)$ der entsprechende Verzweigungsvektor, so gilt

$$\alpha^k \geq \sum_{i=1}^{p} \alpha^{k-k_i} \Leftrightarrow \alpha_k - \sum_{i=1}^{p} \alpha^{k-k_i} \geq 0.$$

Beispiel. Wir betrachten erneut das VC-Problem. Ein Ansatz zur Lösung ist, für jede Kante (u, v) von G zu entscheiden, ob $u \in VC$ oder $v \in VC$ (siehe Abb. 7.3). Dadurch erhalten wir in jedem Schritt den Verzweigungsvektor $(1, 1)$ und damit $\alpha^k \geq \alpha^{k-1} + \alpha^{k-1} \overset{\alpha \geq 1}{\Leftrightarrow} \alpha - 2 \geq 0$. Die zugehörige Nullstelle ist $\alpha = 2$ und wir erhalten $L(T_k) = 2^k$.

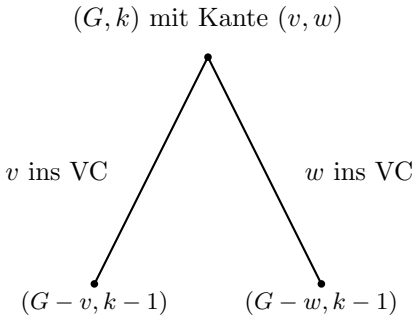

Abbildung 7.3: Verzweigungsbaum für das VC-Problem

Das *charakteristische Polynom* für einen Verzweigungsvektor $(k_1, \ldots, k_p)$ ist gegeben durch

$$f(k_1, \ldots, k_p) = z^r - \sum_{i=1}^{p} z^{r-k_i},$$

wobei $r = \max\{k_1, \ldots, k_p\}$.
Für das obige Beispiel mit Verzweigungsvektor $(1, 1)$ ergibt sich also $z^1 - z^0 - z^0 = z - 2$.

Anmerkungen.

- Wenn α Nullstelle des charakteristischen Polynoms $f(k_1, \ldots, k_p)$ ist, dann gilt $\mathcal{O}(L(T_k)) = \mathcal{O}(\alpha^k)$.
- In unserem Fall ist α immer ein eindeutiger, positiver, reeller Wert und wird *branching-Faktor* genannt.
- Falls mehrere Fälle mit verschiedenen Verzweigungsvektoren auftreten, so ist $L(T) = \max\{\alpha^k \mid \alpha \text{ ist branching-Faktor der verschiedenen Fälle}\}$.
- Branching-Faktoren geben immer das „worst-case"-Szenario an; oft geht es schneller.

7.2.2 Beispiel – Vertex Cover

Das Ziel des folgenden Abschnitts ist es, den Suchbaum für das VC-Problem zu verkleinern. Dazu sollen verschiedene Fälle geschickt unterschieden werden, sodass sich Verzweigungsvektoren ergeben, aus welchen sich sehr „kleine" Suchbäume berechnen lassen.

Einfache Fallunterscheidung: Für eine Eingabe $G = (V, E)$ und $k \geq 3$ unterschieden wir (in dieser Reihenfolge) folgende Fälle:

(1) $\exists v \in V$ mit $deg(v) = 1$, d.h. $N(v) = \{a\}$:
- Nimm a in das VC auf.
- Setze $G \leftarrow G - a$ und $k \leftarrow k - 1$ (kein Verzweigunsvektor)

(2) $\exists v \in V$ mit $deg(v) = 2$, d.h. $N(v) = \{a, b\}$:

(2.1) $\exists (a, b) \in E$:
- Nimm a und b in das VC auf.
- Setze $G \leftarrow G - \{a, b\}$ und $k \leftarrow k - 2$ (kein Verweigunsvektor)

(2.2) $\nexists (a, b) \in E$ und $N(a) = N(b) = \{v, x\}$:
- Nimm v und x in das VC auf.
- Setze $G \leftarrow G - \{v, x\}$ und $k \leftarrow k - 2$ (kein Verzweigungsvektor)

(2.3) $\nexists (a, b) \in E$ und $|N(a) \cup N(b)| \geq 3$:
- Nimm entweder $N(v) = \{a, b\}$ oder $N(a) \cup N(b)$ in das VC auf.
- Setze also $G \leftarrow G - \{a, b\}, k \leftarrow k - 2$ bzw.
  $G \leftarrow G - (N(a) \cup N(b)), k \leftarrow k - \underbrace{|N(a) \cup N(b)|}_{\geq 3}$ $\Rightarrow$ Verzweigunsvektor $(2, 3)$
- Charakteristisches Polynom: $z^3 = z + 1 \Rightarrow z = 1,3247$

(3) $\forall v \in V$ gilt $deg(v) \geq 3$:

- Nimm entweder v oder $N(v)$ in das VC auf.
- Setze also $G \leftarrow G - v, k \leftarrow k - 1$ bzw. $G \leftarrow G - N(v), k \leftarrow k - |N(v)| \Rightarrow$ Verweigungsvektor $(1, 3)$
- Charakteristisches Polynom: $z^3 = z^2 + 1 \Rightarrow z = 1,4656$

Im worst case erhalten wir also $L(T_k) = \mathcal{O}(1,4656^k)$. Dies können wir in Theorem 7.5 einsetzen und erhalten

Theorem 7.7. *Das Problem para-VC kann in $\mathcal{O}(kn + 1,4656^k k^2)$ Zeit gelöst werden.*

7.3 Vererbbare Grapheigenschaften und Graphmodifikation

7.3.1 Einführung

Definition (Grapheigenschaft)**.** Eine *Grapheigenschaft* ist eine Graphklasse, die bezüglich Graph-isomorphie abgeschlossen ist. Also, zwei isomorphe Graphen haben die Eigenschaft, oder eben nicht.

Beispiele.
Beispiel für Grapheigenschaften:

- zusammenhängende Graphen
- bipartite Graphen
- Bäume

Dagegen keine Grapheigenschaften sind:

- Graphen, deren Knoten mit natürlichen Zahlen gelabelt sind.
- Graphen, deren Kanten positiv gewichtet sind.

Definition (Vererbbarkeit)**.** Eine Grapheigenschaft Σ ist *vererbbar*, wenn für $G \in \Sigma$ auch $H \in \Sigma$ für alle induzierten Teilgraphen $H \subseteq G$ von G.

Beispiele.

- bipartite Graphen
- vollständige Graphen
- planare Graphen
- Wälder (aber nicht Bäume, da induzierte Teilgraphen von Bäumen auch Wälder sein können)

- Cographen

Definition (Charakterisierung durch Ausschlussmenge)**.** Eine Grapheigenschaft Σ besitzt eine *Charakterisierung durch eine Ausschlussmenge*, wenn es eine Graphklasse $\mathbb{F}$ gibt, sodass für alle induzierten Teilgraphen $H \subseteq G$ gilt:

$$G \in \Sigma \Leftrightarrow H \notin \mathbb{F}$$

Die Elemente aus $\mathbb{F}$ heißen auch *verbotene* (induzierte) Teilgraphen.

Lemma 7.8. *Eine Grapheigenschaft Σ ist vererbbar $\Leftrightarrow$ Σ besitzt eine Charakterisierung durch eine Ausschlussmenge $\mathbb{F}$.*

Beweis.

„$\Rightarrow$“ Wähle $\mathbb{F}$ als die Menge der Graphen, die nicht in Σ liegen.

„$\Leftarrow$“ Sei $G \in \Sigma$, wobei Σ über die Ausschlussmenge $\mathbb{F}$ charakterisiert sei. Sei weiter $H \subseteq G$ ein induzierter Teilgraph von G. Da G keinen induzierten Teilgraphen aus $\mathbb{F}$ besitzt, gilt dies auch für H. Somit ist die Grapheigenschaft Σ vererbbar.

□

Anmerkung. In der Regel sind dabei endliche Ausschlussmengen von Interesse.

7.3.2 Beispiel: Cographen

Cographen (complement-reducible graphs) sind rekursiv definiert, wobei verschiedene (äquivalente) Definitionen existieren:

Definition (Definition 1)**.**

(1) K_1 ist ein Cograph.

(2) Wenn G ein Cograph ist, so ist auch das Komplement $\overline{G}$ ein Cograph.

(3) Wenn G und H Cographen sind, so ist auch ihre disjunkte Vereinigung $G + H = G \dot\cup H$ ein Cograph.

Definition (Definition 2)**.**

(1) K_1 ist ein Cograph.

(2) Wenn G und H Cographen sind, so ist auch ihre disjunkte Vereinigung $G + H = G \dot\cup H$ ein Cograph.

(3) Wenn G und H Cographen sind, so ist auch ihr join $G \oplus H$ ein Cograph, wobei $G \oplus H = (V(G) \cup V(H), E(G) \cup E(H) \cup E')$ mit $E' = \{(x, y) \mid x \in V(G), y \in V(H)\}$.

Anmerkung. Eigenschaft (2) aus Definition 1 und Eigenschaft (3) aus Definition 2 sind äquivalent, da $G \oplus H = \overline{(\overline{G} + \overline{H})}$.

Cographen sind interessant, da viele $\mathcal{NP}$-schwere Probleme auf Cographen „einfach“ werden.

Jeder Cograph besitzt außerdem eine eindeutige Repräsentation als Cobaum:
Ein Cobaum ist dabei ein Baum, bei dem alle inneren Knoten mit 0 oder 1 gelabelt sind. Für einen Cograph G und den dazugehörigen Cobaum T gilt dabei folgender Zusammenhang: 2 Knoten x, y sind genau dann in G durch eine Kante verbunden, wenn der letzte gemeinsame Vorfahre der Blätter x und x in T mit 1 gelabelt ist, d.h. $(x, y) \in E(G) \Leftrightarrow lca(x, y) = 1$ (siehe Abb. 7.4).

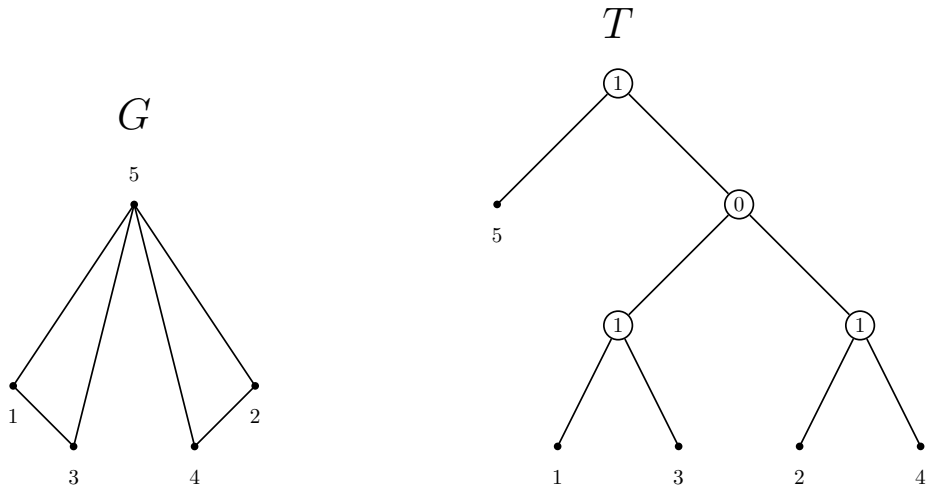

Abbildung 7.4: Cograph G und Cobaum T

Theorem 7.9. *G ist Cograph $\Leftrightarrow$ G enthält keinen Weg auf 4 Knoten (P_4) als induzierten Teilgraph.*

Beweis. Wir zeigen zunächst:
Wenn G ein Cograph ist und $G \neq K_1$ $\Rightarrow$entweder G ist zusammenhängend oder $\overline{G}$ ist zusammenhängend.

- G nicht zusammenhängend $\Rightarrow$ $G = H_1 + H_2 + \ldots + H_r$, wobei $H_i, i = 1, \ldots, r$ die Zusammenhangskomponenten von G sind $\Rightarrow$ $\overline{G}$ zusammenhängend.
- G zusammenhängend $\Rightarrow$ $G = H_1 \oplus H_2$ $\Rightarrow$ $\overline{G}$ nicht zusammenhängend.

Nun zeigen wir die eigentliche Aussage des Satzes:

> „$\Rightarrow$“: Angenommen G ist ein Cograph und enthält einen P_4 als induzierten Teilgraph. Da das Komplement eines P_4 wieder ein P_4 ist, kann G nicht auf einen K_1 „reduziert“ werden $\lightning$.

„⇐“: Widerspruchsbeweis. Sei $G = (V, E)$ ein minimales Gegenbeispiel, d.h. G ist P_4-frei und ist kein Cograph, aber jeder P_4-freie Graph G' mit $|V(G')| < |V(G)|$ ist ein Cograph.
Wir zeigen zunächst, dass G und $\overline{G}$ zusammenhängend sind:

- Angenommen G ist nicht zusammenhängend
  $\Rightarrow G = H_1 + H_2$
  $\Rightarrow H_1$ und H_2 sind P_4-frei und $|V(H_i)| < |V(G)|$ für $i = 1, 2$
  $\Rightarrow H_1$ und H_2 sind Cographen
  $\Rightarrow G = H_1 + H_2$ ist ein Cograph ↯
- Angenommen $\overline{G}$ ist nicht zusammenhängend
  $\Rightarrow \overline{G} = H_1 + H_2$. Außerdem ist $\overline{G}$ P_4-frei (da G P_4-frei)
  $\Rightarrow$ H_1 und H_2 sind P_4 frei und damit Cographen $\Rightarrow \overline{G} = H_1 + H_2$ ist ein Cograph $\Rightarrow \overline{\overline{G}} = G$ ist ein Cograph ↯

Sei nun $x \in V(G)$. Dann ist $G - x$ ein Cograph.
oBdA können wir annehmen, dass $G - x$ nicht zusammenhängend ist (sonst betrachte das Komplement $\overline{G - x}$).
Da $\overline{G}$ zusammenhängend ist, existiert ein Knoten $y \in V(G)$, der nicht adjazent zu x ist.
Sei C die Zusammenhangskomponente von $G - x$, die y enthält. Da außerdem auch G zusammenhängend ist, hat x einen Nachbarn z in C, d.h. es existiert ein $z \in C$, sodass $(x, z) \in E(G)$. Insbesondere können wir zwei adjazente Knoten u und v in C finden, sodass $(u, v) \in E(G)$, $(u, x) \in E(G)$, aber $(v, x) \notin E(G)$.
Sei D eine weitere Zusammenhangskomponente von $G - x$ und sei $d \in D$ ein Nachbar von x, d.h. $(x, d) \in E(G)$.
Dann enthält G einen induzierten P_4 auf den Knoten (v, u, x, d) ↯

□

Anmerkung. Cographen und deren Verallgemeinerungen spielen eine zentrale Rolle in der Biomathematik und Bioinformatik, um Genbäume aus geschätzten evolutionären Verwandtschaftsverhältnissen zwischen Genen zu rekonstruieren.

Diese Schätzungen sind oft (aufgrund von Rauschen in den Daten oder Messfehlern) keine Cographen.

Ziel ist es, sie in wenigen Schritten zu Cographen zu editieren.

7.3.3 Σ-Graphmodifikation

Σ-Graphmodifikation
Gegeben: Graph $G = (V, E)$, natürliche Zahlen $i, j, k \in \mathbb{N}_0$
Frage: Kann man durch

- Löschen von bis zu i Knoten aus G
- Löschen von bis zu j Kanten aus G
- Hinzufügen von bis zu k Kanten zu G

einen Graphen aus Σ erhalten?

Theorem 7.10. *Σ-Graphmodifikation ist $\mathcal{NP}$-vollständig für jede nicht-triviale Grapheigenschaft Σ (Eine Grapheigenschaft Σ ist nicht-trivial, falls es unendlich viele Graphen gibt, die in Σ liegen und unendlich viele, die nicht in Σ liegen) (ohne Beweis)*

Anmerkung. Bereits die folgenden Spezialfälle sind $\mathcal{NP}$-hart:

- nur Knoten löschen
- nur Kanten löschen
- Knoten/Kanten löschen
- nur Kanten hinzufügen

para-Σ-Graphmodifikation
Gegeben: Graph $G = (V, E)$, Parameter $i + j + k$
Frage: Kann man durch Löschen von bis zu i Knoten und bis zu j Kanten aus G , sowie Hinzufügen von bis zu k Kanten zu G einen Graphen aus Σ erhalten?

Wir werden im Folgenden zeigen, dass das Problem para-Σ-Graphmodifikation FPT ist. Dazu betrachten wir zunächst folgendes Lemma.

Lemma 7.11. *Sei Σ eine vererbbare Grapheigenschaft, die für einen Graphen $G = (V, E)$ in Zeit $t(G)$ überprüft werden kann. Dann kann für jeden Graphen $G \notin \Sigma$ in Zeit $\mathcal{O}(|V(G)| \cdot t(G))$ ein minimaler verbotener induzierter Teilgraph gefunden werden.*

Beweis. Sei $G = (V, E) \notin \Sigma$ mit $V = \{v_1, \ldots, v_n\}$. Der folgende Algorithmus bestimmt in Zeit $\mathcal{O}(|V(G)| \cdot t(G))$ einen minimalen verbotenen induzierten Teilgraphen H von G:

```
1: H ← G
2: for i = 1, ..., n do
3:     if H − v_i ∉ Σ then
4:         H ← H − v_i
5:     end if
6: end for
7: return H
```

□

Theorem 7.12. *Sei Σ eine vererbbare Grapheigenschaft, welche durch eine endliche Ausschlussmenge charakterisiert ist. Dann ist das Problem para-Σ-Graphmodifikation (i, j, k) lösbar in Zeit $\mathcal{O}(N^{i+2j+2k} \cdot |V|^{N+1})$, wobei N die maximale Knotenanzahl der verbotenen Teilgraphen ist.*

Beweis. Wir geben direkt einen FPT-Algorithmus für das Problem an:
GRAPH_MOD$(G = (V, E), i, j, k)$

```
1: if G ∈ Σ then
2:     return true
3: end if
4: H = (V_H, E_H) sei ein minimaler verbotener induzierter Teilgraph von G
5: if i > 0 then
6:     for all v ∈ V_H do
7:         if GRAPH_MOD(G − v, i − 1, j, k)) then
8:             return true
9:         end if
10:    end for
11: end if
12: if j > 0 then
13:    for all e = (v, w) ∈ E_H do
14:        if GRAPH_MOD(G − e, i, j − 1, k)) then
15:            return true
16:        end if
17:    end for
18: end if
19: if k > 0 then
20:    for all e = (v, w) ∈ V_H × V_H, v ≠ w, e = (v, w) ∉ E_H do
21:        if GRAPH_MOD(G + e, i, j, k − 1)) then
22:            return true
23:        end if
24:    end for
25: end if
26: return false
```

Die Korrektheit des Algorithmus ist klar.

Die Laufzeit ergibt sich wie folgt: Nach Lemma 7.11 kann H in $\mathcal{O}(|V|\cdot|V|^N)$ gefunden werden. Wir können maximal N Knoten aus H löschen und maximal $\binom{N}{2}$ Kanten löschen bzw. hinzufügen. Also werden höchstens $\binom{N}{2}^{j+k} \cdot N^i \in \mathcal{O}(N^{i+2j+2k}$ Graphen „erzeugt“. Ob $G \in \Sigma$, kann in Zeit $\mathcal{O}(|V|^N)$ getestet werden, denn alle $\binom{|V(G)|}{|V(H)|} \leq |V|^N$ Teilgraphen der Größe V_H müssen untersucht werden.
Insgesamt ergibt sich also eine Laufzeit von $\mathcal{O}(N^{i+2j+2k} \cdot |V|^{N+1})$.

□