Kombinatorische Optimierung

Mitschrift von www.kuertz.name

Hinweis: Dies ist **kein offizielles Script**, sondern nur eine private Mitschrift. Die Mitschriften sind teweilse **unvollständig, falsch oder inaktuell**, da sie aus dem Zeitraum 2001–2005 stammen. Falls jemand einen Fehler entdeckt, so freue ich mich dennoch über einen kurzen Hinweis per E-Mail – vielen Dank!

Mihhail Aizatulin (avatar@hot.ee)

Inhaltsverzeichnis

1 Grundbegriffe

1.1 Elementares

DEFINITIONEN:

- Sei $\mathbb{N} = \{1, 2, \ldots\}$ und $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.
- Für $r \in \mathbb{R}$ setze $[r] = \{ n \in \mathbb{N} \mid n \leq r\}$.
- Für eine Menge M und $n \in \mathbb{N}_0$ setze

$$\binom{M}{n} = \{T \subseteq M \mid |T| = n\}$$

  Es gilt:

$$\left|\binom{M}{n}\right| = \binom{|M|}{n} = \frac{|M|!}{n!(|M|-n)!}$$

- Für zwei Mengen A, B sei die *symmetrische Differenz*

$$A \,\Delta\, B = (A \cup B) \setminus (A \cap B)$$

1.2 Graphen

DEFINITIONEN:

- Ein endlicher Graph ist ein Paar $G = (V, E)$, wobei V eine endliche Menge ist und $E \subseteq \binom{V}{2}$. Vereinbarung: $n := |V|$ und $m := |E|$.
- Sprechweisen: falls $\{u, v\} \in E$, so sagen wir: „u und v sind benachbart (adjazent)“. Falls $u \in V, e \in E, u \in e$: „$u$ und e sind inzident“.
- Für $e \in E$ und $u \in V$ setze

$$\begin{aligned} G - e &:= (V, E \setminus \{e\}) \\ G - v &:= \left(V \setminus \{v\}, E \cap \binom{V \setminus \{v\}}{2}\right) \end{aligned}$$

- Für $n \in \mathbb{N}$ heißt $K_n = \left([n], \binom{[n]}{2}\right)$ *vollständiger* Graph und $E_n = ([n], \emptyset)$ *leerer* Graph.
- Seien $G = (V, E)$ und $H = (U, F)$ Graphen. H heißt *Teilgraph* von G, falls $U \subseteq V$ und $F \subseteq E$ ist. H heißt *induzierter* Teilgraph, falls zusätzlich $F = E \cap \binom{U}{2}$.

- Ein vollständiger Teilgraph heißt Clique, ein leerer induzierter Teilgraph heißt *unabhängige Menge*.
- Der *Grad* $d(v)$ eines Knoten ist die Anzahl der Kanten, die mit v inzident sind. Der *Minimalgrad* von G ist $\delta(G) = \min\{d(v) \mid v \in V\}$, der *Maximalgrad* ist $\Delta(G) = \max\{d(v) \mid v \in V\}$.
- Ist $G = (V, E)$ ein Graph, so sei $\bar{G} = \left(V, \binom{V}{2} \setminus E\right)$.
- Eine *Knotenfärbung* ist eine Funktion $f : V \to \mathbb{N}$ mit der Eigenschaft, dass $f(u) \neq f(v)$ für alle $\{u, v\} \in E$.
- Ein Graph heißt *k-färbbar* ($k \in \mathbb{N}$), wenn es eine Knotenfärbung $f : V \to [k]$ gibt.

SATZ: Jeder planare Graph ist 4-färbbar. Bewiesen 1976 von Haken und Appel mit Hilfe eines Computers.

DEFINITIONEN:

- $\chi(G) := \min\{k \in \mathbb{N}_0 \mid G \text{ ist } k\text{-färbbar}\}$ - *chromatische Zahl*.
- $\omega(G) := \max\{k \in \mathbb{N}_0 \mid G \text{ enthält Clique auf } k \text{ Knoten}\}$ - *Cliquenzahl*
- $\alpha(G) := \max\{k \in \mathbb{N}_0 \mid G \text{ enthält eine Unabhängige Menge auf } k \text{ Knoten}\}$

BEMERKUNG: $w(G) = \alpha(\bar{G})$.

DEFINITIONEN: O-Notation. Seien $f, g : \mathbb{N} \to \mathbb{R}$.

- Man schreibt $f = O(g)$, falls ein $n_0 \in \mathbb{N}$ und $c \in \mathbb{R}_{>0}$ existieren, so dass für alle $n \in \mathbb{N}$ gilt: $n \geq n_0 \Rightarrow f(n) \leq cg(n)$.
- $f = \Omega(g)$, falls $g = O(f)$.
- $f = \Theta(g)$, falls $f = O(g)$ und $g = O(f)$.

BEISPIELE:

- Sei $f(n) = 170n^2 + 13n - 21$ und $g(n) = n^2$. Dann ist $f = O(g)$. Schreibweise: $f = O(n^2)$.
- Sei $f(n) = \log_2(n)$. Dann ist $f = \Theta(\log_{10} n) = O(\log n)$.

1.3 Probleme, Effizienz, Kodierung

PROBLEM **Clique.** Wir betrachten drei Varianten:

- Entscheidungsproblem. Eingabe: Graph G und $k \in \mathbb{N}_0$. Frage: gibt es eine Clique auf k Knoten in G?
- Optimierungsproblem. Eingabe: Graph G. Frage: was ist die Mächtigkeit einer größten Clique?
- Suchproblem. Eingabe: Graph G. Aufgabe: finde eine größte Clique.

In der Theorie betrachtet man in der Regel die Entscheidungsprobleme.

SATZ: Wenn es für eines der drei Probleme Clique-Entscheidung, Clique-Optimierung oder Clique-Suche einen Algorithmus gibt, dessen Laufzeit polynomiell in der Knotenzahl n beschränkt ist, dann auch für die beiden anderen.

BEWEIS:

- Entscheidung → Optimierung. Algorithmus für Optimierung: Mit dem Entscheidungsalgorithmus teste für alle $k \in [n]$, ob eine Clique der Größe k existiert. Ausgabe: größtes k mit positiver Entscheidung. Der Algorithmus ist korrekt. Zur Laufzeit: sei p das Polynom, das die Laufzeit des Entscheidungsalgorithmus beschränkt. Dann ist die Laufzeit des Optimierungsalgorithmus $np(n)$.
- Optimierung → Suche. Algorithmus für Suche: Für alle $e \in E$ do: Falls $\omega(G - e) = \omega(G)$, setze $G = G - e$. Ausgabe: G.

  Behauptung: Der Algorithmus terminiert mit einer maximalen Clique von G. *Beweis:* Sei C die Ausgabe des Algorithmus. Dann gilt: $\omega(C) = \omega(G)$. Ferner gilt: $\omega(C - e) < \omega(C)$ für alle Kanten $e \in E(C)$. Also bilden die Kanten von C eine Clique.

  Zur Laufzeit: Sei p die Laufzeitschranke des Optimierungsalgorithmus. Dann hat der Suchalgorithmus die Laufzeit höchstens $|E|\, p(n) \leq n^2 p(n)$.
- Suche → Entscheidung. Trivial.

PROBLEM **Travelling Salesman (TSP)** Entscheidungsvariante: Eingabe: $d_{ij} \in \mathbb{N}, i, j \in [n], k \in \mathbb{N}$. Dabei bezeichnet d_{ij} die Distanz zwischen zwei Städten i und j. Ausgabe: Gibt es eine „Tour“ durch alle Städte mit Gesamtlänge kleiner gleich k?

Problemgröße (Kodierungslänge der Eingabe) ist $n^2 \log_2(\max d_{ij})$. Der Logarithmus kommt dadurch, dass die Zahlen im Rechner einen logarithmischen Platzverbrauch für die Darstellung haben.

DEFINITION: *Problemgröße* ist die binäre Kodierungslänge des Problems (wieviele Bits brauche ich um das Problem in meinem Rechner zu beschreiben?).

Unter *Effizienz* verstehen wir: Die Rechenzeit zum Lösen eines Problems soll ein Polynom in der Problemgröße sein. Das definieren wir später formal.

BEISPIELE:

- Graphenkodierung über Adjazenzmatrix. Den Graphen $G = ([n], E)$ kodieren wir mit $A = (a_{ij})_{i,j\in[n]}$ mit

$$a_{ij} \quad = \quad \begin{cases} 1, & \text{falls } \{i,j\} \in E \\ 0 & \text{sonst} \end{cases}$$

  Kodierungslänge: n^2.

- Natürliche Zahlen. Darstellung im Binärsystem. Für $n \in \mathbb{N}$ benötigen wir $\lceil \log_2 n \rceil$ Bits.

- Problem PRIMES. Eingabe: $n \in \mathbb{N}$. Ausgabe: ist n eine Primzahl? Sieb des Eratosthenes hat den Aufwand $O(n^{3/2})$. Nicht effizient, weil die Größe der Eingabe gleich $\log n$ ist.

- Gauß-Elimination zum Lösen von linearen Gleichungssystemen hat den Aufwand von $O(n^3)$ und ist somit effizient.

- Clique: Durchprobieren aller Knotenmengen. Aufwand: $O(2^n n^2)$, nicht effizient.

SATZ: (Agrawal, Kayal, Saxena (2002)). Es gibt einen polynomiellen Algorithmus für PRIMES. Die Laufzeit ist etwa $O((\log n)^6)$.

2 Komplexitätstheorie

2.1 Algorithmen

DEFINITIONEN:

- Ein *Alphabet* Σ ist eine endliche Menge. Mit Σ^* bezeichnen wir die Menge der endlichen Folgen (Wörter) über Σ. Für $x \in \Sigma^*$ sei $|x|$ die Länge von x. Ein *Algorithmus* berechnet auf eindeutige Weise eine Funktion $\Sigma^* \to \Sigma^*$.
- Die *Laufzeit* eines Algorithmus ist die Zahl der benötigten elementaren Operationen (elementare Arithmetik, Speicherzugriffe, Vergleiche...)
- Für einen Algorithmus A und $x \in \Sigma^*$ bezeichne $t_A(x)$ die Anzahl der elementaren Operationen, die A mit der Eingabe x ausführt. Die *zeitliche Komplexität* von A ist

$$t_A : \mathbb{N} \to \mathbb{N}; n \mapsto \max\{t_A(x) \mid x \in \Sigma^*, |x| = n\}$$

  Ein Algorithmus heißt *effizient*, wenn es ein $c \in \mathbb{N}$ gibt mit $t_A = O(n^c)$.

2.2 Die Klasse P

DEFINITIONEN: Sei Σ ein Alphabet, das die Zeichen 0 und 1 umfaßt.

- Eine Teilmenge $L \subseteq \Sigma^*$ heißt *Sprache*.
- Ein *Entscheidungsproblem (EP)* ist ein Tripel (L, U, Σ) mit $L, U \subseteq \Sigma^*$.
- Ein Algorithmus A *löst* das EP (L, U, Σ), wenn für alle $x \in U$ gilt: Ausgabe von $A(x)$ ist gleich 1 für $x \in L$ und 0 für $x \notin L$.

Im folgenden werden wir meistens nur Entscheidungsprobleme betrachten.

DEFINITION: Sei $\mathsf{P} = \{ L \subseteq \Sigma^* \mid$ es existiert ein effizienter Algorithmus, der das Entscheidungsproblem (L, Σ^*, Σ^*) löst $\}$ die Klasse der *polynomiell entscheidbaren* Sprachen.

DEFINITIONEN:

- Sei $G = (V, E)$ ein Graph. Ein *Weg* in G ist eine (endliche) Knotenfolge $v_1, \ldots, v_k$ mit $\{v_i, v_{i+1}\} \in E$ für alle $i \in [k-1]$.
- Zwei Knoten heißen *miteinander verbunden*, wenn es einen Weg zwischen ihnen gibt. Für $v \in V$ heißt die Menge

$$C(v) = \{\, u \in v \mid u \text{ und } v \text{ sind verbunden} \,\}$$

Zusammenhangskomponente von v. G heißt *zusammenhängend*, wenn G nur eine Zusammenhangskomponente besitzt.

PROBLEM **Graphensuche.** Eingabe:

- $G = (V, E)$, dargestellt durch Adjazenzlisten: für jedes $v \in V$ haben wir eine Liste $A(v)$ der Nachbaren von v.[1]
- Startknoten $v_1 \in V$.

Ausgabe: Zusammenhangskomponente von v_1.

Algorithmus Graphensuche. Für den Algorithmus verwenden wir die Datenstruktur Q: irgendeine Datenstruktur, die ermöglicht, in konstanter Zeit einen Knoten hinzuzufügen oder einen Knoten auszulesen und zu löschen. Beispiele sind Stack (LIFO - Last In First Out) und Warteschlange (FIFO - First In First Out).

```
lösche alle Knotenmarkierungen
Q = v_1, markiere v_1
solange Q ≠ ∅
        wähle v ∈ Q und lösche v aus Q
        für alle v' ∈ A(v):
                falls v' nicht markiert:
                        füge v' zu Q hinzu und markiere v'

Ausgabe: alle markierten Knoten
```

SATZ: Der Algorithmus Graphensuche ist korrekt und hat eine Laufzeit von $O(n + m)$.

BEWEIS:

- Korrektheit: Genau die Knoten sind markiert, die mit v_1 verbunden sind. Angenommen, es existiert ein Knoten u, der mit v_1 verbunden ist,

aber nicht markiert wurde. Sei $v_1, \ldots, v_{k-1}, u$ ein Weg in G. O.B.d.A. können wir annehmen, dass $v_1, \ldots, v_{k-1}$ markiert wurden. Nachdem v_{k-1} markiert wurde, wurde die Anweisung „Für alle $v' \in A(v_{k-1}) \ldots$“ ausgeführt. Da $u \in A(v_{k-1})$ und u nicht markiert ist, wurde u zu Q hinzugefügt. Da der Algorithmus mit $Q = \emptyset$ terminiert, wurde u irgendwann aus Q entfernt und wurde dabei markiert.

Weiter ist zu zeigen, dass nur die Knoten markiert wurden, die mit v_1 verbunden sind. Dazu verifiziert man die folgende Invariante: *„Alle markierten Knoten und alle Knoten aus Q sind mit v_1 verbunden“*

- Laufzeit: Löschen aller Markierungen benötigt $O(n)$. Die Schleife wird höchstens einmal für jeden Knoten durchlaufen. Dabei wird die Anweisung „falls …“ $\deg(v)$ mal ausgeführt. Der Aufwand ist also

$$\sum_{v \in V} \deg(v) = 2m = O(m)$$

□

Falls man für die Graphensuche Stack verwendet, so heißt sie *Tiefensuche*, mit Queue *Breitensuche*.

SATZ: Das Problem „Sind zwei Knoten in einem Graphen verbunden?“ liegt in P.

BEWEIS: Starte Graphensuche mit dem Graphen und einem der beiden Knoten. Falls der zweite markiert wurde, sind sie verbunden, sonst nicht. □

2.3 Die Klasse NP

DEFINITIONEN:

- Sei $L \subseteq \Sigma^*$. Ein (deterministischer) Algorithmus $A\colon \Sigma^* \times \Sigma^* \to \{0,1\}$ heißt *Verifier* für L, wenn L genau aus den Worten x besteht, für die es ein $c \in \Sigma^*$ gibt mit $A(x,c) = 1$. Dabei heißt c *Zertifikat*.
- Ein *polynomieller Verifier* ist ein Verifier, für den es ein $d \in \mathbb{N}$ gibt so, dass für jedes $x \in L$ ein $c \in \Sigma^*$ existiert mit $t_A(x,c) = O(|x|^d)$.

BEMERKUNG: Wir gehen davon aus, dass Algorithmen die Eingabe sequentiell einlesen müssen. Folglich können wir annehmen, dass die Zertifikate ebenfalls $|c| = O(|x|^d)$ erfüllen.

DEFINITION: Sei $\mathsf{NP} = \{ L \subseteq \Sigma^* \mid$ existiert ein polynomieller Verifier für $L\}$ die Klasse der *nichtdeterministisch polynomiell entscheidbaren* Sprachen.

BEISPIEL: Clique $\in$ NP. *Beweis:* Sei die Eingabe $(G = (V, E),\ k \in \mathbb{N})$. Gibt es eine Clique C auf k Knoten, so fasse diese als Zertifikat auf. Der folgende Verifier entscheidet das Clique-Problem:

```
Prüfe, ob |C| ≥ k
Prüfe, ob {u,v} ∈ E für alle u,v ∈ C
Falls alle Tests positiv, gib 1 aus
```

Die Laufzeit des Verifiers ist $O(n^2)$.

LEMMA: Es gilt $\mathsf{P} \subseteq \mathsf{NP}$

BEWEIS: In der Übung.

2.4 NP-Vollständikgkeit

DEFINITIONEN: Seien L_1, L_2 Sprachen über Alphabeten Σ_1, Σ_2.

1. Eine Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ heißt *Transformation* von L_1 auf L_2, wenn gilt:
$$\forall w \in \Sigma_1^* : w \in L_1 \iff f(w) \in L_2$$
2. Eine solche Transformation heißt *polynomielle Transformation*, wenn es ein Polynom p und einen Algorithmus $A\colon \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $w \in \Sigma_1^*$ gilt:
   - $A(w) = f(w)$
   - $t_A(w) \leq p(|w|)$
   - $|f(w)| \leq p(|w|)$
3. L_1 heißt *polynomiell auf L_2 reduzierbar*, in Zeichen $L_1 \leq_p L_2$, wenn es eine polynomielle Transformation von L_1 auf L_2 gibt.

SATZ:

1. Ist $L_1 \leq_p L_2$ und $L_2 \in \mathsf{P}$, so ist $L_1 \in \mathsf{P}$.
2. Die Relation „$\leq_p$“ ist transitiv: Für alle L_1, L_2, L_3 gilt:
$$(L_1 \leq_p L_2) \wedge (L_2 \leq L_3) \Longrightarrow L_1 \leq_p L_3$$

BEWEIS:

1. Seien f, A, p wie in der Definition. Sei B ein Algorithmus, der L_2 entscheidet und dessen Laufzeit durch ein Polynom q beschränkt ist. O.B.d.A. können wir annehmen, dass q monoton wachsend ist. Folgender Algorithmus entscheidet L_1:

```
Eingabe: w ∈ Σ1*
1. Berechne f(w) mit A
2. Mit B entscheide, ob f(w) ∈ L2, gib Ergebnis zurück
```

   Nach Konstruktion ist dieser Algorithmus korrekt, d.h. er entscheidet L_1. Die Laufzeit ist beschränkt durch

$$p(|w|) + q(|f(w)|) \leq p(|w|) + q(p(|w|))$$

2. In der Übung.

DEFINITIONEN:

1. Eine Sprache L heißt NP-*Hart*, wenn für alle $M \in \mathsf{NP}$ gilt $M \leq_p L$.
2. L heißt NP-*vollständig*, wenn L NP-hart ist und $L \in \mathsf{NP}$.
3. Die Klasse der NP-vollständigen Sprachen bezeichnen wir mit NPC.

KOROLLAR: Gibt es eine Sprache in $\mathsf{P} \cap \mathsf{NPC}$, so gilt $\mathsf{P} = \mathsf{NP}$.

BEMERKUNG: Die Frage, ob $\mathsf{P} = \mathsf{NP}$ oder $\mathsf{P} \subsetneq \mathsf{NP}$ bleibt offen. [2]

PROBLEM **Erfüllbarkeitsproblem** (SAT): Seien $x_1, \ldots, x_n$ Boolsche Variablen. Mit $\bar{x}_i$ bezeichnen wir das Negierte von x_i. Die $x_i, \bar{x}_i$ heißen *Literale*. Eine *Klausel* über $x_1, \ldots, x_n$ ist eine Disjunktion von Literalen (z.B. $x_1 \vee \bar{x}_3 \vee x_4$). Eine Instanz des Erfüllbarkeitsproblems ist eine Menge von Klauseln. Zu entscheiden ist, ob eine Belegung von Variablen mit Wahrheitswerten existiert so, dass alle Klauseln erfüllt sind.

SATZ: (Cook, 1971) SAT ist NP-vollständig.

PROBLEM **3-SAT**: Die Einschränkung des Erfüllbarkeitsproblems auf Instanzen, in denen alle Klauseln nur drei Literale enthalten.

SATZ: $3-\mathrm{SAT}$ ist NP-vollständig.

BEWEIS: In der Übung.

[2] Clay Mathematics Institute bietet einen Preis von 1.000.000$ für die Lösung des Problems - siehe http://www.claymath.org/millennium

SATZ: Clique ist NP-Vollständig.

BEWEIS: Wir haben schon gezeigt: Clique $\in$ NP. Wir zeigen nun: $3-\text{SAT} \leq_p$ Clique. Für alle $L \in$ NP gilt dann: $L \leq_p 3-\text{SAT} \leq_p$ Clique, also ist Clique NP-hart.

Sei also $X = \{x_1, \ldots, x_n\}$ und $\varphi = \{C_1, \ldots, C_m\}$ eine Instanz von $3-\text{SAT}$. Sei $C_i = Z_i^1 \vee Z_i^2 \vee Z_i^3$ mit Literalen $Z_i^j, j \in [3]$. Wir konstruieren daraus folgende Clique-Instanz: Sei $G = (V, E)$ mit $V = [m] \times [3]$ und

$$E = \left\{ \{(i,j),(k,l)\} \mid i \neq k \text{ und } Z_i^j \neq \bar{Z}_k^l \right\}$$

Offenbar ist G in polynomieller Zeit aus der $3-\text{SAT}$ Instanz konstruierbar.

Behauptung: G enthält eine Clique der Größe m genau dann, wenn es eine erfüllende Belegung für die $3-\text{SAT}$ Instanz gibt.

Beweis: Sei H eine Clique der Größe m in G. Nach Konstruktion von G gehören die Knoten von H zu paarweise verschiedenen Klauseln. Die zu den Knoten gehörigen Literale können gleichzeitig erfüllt werden. Damit sind alle Klauseln erfüllt.

Sei nun eine erfüllende Belegung der $3-\text{SAT}$ Instanz gegeben. Für $i \in [m]$ existiert also ein $j_i \in [3]$ mit $Z_i^{j_i}$ wahr. Damit ist $H = \{(i, j_i) \mid i \in [m]\}$ eine Clique. □

DEFINITION: Sei $G = (V, E)$ ein Graph. Eine Kantenfolge $x_1, \ldots, x_k$ heißt *Kreis* in G, falls

- für alle $i \in [k-1]$ gilt: $\{x_i, x_{i+1}\} \in E$
- $\{x_k, x_1\} \in E$
- $|\{\{x_i, x_{i+1}\} \mid i \in [k-1]\} \cup \{\{x_k, x_1\}\}| = k$

Ein *Hamiltonkreis* ist ein Kreis, der alle Knoten genau einmal enthält.

PROBLEM **Hamiltonkreis**. Eingabe: ein Graph $G = (V, E)$. Frage: hat G einen Hamiltonkreis?

PROBLEM **Graphenfärbarkeit**. Eingabe: ein Graph $G = (V, E)$ und $k \in \mathbb{N}$. Frage: besitzt G eine k-Färbung?

SATZ:

- Hamiltonkreis $\in$ NPC
- TSP $\in$ NPC
- Graphenfärbarkeit $\in$ NPC

SATZ: Das Problem „Hat G eine 2-Färbung?“ liegt in P.

3 Suboptimalität

Unter der Annahme $\mathsf{P} \neq \mathsf{NP}$ kann man für viele Optimierungsprobleme keine exakten effizienten (polynomiellen) Algorithmen erwarten. Als Lockerung bieten sich pseudopolynomielle oder approximative Algorithmen an.

3.1 Pseudopolynomialität

Bestimmte Probleme werden „nur“ dadurch schwer, dass Zahlen nur logarithmische Kodierungslänge besitzen. Für ein solches Problem bezeichne $\mathrm{MAX}(I)$ den Wert der größten in der Instanz I vorkommenden Zahl und $L(I)$ die binäre Kodierungslänge.

Beispiel: Für TSP gilt:

$$\mathrm{MAX}(I) = \mathrm{MAX}\{\, d_{ij} \mid i, j \in [n] \}\,; \quad L(I) = O(n^2 \log(\mathrm{MAX}(I)))$$

Bemerkung: Es gilt bei den meisten Problemen

$$\begin{aligned} L(I) &= O(\text{ Anzahl Zahlen } \cdot \log(\mathrm{MAX}(I)) + \text{ Kodierungslänge Rest }) \\ &= O(\max\{\text{ Anzahl Zahlen } \cdot \log(\mathrm{MAX}(I)),\ \text{Kodierungslänge Rest }\}) \end{aligned}$$

Definition: Ein Problem heißt *Zahlproblem*, wenn sich $\mathrm{MAX}(I)$ nicht durch ein Polynom in $L(I)$ beschränken lässt.

Beispiele:

1. Clique ist kein Zahlproblem, da $L(I) = O(n^2 + \log_2 k)$ und $\mathrm{MAX}(I) = k \leq n$.

2. Clique mit Kantengewichten $w_1, \ldots, w_m \in \mathbb{N}$ ist ein Zahlproblem. Es gilt:
$$L(I) = O\left(n^2 + \sum_i \log w_i\right)\,; \quad \mathrm{MAX}(I) = \max_i w_i$$
Ist $w_1 = 2^n$ und $w_i = 1$ für $i \geq 2$, dann ist $L(I) = O(n^2)$, aber $\mathrm{MAX}(I) = 2^n$.

Definition: Ein Algorithmus für ein Zahlproblem heißt *pseudopolynomiell*, wenn seine Laufzeit durch ein Polynom in $L(I)$ und $\mathrm{MAX}(I)$ beschränkt ist. Ein Algorithmus für ein Zahlproblem heißt *polynomiell*, wenn seine Laufzeit durch ein Polynom in $L(I)$ beschränkt ist.

DEFINITION: Für ein Zahlproblem Π und ein Polynom p bezeichne Π_p das Teilproblem von Π, bei dem nur Eingaben I zugelassen sind, für die $\mathrm{MAX}(I) \leq p(L(I))$ ist. Ein Zahlproblem Π heißt *stark* NP*-vollständig*, wenn es ein Polynom p gibt so, dass $\Pi_p \in \mathsf{NPC}$ gilt.

SATZ: Ist Π stark NP-vollständig, so gibt es keine pseudopolynomiellen Algorithmen für Π, es sei denn $\mathsf{P} = \mathsf{NP}$.

BEWEIS: Sei p ein Polynom, so dass $\Pi_p \in \mathsf{NPC}$. Angenommen, Π besitzt einen pseudopolynomiellen Algorithmus mit Laufzeitbeschränkung $q(L(I), \mathrm{MAX}(I))$ für ein Polynom q. O.B.d.A. sei q monoton wachsend in der zweiten Komponente. Für Instanzen I von Π_p gilt $\mathrm{MAX}(I) \leq p(L(I))$. Also ist die Laufzeit des Algorithmus für solche Instanzen beschränkt durch

$$q(L(I), \mathrm{MAX}(I)) \leq q(L(I), p(L(I)))$$

also durch ein Polynom in $L(I)$. Da $\Pi_p \in \mathsf{NPC}$, folgt $\mathsf{P} = \mathsf{NP}$. □

PROBLEM **Rücksack (RSP)**. Eingabe: Nutzerwerte $a_1, \ldots, a_k \in \mathbb{N}_0$, Gewichte $g_1, \ldots, g_k \in \mathbb{N}_0$, Grenzgewicht $G \in \mathbb{N}_0$ und Mindestwert $A \in \mathbb{N}_0$. Frage: gibt es $M \subseteq [k]$ mit

$$\sum_{i \in M} g_i \leq G \text{ und } \sum_{i \in M} a_i \geq A$$

SATZ: RSP $\in \mathsf{NPC}$.

BEWEIS: Offenbar ist RSP $\in \mathsf{NP}$. Wir zeigen: $3-\mathrm{SAT} \leq_p \mathrm{RSP}$. Sei $I = (X, \mathcal{C})$ mit $X = \{x_1, \ldots, , x_n\}$ und $\mathcal{C} = \{C_1, \ldots, C_m\}$ eine Instanz von $3-\mathrm{SAT}$. Wir konstruieren daraus eine RSP-Instanz I' wie folgt: Sei A die Zahl mit Dezimaldarstellung $(4)^m(1)^n$ [3] und $G = A$. Wir benutzen $2n + 2m$ Objekte mit Nutzen $a_i, b_i, i \in [n]$ und $c_j, d_j, j \in [m]$. Diese Werte sind alle Dezimalzahlen mit $n + m$ Stellen:

- In a_i im vorderen Block mit m Stellen an Position j steht, wie oft das Literal x_i in C_j vorkommt. Im hinteren Block (der Länge n) steht an Position i eine 1. An allen übrigen Positionen stehen Nullen.
- Sei b_i wie a_i mit dem Literal $\bar{x}_i$ statt x_i.
- In c_j im vorderen Block der Länge m steht an Position j eine 1, der Rest sind Nullen.

[3] Mit $(a)^n$ bezeichnen wir das Wort $(a \ldots a)$, wo der Buchstabe a n-mal vorkommt.

- Sei $d_j = 2c_j$

Für alle Objekte soll das Gewicht gleich dem Nutzen sein. Wir zeigen nun: I ist erfüllbar genau dann, wenn I' eine Lösung besitzt.

„⇒“ Sei I erfüllbar. Sei $x_1^*, \ldots, x_n^*$ eine erfüllende Belegung. Ist x_i^* Wahr, so nimm das Objekt mit Nutzen a_i in die Lösung auf, sonst das mit Nutzen b_i. Damit haben wir schon mal einen Nutzen von $\ldots(1)^n$ erreicht. Da die x_i^* eine erfüllende Belegung bilden, haben wir an allen Stellen im vorderen Block mindestens eine 1. Durch geeignete Wahl von Objekten mit Nutzen c_j, d_j erhalten wir einen Gesamtnutzen von A.

„⇐“ Sei nun die RSP-Instanz I' lösbar. Fixiere eine Lösung. Nach Konstruktion hat sie den Gesamtnutzen A. Daher ist für alle $i \in [n]$ genau eines der beiden Objekte mit Nutzen a_i oder b_i in der Lösung. Setze x_i^* wahr, falls das Objekt mit Nutzen a_i in der Lösung ist. Diese Belegung erfüllt I.

SATZ: Es gibt einen pseudopolynomiellen Algorithmus für RSP, genauer, einen der in Laufzeit $O(n \cdot \sum_{i=1}^{n} a_i)$ den maximalen Nutzen einer Lösung ermittelt, die die Gewichtsrestriktion einhält.

BEWEIS: Sei $I = (a_1, \ldots, a_n, g_1, \ldots, g_n, G)$ eine Instanz der Maximiervariante von RSP. Für $T \subseteq [n]$ setze $a(T) = \sum_{i \in T} a_i$ und $g(T) = \sum_{i \in T} g_i$. Gesucht ist also

$$\mathrm{OPT} = \max\{\, a(M) \mid M \subseteq [n],\ g(M) \leq G\}$$

Setze $m = \sum_{i=1}^{n} a_i$. Für $\alpha \in [m]$ und $k \in [n]$ setze

$$\begin{aligned} S(k,\alpha) &= \{T \subseteq [k] \mid a(T) = \alpha\} \\ G(k,\alpha) &= \min\{\, g(T) \mid T \in S(k,\alpha)\} \end{aligned}$$

Dann gilt

$$S(k,\alpha) = S(k-1,\alpha) \cup \{T \cup \{k\} \mid T \in S(k-1, \alpha - a_k)\}$$

Also gilt

$$\begin{aligned} G(k,\alpha) &= \min\{\, S(T) \mid T \in S(k,\alpha)\} \\ &= \min\{\min\{\, g(T) \mid T \in S(k-1,\alpha)\}, \\ &\qquad \min\{\, g(T \cup \{k\}) \mid T \in S(k-1, \alpha - a_k)\}\} \\ &= \min\{G(k-1,\alpha), G(k-1,\alpha-a_k) + g_k\} \end{aligned}$$

Folgender Algorithmus löst das RSP:

1. Für alle $\alpha \in [m]$ setze $G(0,\alpha) = \infty$
   Für alle $k \in [n] \cup \{0\}$ setze $G(k,0) = 0$
2. Für alle $k \in [n]$
   Für alle $\alpha \in [m]$
   setze $G(k,\alpha) = \min\{G(k-1,\alpha), G(k-1,\alpha-a_k) + g_k\}$
3. Ausgabe: $\mathrm{OPT} = \max\{\alpha \in [m] \cup \{0\} \mid G(n,\alpha) \leq G\}$

Dabei sollen im Schritt 2 nicht definierte Werte als ∞ gelten.

Offenbar ist der Algorithmus korrekt. Der Aufwand vom 1. Schritt ist $O(n+m)$, vom 2. Schritt $O(nm)$ und vom 3. Schritt $O(m)$. Somit hat der Algorithmus die Laufzeit $O(nm) = O(n^2 \,\mathrm{MAX}(I))$.[4] □

PROBLEM **Partition**. Eingabe: $a_1, \ldots, a_n \in \mathbb{N}$. Frage: Gibt es $M \subseteq [n]$ mit

$$\sum_{i \in M} a_i = \sum_{i \in [n] \setminus M} a_i$$

SATZ: Partition $\in$ NPC

BEWEIS: Offenbar ist Partition $\in$ NP (mit M als Zertifikat). Sei nun RSP* dasjenige Teilproblem von RSP, bei dem lediglich entschieden wird, ob ein M existiert mit $\sum_{i \in M} a_i = A$. Nach Beweis für RSP ist auch RSP$^* \in$ NPC. Wir zeigen RSP$^* \leq_p$ Partition. Sei $I = (a_1, \ldots, a_n, A)$ eine Instanz von RSP*. Betrachte die Partition-Instanz

$$I' = (a_1, \ldots, a_n, S - A + 1, A + 1) \text{ mit } S = \sum_{i=1}^{n} a_i$$

Sei M eine Lösung für I, dann ist $M \cup \{n+1\}$ eine Lösung für I', denn

$$\begin{aligned} \sum_{i \in M} a_i + S - A + 1 &= S + 1 = \sum_{i=1}^{n} a_i + 1 \\ &= \sum_{i \in M} + \sum_{i \in [n] \setminus M} a_i + 1 = \sum_{i \in [n] \setminus M} a_i + (A+1) \end{aligned}$$

Sei umgekehrt M eine Lösung von I'. Dann ist entweder $n+1 \in M$ oder $n+2 \in M$. O.B.d.A. sei $n+1 \in M$. Sei $x = \sum_{i \in M \setminus \{n+1\}} a_i$. Dann gilt

$$x + S - A + 1 = \sum_{i \in M} a_i = \sum_{i \in [n+1] \setminus M} a_i = S - x + A + 1$$

Also ist $x = A$ und $M \setminus \{n+1\}$ eine Lösung von I. □

[4] Die hier verwendete Technik heißt *dynamische Programmierung*.

SATZ: Es gibt einen pseudopolynomiellen Algorithmus für Partition.

BEWEIS: In der Übung.

PROBLEM **Binpacking**. Eingabe: $a_1, \ldots, a_n, b, k \in \mathbb{N}$. Frage: gibt es eine Partition $[n] = I_1 \dot\cup \ldots \dot\cup I_k$, so dass für alle $j \in [k]$ gilt $\sum_{i \in I_j} \leq b$ ist.

SATZ: Binpacking $\in$ NPC

BEWEIS: In der Übung.

3.2 Approximationsalgorithmen

Approximationsalgorithmen finden i.a. keine optimale Lösung, sondern nur eine, die relativ gut ist. Wir betrachten endliche Optimierungsprobleme mit Lösungen in $\mathbb{N}$.

DEFINITIONEN:

- Ein *Optimierungsproblem* Π ist ein Tupel $\Pi = (D, S, \omega)$ mit:
  - D ist die Menge der zulässigen Eingaben.
  - Für $I \in D$ ist $S(I)$ eine Menge von zulässigen Lösungen.
  - Für $I \in D$ und $s \in S(I)$ ist $\omega(I, s) \in \mathbb{N}$ der Wert der Lösung s.

  Zusätzlich gehört zu dem Problem eine implizite Angabe, ob ω zu maximieren oder zu minimieren ist. Ist $I \in D$, so schreiben wir lax $I \in \Pi$ und $\mathrm{OPT}(I) = \max / \min \{\omega(I, s) \mid s \in S(I)\}$.
- Ein *Approximationsalgorithmus* A für ein Optimierungsproblem Π hat als Eingabe eine Instanz $I \in \Pi$ und als Ausgabe eine Zahl $A(I)$ mit $A(I) = \omega(I, s)$ für ein $s \in S(I)$.

DEFINITION: Seien A, Π wie eben. Für $I \in \Pi$ ist die *Güte* $R_A(I)$ definiert durch

$$R_A(I) = \begin{cases} \frac{A(I)}{\mathrm{OPT}(I)} & \text{falls } \Pi \text{ ein Minimierungsproblem ist} \\ \frac{\mathrm{OPT}(I)}{A(I)} & \text{falls } \Pi \text{ ein Maximierungsproblem ist} \end{cases}$$

Die *Worst-Case-Güte* von A ist $R_A = \sup_{I \in \Pi} R_A(I)$.

BEMERKUNG: Offenbar gilt stets $R_A(I) \geq 1$.

DEFINITION: Man sagt, A hat einen *additiven Fehler von höchstens* ε, falls für alle $I \in \Pi$ gilt

$$|\operatorname{OPT}(I) - A(I)| \leq \varepsilon$$

SATZ: Unter der Annahme $\mathsf{P} \neq \mathsf{NP}$ gibt es für die Optimierungsversion von RSP keinen polynomiellen Approximationsalgorithmus mit additivem Fehler.

BEWEIS: Angenommen, es gibt einen Approximationsalgorithmus mit additivem Fehler c. Sei $I = (a_1, \ldots, a_n, g_1, \ldots, g_n, G)$ eine Instanz von RSP. Betrachte $I^* = (a_1^*, \ldots, a_n^*, g_1, \ldots, g_n, G)$ mit $a_i^* = (c+1)a_i$ für alle $i \in [n]$. Dann ist $\operatorname{OPT}(I^*) = (c+1)\operatorname{OPT}(I)$, da I und I^* dieselben zulässigen Lösungen besitzen. Sei M eine Lösung von I^* mit $\omega(I^*, M) = A(I^*)$. Dann gilt

$$c \geq |\operatorname{OPT}(I^*) - A(I^*)| = |\operatorname{OPT}(I^*) - \omega(I^*, M)| = (c+1)\,|\operatorname{OPT}(I) - \omega(I, M)|$$

Also gilt

$$|\operatorname{OPT}(I) - \omega(I, M)| \leq \frac{c}{c+1} < 1$$

Da $\operatorname{OPT}(I), \omega(I, M) \in \mathbb{N}$, folgt $\operatorname{OPT}(I) = \omega(I, M)$. Also ist M optimale Lösung der Instanz I. Folglich ist „Berechne I^* und löse sie mit A“ ein exacter polynomieller Algorithmus für RSP. Somit ist $\mathsf{P} = \mathsf{NP}$, da RSP $\in \mathsf{NPC}$. □

DEFINITIONEN:

- Ein *vollständig polynomielles Approximationsschema (FP(T)AS)* für ein Optimierungsproblem Π ist ein Algorithmus A, der zu jedem $I \in \Pi$ und $\varepsilon > 0$ eine Lösung mit Güte $R_A(I) \leq 1 + \varepsilon$ berechnet in Laufzeit

  $$t_A(I, \varepsilon) \leq p\left(|I|, \varepsilon^{-1}\right)$$

  für ein Polynom p.

- A ist ein *polynomielles Approximationsschema*, wenn $t_A(I, \varepsilon) \leq p_\varepsilon(|I|)$ ist, wobei p_ε für $\varepsilon > 0$ ein Polynom ist.

BEMERKUNG: Formal handelt es sich um eine Klasse $(A_\varepsilon)_{\varepsilon > 0}$ von Algorithmen, d.h. ε ist kein Teil der Eingabe.

SATZ: Sei q ein Polynom in zwei Variablen. Sei Π ein stark NP-vollständiges Optimierungsproblem[5] mit $\operatorname{OPT}(I) \leq q(L(I), \operatorname{MAX}(I))$ für alle $I \in \Pi$. Dann besitzt Π kein FPAS, es sei denn $\mathsf{P} = \mathsf{NP}$.

BEWEIS: In der Übung.

SATZ: (Ibarra, Kim, 1976). Für das RSP existiert ein FPAS.

BEWEIS: Wir skalieren die Zahlen geeignet herunter und wenden den pseudopolynomiellen Algorithmus an. Sei $I = (a_1, \ldots, a_n, g_1, \ldots, g_n, G)$ eine Instanz der Optimierungsversion von RSP. Sei $\varepsilon > 0$ gegeben. O.B.d.A. sei $g_i \leq G$ für alle $i \in [n]$. Sei $I' = (a'_1, \ldots, a'_n, g_1, \ldots, g_n, G)$ eine neue Instanz mit

$$a'_i = \left\lfloor \frac{a_i}{b} \right\rfloor, \quad \text{wobei} \quad b = \frac{\varepsilon a_{\max}}{n}, \quad a_{\max} = \max\{\, a_i \mid i \in [n]\}$$

Nach Konstruktion besitzen I und I' dieselben Lösungen. Sei M eine Solche Lösung. Dann gilt

$$\omega(I', M) \leq \frac{1}{b}\omega(I, M) < \omega(I', M) + |M| \leq \omega(I', M) + n \tag{1}$$

Sind M und M' optimale Lösungen von I bzw. I', so folgt

$$\mathrm{OPT}(I) = \omega(I, M) < b\omega(I', M) + bn \leq b\omega(I', M') + bn = b\,\mathrm{OPT}(I') + bn \tag{2}$$

Wir können annehmen, dass $\omega(I, M') \geq a_{\max}$ ist: Entweder ist $\omega(I', M') > \frac{a_{\max}}{b}$, dann ist $\omega(I, M') > a_{\max}$, oder es ist $\omega(I', M') = \left\lfloor \frac{a_{\max}}{b} \right\rfloor$ und wir können annehmen, dass M' genau das Objekt mit Nutzen $a_{\max}$ enthält. Es folgt

$$\frac{\mathrm{OPT}(I)}{\omega(I, M')} \overset{(2)}{\leq} \frac{b\,\mathrm{OPT}(I')}{\omega(I, M')} + \frac{bn}{\omega(I, M')} \overset{(1)}{\leq} \frac{b\,\mathrm{OPT}(I')}{b\omega(I', M')} + \frac{bn}{a_{\max}} = 1 + \varepsilon$$

Also liefert der Algorithmus „Löse I' statt I mit dem pseudopolynomiellen Algorithmus“ eine Lösung der Güte $1 + \varepsilon$. Die Laufzeit ist

$$O\left(n \cdot \frac{\sum a'_i}{b}\right) = O(n^3 \varepsilon^{-1})$$

SATZ: (Korte, Schrader, 1981). NP-vollständige Probleme, für die es ein FPAS gibt, sind zu RSP (im geeigneten Sinne) äquivalent.

PROBLEM **Euklidisches TSP**. Eingabe: n Punkte im $\mathbb{R}^2$. Aufgabe: finde eine Tour minimaler Länge durch alle Punkte.

SATZ: (Arora, 1999) Für alle $c > 1$ existiert ein $(1 + c^{-1})$-Approximationsalgorithmus für das euklidische TSP mit der Laufzeit $O(n^3 (\log n)^{O(c)})$

[5]Unter einem NP-vollständigem Optimierungsproblem verstehen wir ein Problem, bei dem die Entscheidungsvariante NP-vollständig ist.

3.3 Approximierbarkeit von MaxCut und MaxSat

DEFINITION: Sei $G = (V, E)$ ein Graph. Für $S \subseteq V$ sei der von S induzierte *Schnitt* $\delta(S) = \{e \in E \mid |e \cap S| = 1\}$

PROBLEM **MaxCut**. Eingabe: Graph $G = (V, E), k \in \mathbb{N}$. Frage: gibt es ein $S \subseteq V$ mit $|\delta(S)| \geq k$.

DEFINITION: Sei $V = [n]$. Für $S \leq V$ sei $x_S \in \{0,1\}^n$ mit $(x_S)_i = 1$ genau dann, wenn $i \in S$. Sei A die Adjazenzmatrix von G. Für $x \in [0,1]^n$ sei

$$C(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i (1 - x_j) = x^T A (1 - x)$$

BEMERKUNGEN:

1. Für alle $S \subseteq [n]$ ist $C(x_S) = |\delta(S)|$.
2. Es gilt $C\left(\frac{1}{2}, \ldots, \frac{1}{2}\right) = \frac{1}{2}|E|$.

Algorithmus GreedyCut

1. Wähle $x_1 \in \{0,1\}$ so, dass
   $C\left(x_1, \frac{1}{2}, \ldots, \frac{1}{2}\right) = \max\left\{C\left(0, \frac{1}{2}, \ldots, \frac{1}{2}\right), C\left(1, \frac{1}{2}, \ldots, \frac{1}{2}\right)\right\}$
2. Wähle $x_2 \in \{0,1\}$ so, dass
   $C\left(x_1, x_2, \frac{1}{2}, \ldots, \frac{1}{2}\right) = \max\left\{C\left(x_1, 0, \frac{1}{2}, \ldots, \frac{1}{2}\right), C\left(x_1, 1, \frac{1}{2}, \ldots, \frac{1}{2}\right)\right\}$

   $\vdots$

n. Wähle $x_n \in \{0,1\}$ so, dass
   $C(x_1, \ldots, x_{n-1}, x_n) = \max\{C(x_1, \ldots, x_{n-1}, 0), C(x_1, \ldots, x_{n-1}, 1)\}$

Ausgabe: $x_1, \ldots, x_n$ oder $S = \{i \in [n] \mid x_i = 1\}$

SATZ: (Sahni, Gonzales, 1976). GreedyCut berechnet ein $S \subseteq [n]$ mit $|\delta(S)| \geq \frac{1}{2}|E| \geq \frac{1}{2}\,\mathrm{OPT}$ in Zeit $O(n^3)$.

BEWEIS: Die Funktion $C\colon \mathbb{R}^n \to \mathbb{R}$ ist linear in jeder Komponente, da alle $a_{ii} = 0$ sind. Also gilt für alle $i \in [n]$ und $x_1, \ldots, x_n \in [0,1]$:

$$\begin{aligned} & C(x_1, \ldots, x_{i-1}, 1/2, x_{i+1}, \ldots, x_n) \\ = \; & \frac{1}{2} C(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + \frac{1}{2} C(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \\ \leq \; & \max\{C(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n), C(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)\} \end{aligned}$$

Sei x^* die Lösung von GreedyCut. Dann gilt

$$\frac{1}{2}\,\mathrm{OPT}(I) \leq \frac{1}{2}|E| = C\left(\frac{1}{2},\ldots,\frac{1}{2}\right)$$
$$\leq C\left(x_1^*,\frac{1}{2},\ldots,\frac{1}{2}\right) \leq \ldots \leq C\left(x_1^*,\ldots,x_n^*\right) = |\delta(S)|$$

□

BEMERKUNG: Das Verfahren GreedyCut funktioniert auch mit jedem anderen Startpunkt $p \in [0,1]^n$ statt $\left(\frac{1}{2},\ldots,\frac{1}{2}\right)$ und liefert dann einen Schnitt mit Wert $C(p)$.

Randomizierter Algorithmus für MaxCut: Nimm jeden Knoten mit Wahrscheinlichkeit $\frac{1}{2}$ (unabhängig) in S auf. Für die einzelnen Zufallsvariablen gilt dann

$$E(X_e) = 0 \cdot P(X_e = 0) + 1 \cdot P(X_e = \frac{1}{2}) = \frac{1}{2}$$

Somit gilt für den Erwartungswert

$$E(|\delta(S)|) = E\left(\sum_{e\in E} X_e\right) = \sum_{e\in E} E\left(X_e\right) = \frac{1}{2}|E|$$

SATZ: Es gibt einen Greedy-Algorithmus, der MaxSat (Optimierungsversion von SAT) mit Güte 2 in Zeit $O(n^2m)$ löst.

BEWEIS: In der Übung.

3.4 Nichtapproximierbarkeit

Zur Nichtapproximierbarkeit gibt es in der Regel zwei Beweismethoden:

- Einfache Reduktion auf NP-vollständige Probleme.
- PCP-Theorie.

Wir werden hier die erste Methode benutzen.

DEFINITION: Sei Π ein Optimierungsproblem. Sei (unter Annahme $\mathsf{P} \neq \mathsf{NP}$)

$$\mathrm{AT}(\Pi) = \inf\{r > 1 \mid \text{existiert ein polynomieller Approximationsalgorithmus der Güte } r \text{ für } \Pi\}$$

SATZ:

- AT(RSP) = 1
- AT(Graphenfärbarkeit) = $\frac{4}{3}$
- AT(Binpacking) = $\frac{3}{2}$
- AT(TSP) = ∞

BEWEIS: Wir beweisen die Aussage für TSP.[6] Sei $\varepsilon > 0$. Angenommen, es existiert ein Approximationsalgorithmus A für TSP mit Güte $1+\varepsilon$. Sei $n_0 \in \mathbb{N}$ mit $2^{n_0} > \varepsilon$. Sei $I = ([n], E)$ eine Eingabe für Hamiltonkreis mit $n \geq n_0$. Betrachte die TSP-Insanz I' mit

$$c_{ij} = \begin{cases} 1 & \text{falls } \{i,j\} \in E \\ n \cdot 2^n + 1 & \text{sonst} \end{cases}$$

Ist $A(I') \geq n + n \cdot 2^n$, so besitzt G keinen Hamiltonkreis, da sonst $\mathrm{OPT}(I') = n$ und

$$\frac{A(I')}{\mathrm{OPT}(I')} \geq 1 + 2^n > 1 + \varepsilon$$

Ist $A(I') < n + n \cdot 2^n$, so kann die Lösung S mit $\omega(I', S) = A(I')$ keine Kante mit Gewicht $n \cdot 2^n + 1$ enthalten. Also ist S ein Hamiltonkreis in I. Folglich entscheidet folgender Algorithmus das Problem Hamiltonkreis:

Eingabe: $G = ([n], E)$

1. Ist $n < n_0$, so überprüfe alle $n!$ Hamiltonkreise und gib die entsprechende Antwort aus.
2. Anderenfalls wende den Approximationsalgorithmus auf die zugehörige TSP-Instanz G' an. Ist dessen Lösung mindestens $n + n \cdot 2^n$, dann Ausgabe „Nein“, sonst „Ja“.

Dieser Algorithmus ist polynomiell und entscheidet das Hamiltonproblem. Da Hamiltonkreis $\in$ NPC, folgt P = NP.

[6] Beweisidee zu Graphenfärbarkeit: Es ist NP-hart, zu entscheiden, ob ein Graph 3-färbbar ist.

4 Bäume und Wege in Graphen

4.1 Minimale aufspannende Bäume

DEFINITIONEN: Sei $G = (V, E)$ ein Graph.

- Eine Knotenfolge $v_1, \ldots, v_k$ heißt *Weg der Länge* $k-1$, falls $\{v_i, v_{i+1}\}$ eine Kante ist für alle $i \in [k-1]$ (die Länge 0 ist zugelassen).
- Der Weg heißt *geschlossen*, falls $k \geq 3$ und $\{v_1, v_k\} \in E$
- Ein Weg $v_1, \ldots, v_k$ der Länge k heißt *Kreis*, falls er geschlossen ist und alle seine Kanten $\{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}, \{v_k, v_1\}$ paarweise verschieden sind.
- Ein Weg $v_1, \ldots, v_k$ heißt *Pfad*, falls $v_i \neq v_j$ für alle $i, j \in [k]$ mit $i \neq j$ gilt.
- Ein Graph heißt *zusammenhängend*, falls je zwei Knoten durch einen Weg verbunden werden können.
- Eine *Zusammenhangskomponente* ist ein maximaler zusammenhängender Teilgraph.
- Ein zusammenhängender kreisfreier Graph heißt *Baum*.
- Ein kreisfreier Graph heißt *Wald*.
- Ist $H = (V_0, E_0)$ ein Teilgraph von G und $e \in E$ mit $e \subseteq V_0$, so setze $H + e = (V_0, E_0 \cup \{e\})$

LEMMA: Sei $G = (V, E)$ ein Graph. Folgende Aussagen sind äquivalent:

1. G ist ein Baum.
2. Zwischen je zwei Knoten von G existiert genau ein Pfad.
3. G ist zusammenhängend und $|E| = |V| - 1$.

BEWEIS: In der Übung.

DEFINITION: Sei $G = (V, E)$ ein zusammenhängender Graph mit Kantengewichten $c\colon E \to \mathbb{Q}_0^+$. Für $F \subseteq E$ sei $c(F) = \sum_{e \in F} c(e)$. Ein *minimal spannender Baum* von G ist ein Graph $T = (V, E_T)$ mit minimalem Gewicht $c(E_T)$.

Algorithmus MST (Minimal Spanning Tree)(Kruskal, 1956)

```
// Eingabe: G = (V, E), c: E → ℚ.
// Ausgabe: ein minimal spannender Baum T = (V, E_T).
1. Sortiere die Kanten nach Gewicht:
       E = {e_1, ..., e_m} mit c(e_1) ≤ c(e_{i+1}) für alle i ∈ [m − 1]
   E_T = ∅
2. Für alle i ∈ [m]
       Wenn (V, E_T) + e_i kreisfrei, dann E_T = E_T ∪ {e_i}
3. Ausgabe: (V, E_T)
```

SATZ: Der Algorithmus MST findet einen minimal spannenden Baum in Zeit $O(mn)$.

BEWEIS: Sei $T = (V, E_T)$ die Ausgabe des Algorithmus. Angenommen, T ist nicht zusammenhängend. Dann existieren zwei Zusammenhangskomponenten und eine Kante e, die diese verbindet. Da $T + e$ kreisfrei ist, hätte e vom Algorithmus in E_T aufgenommen werden müssen, ein Widerspruch. Offenbar ist T kreisfrei, also ein Baum.

Angenommen, es existiert ein minimal spannender Baum $T^* = (V, E_{T^*})$ und $c(E_{T^*}) < c(E_T)$. Sei $E_T = \{f_1, \ldots, f_{n-1}\}$, wobei f_i die Kante ist, die als i-te zu E_T hinzugefügt wurde. Sei $k \in [n-1]$ minimal mit $f_k \notin E_{T^*}$. Sei T^* so gewählt, dass k maximal ist. $T^* + f_k$ besitzt einen Kreis C. Dieser enthält eine Kante g mit $g \notin E_T$. Dann ist $T' = T^* + f_k - g$ ein Baum. Da $f_k \in E_T$ und nicht $g \in E_T$, wurde f_k vor g betrachtet, also $c(f_k) \leq c(g)$. Damit ist

$$c(E_{T'}) = c(E_{T^*}) + c(f_k) - c(g) \leq c(E_{T^*})$$

Damit ist T' ein minimal spannender Baum. T' enthält die Kanten $f_1, \ldots, f_k$. Damit hat die „erste“ Kante aus T' ohne T einen Index größer als k im Widerspruch zur Wahl von T^*.

Sortieren der Kanten erfordert die Zeit $O(m \log(m)) \leq O(m \log(n)) \leq O(mn)$. Der Test auf Kreisfreiheit kann in Zeit $O(n)$ durchgeführt werden, also braucht der Schritt (2.) die Zeit $O(mn)$. □

BEMERKUNG: Es gibt Algorithmen, die in Zeit $O(m \log(n))$ einen minimal spannenden Baum berechnen.

DEFINITION: Eine *Eulertour* ist eine Enumeration $e_1, \ldots, e_m$ von E, so dass $|e_i \cap e_{i+1}| = 1$ für alle $i \in [m-1]$ und $|e_m \cap e_1| = 1$.

SATZ: (Euler) Ein Graph besitzt eine Eulertour genau dann, wenn alle Knoten geraden Grad haben.

BEWEIS: In der Übung.

BEMERKUNG: Der Satz gilt auch für Multigraphen (d.h. Graphen, in den mehrfache Kanten zwischen je zwei Knoten erlaubt sind)

Minimum Spanning Tree Heuristik für TSP mit Dreiecksungleichung. [7]

```
// Eingabe: Graph G
1. Finde einen minimal spannenden Baum T von G
      (Gesamtgewicht ≤ Länge der optimalen Tour)
2. Verdopple alle Kanten in T
      (Gesamtgewicht ≤ 2×(Länge der optimalen Tour))
3. Finde eine Eulertour auf T
4. Durch abkürzen mache daraus eine Hamiltontour.
```

SATZ: Die MST-Heuristik ist eine 2-Approximation für das TSP mit Dreiecksungleichung.

4.2 Matroide

DEFINITION: Ein *Unabhängigkeitssystem* ist ein Paar (E, S), wobei E eine Menge und $S \subseteq \mathcal{P}(E)$ mit $S \neq \emptyset$ ist, mit der Eigenschaft

$$\forall A, B\colon A \in S \wedge B \subseteq A \Longrightarrow B \in S$$

Die Elemente von S heißen *unabhängige Mengen*. In der Vorlesung betrachten wir nur endliche Unabhängigkeitssysteme.

BEISPIEL: Ist V ein Vektorraum und S die Menge der linear unabhängigen Mengen in V, so ist (V, S) ein Unabhängigkeitssystem.

PROBLEM: Zu einer Gewichtsfunktion $\omega\colon E \to \mathbb{Q}_0^+$ finde $A \in S$ mit $\omega(A) = \sum_{e \in A} \omega(e)$ maximal.

Greedy-Algorithmus

```
// Eingabe: (E,S), ω
// Ausgabe: A ∈ S mit ω(A) "groß"
1. A = ∅
2. Sortiere E nach Gewicht: ω(e1) ≥ ω(e2) ≥ ...
3. Füge der Reihe nach die e1 zu A hinzu, falls A ∪ e1 ∈ S
```

DEFINITION: Ein Unabhängigkeitssystem heißt *Matroid*, falls der Greedy-Algorithmus die zugehörigen Optimierungsprobleme löst.

[7] Dreiecksungleichung: $c(\{v_1, v_2\}) + c(\{v_2, v_3\}) \geq c(\{v_1, v_3\})$

SATZ: sei $G = (V, E)$ Ein Graph. Sei S die Menge der kreisfreien Teilmengen von E. Dann ist das Paar (E, S) ein Matroid.

BEWEIS: Der Kruskal-Algorithmus ist ein Greedy-Algorithmus für das Minimierungsproblem im Unahbängigkeitssystem (E, S).

SATZ: Sei $M = (E, S)$ ein Unabhängigkeitssystem. Dann sind äquivalent:

1. M ist ein Matroid.
2. Für $J, K \in S$ mit $|J| = |K| + 1$ gibt es ein $a \in J \setminus K$ mit $K \cup \{a\} \in S$.
3. Für alle $E_0 \in E$ haben alle maximalen unabhängigen Teilmengen von E_0 dieselbe Mächtigkeit.

BEWEIS:

(1)⇒(2) (Kontraposition) Es gelte (2) nicht. Dann existieren $J, K \in S$ mit $|J| = |K| + 1$, so dass für jedes $a \in J$ gilt $K \cup \{a\} \notin S$. Setze $k = |K|$. Definiere $\omega \colon E \to \mathbb{N}_0$ durch

$$\omega(e) = \begin{cases} k+2 & \text{falls } e \in K \\ k+1 & \text{falls } e \in J \setminus K \\ 0 & \text{sonst} \end{cases}$$

Dann findet der Greedy-Algorithmus K als Lösung mit Wert $\omega(K) = k(k+2)$. Eine optimale Lösung ist aber J mit $\omega(J) \geq (k+1)^2 > \omega(K)$. Damit löst der Greedy-Algorithmus das Optimierungsproblem nicht, d.h. M ist kein Matroid.

(2)⇒(3) Sei $E_0 \subseteq E$. Seien J, K maximale unabhängige Teilmengen von E_0. Angenommen $|J| > |K|$. Sei $J' \subseteq J$ mit $|J'| = |K| + 1$. Nach (2) existiert $a \in J' \setminus K$ mit $K \cup \{a\} \in S$. Dies ist ein Widerspruch zur Maximalität von K, da $K \cup \{a\}$ eine unabhängige Teilmenge von E_0 ist und $K \subsetneq K \cup \{a\}$.

(3)⇒(1) Angenommen, M ist kein Matroid, aber erfüllt (3). Dann existiert $\omega \colon E \to \mathbb{Q}_0^+$, so dass der Greedy-Algorithmus keine optimale Lösung findet. Sei $K = \{e_1, \ldots, e_k\}$ die Lösung des Greedy-Algorithmus und $J = \{e'_1, \ldots, e'_h\}$ eine unabhängige Menge größeren Gewichts. O.B.d.A. seien die e_i, e'_i der Größe nach geordnet und J maximale unabhängige Teilmenge. Nach Konstruktion ist K ebenfalls eine maximale unabhängige Teilmenge, also $k = h$.

Wir zeigen $\omega(e_i) \geq \omega(e'_i)$ für alle $i \in [k]$. Für $i = 1$ ist das trivial. Gelte also $\omega(e_i) \geq \omega(e'_i)$ für alle $i \in [k'-1]$ für ein $k' \leq k$. Angenommen, $\omega(e_{k'}) < \omega(e'_{k'})$. Setze $E_0 = \{e \in E \mid \omega(e) \geq \omega(e'_{k'})\}$. Ist $e \in E$ mit $\{e_1, \ldots, e_{k'-1}, e\}$ unabhängig, dann ist $\omega(e) \leq \omega(e_{k'}) < \omega(e'_{k'})$. Also $e \notin E_0$ und $\{e_1, \ldots, e_{k'-1}\}$ ist eine maximal unabhängige Teilmenge von E_0. Da $\{e'_1, \ldots, e'_{k'}\}$ auch eine unabhängige Teilmenge von E_0 ist, ergibt sich ein Widerspruch zu (3).

4.3 Kürzeste Wege in Graphen

Sei $G = (V, E)$ ein zusammenhängender Graph mit nicht-negativen Kantengewichten $l\colon E \to \mathbb{Q}_0^+$.

DEFINITION: Für einen Weg $P = (v_1, \ldots, v_k)$ setze $l(P) = \sum_{i=1}^{k-1} l(\{v_i, v_{i+1}\})$. Setze

$$d(u, v) = \min\{l(P) \mid P \text{ ist ein Weg von } u \text{ nach } v\}$$

für alle $u, v \in V$. Für $v \in V$ setze

$$N(v) = \{u \in V \mid \{u, v\} \in E\}$$

BEMERKUNG: d ist eine Halbmetrik, d.h. für alle $u, v, w \in V$ gilt:

- $d(v, v) = 0$
- $d(u, v) = d(v, u)$
- $d(u, w) \leq d(u, v) + d(v, w)$

PROBLEM **Kürzeste Wege:** Eingabe: $G = (V, E)$, Startknoten $u_0 \in V$ und $l\colon E \to \mathbb{Q}_0^+$. Ausgabe: Für alle $v \in V$ ein kürzester Weg (bzw. dessen Länge) von u_0 nach v. Alternativ: Ein Baum T mit der Eigenschaft, dass für alle $v \in V$ der eindeutig bestimmte u_0-v-Weg in T ein kürzester Weg in G ist.

Algorithmus Dijkstra:

```
// Eingabe: G = (V,E), Startknoten v_0 ∈ V, l: E → Q_0^+
// Ausgabe: Baum T = (V,E_0), so dass
//    für alle v∈V der eindeutig bestimmte
//    v_0–v–Weg in T ein kürzester Weg in G ist.
1. V_0 = {v_0}, E_0 = ∅, l(v_0) = 0
   setze l(v) = l({v_0,v}) für alle v ∈ N(v_0)
   setze l(v) = ∞ für alle übrigen Knoten
```

setze $i = 0$

2. Solange $i < n-1$
   a) finde $v_{i+1} \in V \setminus V_i$ mit $l(v_{i+1})$ minimal
   b) finde $e = \{v, v_{i+1}\}$ mit $e \in E$, $v \in V_i$ und $l(v_{i+1}) = l(v) + l(e)$
   c) setze $V_{i+1} = V_i \cup \{v_{i+1}\}$, $E_{i+1} = E_i \cup \{e\}$, $\text{pred}(v_{i+1}) = v$
   d) für alle $v \in N(v_{i+1})$ setze $l(v) = \min\{l(v), l(v_{i+1}) + l(\{v_{i+1}, v\})\}$
   e) $i = i + 1$
3. Ausgabe $T = (V_{n-1}, E_{n-1})$

SATZ: Sei T eine Ausgabe des Dijkstra-Algorithmus, dann gilt:

1. T ist aufspannender Baum.
2. Der eindeutig bestimmte v_0-v-Weg in T ist ein kürzester Weg in G für alle $v \in V$.
3. Die Laufzeit des Dijkstra-Algorithmus ist $O(n^2)$.

BEWEIS:

1. In der Übung.
2. Wir zeigen, dass für alle $v \in V$ gilt $l(v) = d_G(v_0, v)$ und $l(v) = d_T(v_0, v)$ ab dem Zeitpunkt, wo v in den Baum (d.h. in ein V_i) aufgenommen wurde. Das ist klar für v_0. Sei $v \in V$ und gelte die Aussage für Knoten, die vor v in den Baum aufgenommen wurden. Sei $P = (v_0 = u_0, u_1, \ldots, u_k = v)$ ein Weg in G, der kürzer ist als der Weg in T. Sei j maximal mit der Eigenschaft, dass u_j vor v in den Baum aufgenommen wurde. Dann gilt

$$\begin{aligned} l(u_{j+1}) &\leq l(u_j) + l(\{u_j, u_{j+1}\}) \\ &= d(v_0, u_j) + l(\{u_j, u_{j+1}\}) \\ &= d(v_0, u_{j+1}) \\ &\leq d(v_0, v) \end{aligned}$$

   Wäre $l(v) > d(v_0, v)$, so hätte v_{j+1} vor v in den Baum aufgenommen werden müssen.
3. Die Laufzeit ist klar.

BEMERKUNGEN:

- Durch Setzen von $\text{pred}(v_{i+1}) = v$ im Schritt (2c), lässt sich ein kürzester Weg von v_0 nach v (in umgekehrter Richtung) angeben als $v, \text{pred}(v), \text{pred}(\text{pred}(v)), \ldots, v_0$.

- Es gibt einen Algorithmus für das kürzeste-Wege-Problem mit Laufzeit $O(m \log n)$.
- Es gibt einen Algorithmus für das kürzeste-Wege-Problem mit beliebigen (auch negativen) Kantengewichten, der in Laufzeit $O(n^3)$ entweder einen Kreis negativer Länge findet oder alle kürzesten Wege (die vollständige Entfernungsmatrix) berechnet.

5 Matching und Knotenüberdeckung in bipartiten Graphen

5.1 Der Heiratssatz

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

DEFINITIONEN:

- $M \subset E$ heißt *Matching*, wenn die Kanten aus M paarweise leeren Schnitt haben.
- $v \in V$ wird *überdeckt* von $M \subset E$, wenn es ein $e \in M$ gibt mit $v \in e$.
- Für $k \in \mathbb{N}_0$ heißt ein Teilgraph $G' = (V, E')$ *k-Faktor*, falls $\deg_{G'}(v) = k$ für alle $v \in V$ (wir sagen auch G' ist *$k-$regulär*)
- Ein 1-Faktor heißt *perfektes Matching*.
- Ein Graph heißt *bipartit*, wenn er 2-färbbar ist, d.h. wenn es eine Partition $V = S \dot\cup T$ gibt, so dass jede Kante genau einen Knoten aus S und einen Knoten aus T enthält.
- Für $V_0 \subset V$ sei $G|_{V_0} = \left(V_0, E \cap \binom{V_0}{2}\right)$, der von V_0 induzierte Teilgraph.
- Ein Matching maximaler Kardinalität heißt *Maximum-Matching*. Mit $\nu(G)$ bezeichnen wir die Kardinalität eines Maximum-Matchings.

SATZ: (Heiratssatz, Hall 1935) Sei $G = (S \dot\cup T, E)$ ein bipartiter Graph. Dann ist $\nu(G) = |S|$ genau dann wenn die *Hallbedingung* gilt:

$$\forall A \subseteq S\colon |N(A)| \geq |A|$$

BEWEIS:

„$\Rightarrow$“ Klar

„$\Leftarrow$“ Induktion nach $|S|$. Sei $|S| \geq 2$. Betrachte 2 Fälle:

Fall 1 Für alle $\emptyset \neq A \subsetneq S$ gilt $|N(A)| > |A|$. Sei $e \in E$ mit $e = \{s, t\}$, $s \in S, t \in T$. Setze $G' = G - s - t$. Dann gilt für alle $A \subseteq S \setminus \{s\}$:

$$|N_{G'}(A)| \geq |N_G(A)| - 1 \geq |A|$$

Nach Induktion gibt es ein Matching M' von G' mit $|M| = |S| - 1$. Also ist $M = M' \cup \{e\}$ ein Matching in G mit $|M| = |S|$.

Fall 2 Es existiert ein $A' \subset S,\ A' \neq \emptyset, S$ mit $|N(A')| = |A'|$. Setze $B' = N(A')$. Der Graph $G|_{A' \cup B'}$ erfüllt die Hallbedingung und enthält somit nach Induktion ein Matching M' mit $|M'| = |A'|$. Setze $A^* = S \setminus A'$ und $B^* = T \setminus B'$. Angenommen, es existiert $A \subset A^*$ mit $|N_{G^*}(A)| < |A|$. Dann ist

$$|N_G(A' \cup A)| = |N_{G^*}(A)| + |N(A')| < |A| + |A'| = |A \cup A'|$$

im Widerspruch zur Hallbedingung. Also existiert in G^* ein matching M^* mit $|M^*| = |A^*|$. Damit ist $M' \cup M^*$ ein Matching mit $|M' \cup M^*| = |S|$.

DEFINITION: Der *Matchingdefekt* von $G = (S \dot\cup T, E)$ ist

$$\operatorname{def}(G) = \max\{|A| - |N(A)| \mid A \subseteq S\}$$

BEMERKUNG: Ist die Hallbedingung erfüllt, so ist $\operatorname{def}(G) = 0$ (mit $A = \emptyset$).

SATZ: (Defektversion des Heiratssatzes) Sei $G = (S \dot\cup T, E)$ ein bipartiter Graph. Dann ist $\nu(G) = |S| - \operatorname{def}(G)$

BEWEIS: In der Übung.[89]

DEFINITION: Eine *Knotenüberdeckung (Vertex-Cover)* ist eine Teilmenge der Knotenmenge, mit der jede Kante inzident ist. Mit $\tau(G)$ bezeichnen wir die Größe einer Knotenüberdeckung minimaler Mächtigkeit.

BEMERKUNG: In jedem Graphen G gilt $\nu(G) \leq \tau(G)$.

BEWEIS: In der Übung.[10]

SATZ: (König, 1931) Ist $G = (S \dot\cup T, E)$ bipartit, so gilt $\nu(G) = \tau(G)$

BEWEIS: Sei $A \subseteq S$ mit $\operatorname{def}(G) = |A| - |N(A)|$. Dann ist $C = (S \setminus A) \cup N(A)$ ein Vertex-Cover. Also

$$\tau(G) \leq |C| = |S| - |A| + |N(A)| = |S| - \operatorname{def}(G) = \nu(G)$$

[8] Trick: Defekt „wegzaubern“ ⇒ Heiratssatz ⇒ perfektes Matching.

[9] Ab dem 7.12 gibt es eine Anwesenheitsliste für die Vorlesung. Sie dient nur statistischen Zwecken – die Teilnahme ist nach wie vor nicht obligatorisch.

[10] Beweisidee: Um ein Maximum-Matching zu überdecken, muss man für jede Kante mindestens einen Knoten wählen.

DEFINITION: Eine *Kantenüberdeckung* ist eine Teilmenge der Kantenmenge, so dass jeder Knoten in einer Kante davon enthalten ist. Mit $\rho(G)$ bezeichnen wir die Größe einer Kantenüberdeckung minimaler Mächtigkeit.

BEMERKUNG: In jedem Graphen G gilt $\alpha(G) \leq \rho(G)$.

SATZ: (Gallai, 1958-1959) Sei G ein beliebiger Graph ohne isolierte Knoten. Dann gilt

- $\alpha(G) + \tau(G) = |V|$
- $\nu(G) + \rho(G) = |V|$

BEWEIS: In der Übung. [11]

SATZ: (König, 1932) Ist G bipartit, so gilt $\alpha(G) = \rho(G)$.

BEWEIS: Es gilt

$$\alpha(G) = |V| - \tau(G) = |V| - \nu(G) = \rho(G)$$

5.2 Berechnung eines Maximum-Matching

DEFINITION: Seien $M, M' \subseteq E$. Ein Pfad heißt (M, M')-*alternierend*, wenn er abwechselnd Kanten aus M und M' durchläuft. Für ein Matching M heißt ein Pfad M-*augmentierend*, wenn er $(M, E \setminus M)$-alternierend ist und seine Endknoten nicht von M überdeckt sind.

BEMERKUNG: Sei E_0 die Kantenmenge eines M-augmentierenden Pfades. Dann ist $M \,\Delta\, E_0 = (M \cup E_0) \setminus (M \cap E_0)$ ein Matching der Größe $|M| + 1$.

SATZ: (Berge, 1957) Sei $G = (V, E)$ ein Graph und M ein Matching von G. M ist genau dann ein Maximum-Matching, wenn es keine M-augmentierenden Pfade gibt.

BEWEIS:

„⇒“ Siehe Bemerkung.

„⇐“ Sei M ein Matching, so dass es keine M-augmentierenden Pfade gibt. Sei M^* ein Maximum-Matching. Sei H der von $M \cup M^*$ aufgespannte Teilgraph von G (also $H = (\bigcup(M \cup M^*), M \cup M^*)$). Alle Knoten in H haben Grad höchstens 2, d.h. H besteht aus Kreisen gerader Länge und

[11] Hinweis: Ist U eine unabhängige Menge, so ist $V \setminus U$ eine Knotenüberdeckung und umgekehrt.

(M, M^*)-alternierenden Pfaden. Diese können nicht ungerade Länge haben, da es sonst M- oder M^*-augmentierende Pfade gibt. Also besitzen alle Zusammenhangskomponenten von H gleich viele Kanten aus M und M^*, d.h. $|M| = |M^*|$.

DEFINITION: Ein Paar $\vec{G} = (V, \vec{E})$ mit V endliche Menge, $\vec{E} \subseteq V \times V \setminus \mathrm{id}_V$ heißt *gerichteter Graph*. Eine Knotenfolge $v_1, \ldots, v_k$ heißt *(gerichteter) Pfad*, wenn $(v_i, v_{i+1}) \in \vec{E}$ für alle $i \in [k-1]$.

Algorithmus Maximum-Matching: Eingabe: $G = (S \dot\cup T, E)$ bipartit. Ausgabe: Maximum-Matching M.

1. Sei M irgendein Matching (ggf. $M = \emptyset$)
2. Konstruiere aus G den gerichteten Graphen $H = (V, \vec{E})$: Für $\{s, t\} \in E$ mit $s \in S$ und $t \in T$ füge (s, t) zu $\vec{E}$ hinzu, falls $\{s, t\} \notin M$, sonst (t, s).
3. Suche einen Pfad P in H, der von $S \setminus \bigcup M$ nach $T \setminus \bigcup M$ führt.
   (a) Gibt es einen solchen, so augmentiere M entlang dieses Pfades ($M := M \,\Delta\, E(P)$) und gehe zu 2.
   (b) Gibt es keinen, gib M aus, Ende.

SATZ: Der Algorithmus berechnet ein Maximum-Matching in Zeit $O(mn)$.

BEWEIS: Seien $s \in S, t \in T$ mit $s, t \notin \bigcup M$ und P ein s-t-Pfad in H. Dann ist P M-augmentierend. Umgekehrt ist jeder M-augmentierender Pfad von S nach T ein gerichteter Pfad in H. Daraus folgt die Korrektheit des Algorithmus.

Zur Laufzeit: Schritt 3. ist in $O(m)$ mit modifizierter Graphensuche möglich. Schritt 2. läuft auch in $O(m)$. Die beiden Schritte werden $O(n)$-mal wiederholt.

SATZ: Sei $G = (S \dot\cup T, E)$ ein bipartiter Graph, sei M ein Maximum-Matching. Sei $H = (S \dot\cup T, \vec{E})$ der gerichtete Graph zu G und M wie im Algorithmus, d.h. für alle $s \in S, t \in T$ mit $\{s, t\} \in E$ gilt

$$\begin{aligned} (s, t) &\in \vec{E}, \quad \text{falls} \quad s, t \notin M \\ (t, s) &\in \vec{E}, \quad \text{falls} \quad s, t \in M \end{aligned}$$

Sei $S' = S \setminus \bigcup M, T' = T \setminus \bigcup M$ und R die Menge aller Knoten, die in H von S' aus über einen Pfad erreichbar sind. Dann ist $C = (S \setminus R) \cup (T \cap R)$ ein Minimum-Vertex-Cover.

BEWEIS: Seien $s \in S, t \in T$ mit $\{s,t\} \in E$. Angenommen, $s,t \notin C$. Dann ist $s \in R$ und $t \notin R$. Sei P ein Pfad in H von S' nach s. Da $t \notin R$, ist $(s,t) \notin \vec{E}$, also $\{s,t\} \in M$. Damit $s \notin S'$, also hat s einen Vorgänger $t' \neq t$ in P. Aus $(t',s) \in \vec{E}$ folgt $\{s,t'\} \in M$, ein Widerpsruch, da M ein Matching ist.

Angenommen, es existiert $s \in S, t \in T$ mit $\{s,t\} \in M$ und $s,t \in C$. Dann ist $t \in R$ und $(t,s) \in \vec{E}$, also $s \in R$, Widerspruch. Jede Matchingkante enthält also höchstens einen Knoten aus C, also $|C| \leq |M|$. Somit ist $|C| = |M|$ und C ist ein Minimum-Vertex-Cover.

KOROLLAR: Für bipartite Graphen kann ein Minimum-Vertex-Cover in Zeit $O(mn)$ berechnet werden.

BEMERKUNGEN:

- Es gibt Algorithmen für Minimum-Vertex-Cover und Maximum-Matching in bipartiten Graphen mit Laufzeit $O(m\sqrt{n})$.
- Ist $w\colon E \to \mathbb{R}$ zusätzlich eine Gewichtsfunktion, so kann man ein Matching mit maximalem Gewicht in Zeit $O(n^2 m)$ (einfach) oder in Zeit $O(mn + n^2 \log(n))$ (komplizierter) berechnen (sog. „Ungarische Methode“)

6 Matching in allgemeinen Graphen - Der Satz von Tutte

DEFINITION: Sei $q(G)$ die Anzahl der Zusammenhangskomponenten von G mit ungerader Knotenzahl.

BEMERKUNG: Besitzt G ein perfektes Matching (einen 1-Faktor), so gilt für alle $S \subseteq V$: $|S| \geq q(G - S)$

SATZ: (Tutte, 1947) Ein Graph besitzt einen 1-Faktor genau dann, wenn für alle $S \subseteq V$ gilt

$$|S| \geq q(G - S)$$

BEWEIS:

„⇒“ Bemerkung

„⇐“ Angenommen, G besitzt keinen 1-Faktor. Sei $G^* = (V, E^*)$ ein Graph, der G als Teilgraph enthält, möglichst viele Kanten besitzt und keinen 1-Faktor hat. Offenbar gilt weiterhin für alle $S \subseteq V$:

$$|S| \geq q(G - S) \geq q(G^* - S)$$

Setze $K = \{ v \in V \mid N_{G^*}(v) = V \setminus v\}$. Sei $G' = G^* - K$. Wir werden zeigen, dass alle Zusammenhangskomponenten von G' vollständig sind. Dann können wir einen 1-Faktor für G^* wählen: Die Knoten in G' können alle gematcht werden bis auf je einen aus den ungeraden Zusammenhangskomponenten. Wegen $|K| \geq q(G^* - K)$ können diese mit Knoten aus K gematcht werden. Übrig bleiben gerade viele Knoten aus K (sonst wäre $|V|$ ungerade), diese können in K gematcht werden.

Angenommen, es existiert eine Zusammenhangskomponente von G', die nicht vollständig ist. Dann existieren paarweise verschiedene $a, b, c \in V \setminus K$ mit $\{a, b\}, \{b, c\} \in E$, aber $\{a, c\} \notin E$. Ferner existiert ein $d \in V$ mit $\{b, d\} \notin E$. Da G^* kantenmaximal ist, besitzen $G_1 = G^* + \{a, c\}$ und $G_2 = G^* + \{b, d\}$ jeweils perfekte Matchings M_1, M_2. Da G^* keinen 1-Faktor besitzt, ist $\{a, c\} \in M_1$ und $\{b, d\} \in M_2$. Sei $P = (d, \ldots, y, x)$ ein maximaler in d beginnender (M_1, M_2)-alternierender Pfad in G^*, der mit einer M_1-Kante beginnt.

Fall 1 $\{y, x\} \in M_1$: Aus Maximalität und Konstruktion folgt $x = b$. Setze $M_2' = (M_2 \,\Delta\, E(P)) \setminus \{\{b, d\}\}$. Nach Konstruktion ist M_2' ein Matching in G^* und $|M_2'| = |M_2|$, also ist M_2' ein perfektes Matching von G^*, ein Widerspruch.

Fall 2 $\{y, x\} \in M_2$: Dann ist $x \in \{a, c\}$, o.B.d.A. $x = c$. Setze $M_2' = (M_2 \,\Delta\, E(P)) \cup \{\{b, c\}\} \setminus \{\{b, d\}\}$. Dann ist M_2' ein Matching mit $|M_2'| = |M_2|$, ein Widerspruch.

SATZ: (Petersen, 1891) Jeder 3-reguläre Graph (also mit $\deg(v) = 3$ für alle Knoten v) ohne Brücke (Kante, deren Herausnahme die Zahl der Zusammenhangskomponenten erhöht) hat einen 1-Faktor.

BEWEIS: In der Übung.

Der **Satz von Gallai-Edmonds**, der weitere Aussagen zum Matching-Problem in allgemeinen Graphen macht, wird in dieser Vorlesung nicht behandelt: er ist zu technisch und kompliziert.

7 Färbbarkeit

7.1 Knotenfärbungen in Graphen

DEFINITION: Sei $k \in \mathbb{N}$. Eine *k-(Knoten)färbung* ist eine Abbildung $c\colon V \to [k]$ mit $c(u) \neq c(v)$ für alle $\{u, v\} \in E$. Setze

$$\chi(G) = \min\{k \in \mathbb{N} \mid \text{ es existiert eine } k\text{-Färbung von } G\}$$

BEMERKUNGEN: Sei H ein Teilgraph von G. Dann gilt:

- $\chi(H) \leq \chi(G)$. Insbesondere ist $\chi(G) \geq \omega(G)$.
- $\chi(G) = 2$ genau dann, wenn G bipartit ist und $E \neq \emptyset$.
- $\chi(G) \geq 3$ genau dann, wenn G einen Kreis ungerader Länge enthält.
- $\chi(G) \geq \frac{|V|}{\alpha(G)}$

LEMMA: $\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$

BEWEIS: Sei $c\colon V \to [k]$ mit $k = \chi(G)$ eine Knotenfärbung. Zwischen je zwei Farbklassen verläuft eine Kante, sonst könnte man beide mit derselben Farbe färben. Also ist $m \geq \binom{k}{2} = \frac{k(k-1)}{2}$.

Algorithmus GreedyFärbung:

```
// Eingabe: G = (V,E), Enumeration v_1,...,v_n von V
// Ausgabe: Färbung c: V → [k]
Für i = 1,...,n
      setze c(v_i) als die kleinste natürliche Zahl,
      die keinem der Nachbaren von v zugewiesen wurde.
```

SATZ: Sei G ein Graph mit Maximalgrad $\Delta = \Delta(G)$. Dann findet der Greedy-Algorithmus eine Färbung mit höchstens $\Delta + 1$ Farben. Ist G nicht regulär (nicht alle Knoten haben denselben Grad) und zusammenhängend, so findet eine geeignete Variante des Greedy-Algorithmus eine Färbung mit höchsten Δ Farben.

BEWEIS: Die erste Aussage ist trivial. Sei $v_1 \in V$ mit $\deg(v) < \Delta$ (existiert, da G nicht regulär). Mit Graphensuche bestimme eine Knotenenumeration $v_1, \ldots, v_n$. Färbe die Knoten mit dem Greedy-Algorithmus in der Reihenfolge $v_n, v_{n-1}, \ldots, v_1$. Da jeder Knoten aus $\{v_2, \ldots, v_n\}$ einen Nachbaren mit kleinerer Nummer besitzt, werden $v_2, \ldots, v_n$ mit Δ Farben gefärbt. Da $\deg(v_1) < \Delta$,

ist $c(v_1) \in [\Delta]$. $\square$

Exakte Lösungen des Färbungsproblems:

1. Für $k = 1, 2, \ldots$ teste alle $c\colon V \to [k]$ auf Zulässigkeit. Aufwand $\approx O(\chi(G)^n m)$
2. Für alle Enumerationen von V starte den Greedy-Algorithmus. Korrektheit: siehe unten. Aufwand: $O(n!n) = O\left(n^{3/2}\left(\frac{n}{e}\right)^n\right)$

SATZ: Es existiert immer eine Knotenenumeration, so dass der Greedy-Algorithmus eine optimale Färbung findet.

BEWEIS: In der Übung.

BEMERKUNG: „Der Greedy-Algorithmus kann beliebig schlecht sein“:

1. Für alle $k \in \mathbb{N}$ existieren bipartite Graphen auf $2k$ Knoten und eine Knotenenumeration, so dass der Greedy-Algorithmus k Farben benötigt.
2. Es existieren Bäume auf n Knoten, so dass für eine geeignete Knotenenumeration der Greedy-Algorithmus $\Omega(\log n)$ Farben benötigt.

BEWEIS: In der Übung.

SATZ: Eine geeignete Implementierung des Greedy-Algorithmus findet eine Färbung mit höchstens $1+\max\{\delta(H) \mid H \text{ Teilgraph von } G\}$ Farben, wobei $\delta(H) = \min\{\deg(v) \mid v \text{ Knoten von } H\}$, in Zeit $O(n^2)$

KOROLLAR: Jeder Graph G hat einen Teilgraphen mit Minimalgrad mindestens $\chi(G) - 1$.

BEWEIS: In der Übung.

DEFINITIONEN: Ein Graph G heißt 2-*zusammenhängend*, wenn $G - v$ für alle $v \in V$ zusammenhängend ist und $|V| \geq 3$. Eine *Artikulation* in G ist ein Knoten $v \in V$, so dass $G - v$ mehr Zusammenhangskomponenten besitzt als G. Ein *Block* in G ist ein (inklusions-)maximaler zusammenhängender nicht-trivialer induzierter Teilgraph von G ohne Artikulationen. Sei $\mathcal{B}$ die Menge der Blöcke von G und A die Menge der Artikulationen von G. Setze

$$bc(G) = (A \cup \mathcal{B}, \{\{a, B\} \mid a \in A, B \in \mathcal{B}, a \text{ ist ein Knoten in } B\})$$

SATZ: (Galil 1964, Hardy, Preuss 1966) Ist G zusammenhängend, so ist $bc(G)$ ein Baum.

KOROLLAR: Sei $\mathcal{B}$ die Menge der Blöcke von G. Dann gilt

$$\chi(G) = \max\{\chi(B) \mid B \in \mathcal{B}\}$$

LEMMA: (Lovasz) Sei $G = (V, E)$ ein 2-zusammenhängender Graph, der kein Kreis und kein vollständiger Graph ist. Dann existieren $x, y \in V$ mit $d(x, y) = 2$ und $G - x - y$ zusammenhängend.

BEWEIS: Sei $\Delta = \Delta(G)$ und $v \in V$ mit $\deg(v) = \Delta$. Dann existieren $\hat{x}, \hat{y} \in N(v)$ mit $d(\hat{x}, \hat{y}) = 2$. Anderenfalls sind alle Knoten aus $N(v)$ durch Kanten verbunden, sie haben also alle Δ Nachbaren in $N(v) \cup \{v\}$. Damit ist $N(v) \cup \{v\}$ eine Zusammenhangskomponente von G, also schon G, ein Widerspruch zu G vollständiger Graph.

Sei $G - \hat{x} - \hat{y}$ nicht zusammenhängend (sonst sind wir fertig). Nach Voraussetzung ist $\Delta = \deg(v) \geq 3$. Sei C die Zusammenhangskomponente von $G - \hat{x} - \hat{y}$, die v enthält. $C - v$ enthält einen Knoten x, der zu $\hat{x}$ oder $\hat{y}$ benachbart ist. Sei C' eine weitere (also $C' \neq C$) Zusammenhangskomponente von $G - \hat{x} - \hat{y}$. Sowohl $\hat{x}$ als auch $\hat{y}$ haben Nachbaren in C', also existiert ein $y \in C'$ mit $d(x, y) = 2$. Wir zeigen, dass $G - x - y$ zusammenhängend ist.

Nach Voraussetzung ist $G - x$ zusammenhängend. Also ist jeder Knoten aus $C - x$ in $G - x$ mit $\hat{x}$ oder $\hat{y}$ durch einen Pfad verbunden. Keiner dieser Pfade enthält y, da sonst C und C' in $G - \hat{x} - \hat{y}$ verbunden wären. Genauso ist $G - y$ zusammenhängend. Also ist jeder Knoten aus $V \setminus (C \cup \{\hat{x}, \hat{y}, y\})$ mit $\hat{x}$ oder $\hat{y}$ durch einen Pfad verbunden. Keiner dieser Pfade enthält x, da sonst C mit einem Knoten aus einer anderen Zusammenhangskomponente von $G - \hat{x} - \hat{y}$ verbunden wäre. Also sind alle Knoten von $G - x - y$ an den Pfad $\hat{x}, v, \hat{y}$ angeschlossen, d.h. $G - x - y$ ist zusammenhängend.

SATZ: (Brooks, 1941) Sei G ein zusammenhängender Graph, der kein ungerader Kreis und kein vollständiger Graph ist. Dann gilt $\chi(G) \leq \Delta(G)$.

BEWEIS: Sei $\Delta = \Delta(G)$. Falls G nicht Δ-regulär ist, folgt die Behauptung aus dem Satz am Anfang des Kapitels 7.1. Sei also G Δ-regulär. Ist $\Delta \leq 2$, so ist G ein Kreis gerader Länge, also $\chi(G) = \Delta = 2$. Sei also $\Delta \geq 3$.

Sei zunächst G 2-Zusammenhängend. Nach Lemma von Lovasz existieren $x, y \in V$ mit $d(x, y) = 2$ und $G - x - y$ zusammenhängend. Sei v_1 ein Knoten zwischen x und y. Mit Graphensuche nummerieren wir mit $(v_2, \ldots, v_{n-2})$ in v_1 beginnend die Knoten von $G - x - y$. Wir setzen $v_{n-1} = x$ und $v_n = y$. Mit dem Greedy-Algorithmus färben wir G in der Reihenfolge $v_n, \ldots, v_1$. Dann haben x und y dieselbe Farbe. Für $v_{n-2}, \ldots, v_2$ finden wir eine Farbe in $[\Delta]$, da jeder dieser Knoten einen Nachbar mit kleinerer Nummer besitzt. Für

v_1 finden wir eine Farbe in $[\Delta]$, da zwei seiner Nachbaren, nämlich x und y, dieselbe Farbe haben.

Ist G nicht 2-zusammenhängend, so können wir die Blöcke von G mit Δ Farben färben und so eine Färbung von G gewinnen (vgl. Korollar)

BEMERKUNG: Der Beweis liefert sogar einen polynomiellen Algorithmus, der eine entsprechende Färbung findet.

DEFINITION: Die *Taillenweite* $g(G)$ ist die Länge eines kleinsten Kreises von G.

SATZ: (Erdös, $\sim$ 1950) Für alle $k, l \in \mathbb{N}$ existiert ein Graph G mit $g(G) > k$ und $\chi(G) > l$.

7.2 Kantenfärbungen in Graphen

DEFINITION: Eine Abbildung $f\colon E \to [k]$ heißt *k-Kantenfärbung*, falls gleichfarbige Kanten sich nicht schneiden, d.h. für alle $e, e' \in E$ gilt: ist $|e \cap e'| = 1$, so ist $f(e) \neq f(e')$. Der *chromatische Index* von G sei

$$\chi'(G) = \min \{\, k \in \mathbb{N} \mid \text{ es existiert eine } k\text{-Kantenfärbung von } G \,\}$$

Der Graph $L(G) = (E, \{\, \{e, e'\} \mid e, e' \in E,\ |e \cap e'| = 1\})$ heißt *Linegraph* von G.

LEMMA:

1. $\chi'(G) = \chi(L(G)) \geq \omega(L(G)) \geq \Delta(G)$
2. $\chi'(G) \geq \frac{m}{\nu(G)}$
3. $\chi'(K_n) = n - 1$ falls n gerade und $\chi'(K_n) = n$ falls n ungerade.

BEWEIS: In der Übung.

SATZ: (Vizing, 1964) Für alle Graphen G gilt $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$. Eine Kantenfärbung mit $\Delta(G) + 1$ Farben kann in Zeit $O(mn\Delta)$ berechnet werden.

BEWEIS: Induktion nach $|E|$. Induktionsanfang ist trivial. Sei $G = (V, E), \Delta = \Delta(G)$ und die Behauptung gelte für Graphen mit $|E| - 1$ Kanten. Mit „Färbung“ sei im folgenden immer eine Kantenfärbung mit (maximal) $\Delta + 1$ Farben gemeint.

Nach Induktion gibt es für alle $e \in E$ eine Färbung für $G - e$. Zu jedem $v \in V$

gibt es mindestens eine Farbe β, die „am v fehlt“. Ist α eine weitere Farbe, so existiert ein eindeutiger in v beginnender Weg maximaler Länge, dessen Kanten abwechselnd mit α und β gefärbt sind (diesen nennen wir *α-β-Weg aus v*).

Angenommen, G besitzt keine Färbung. Dann gilt:

(*) Ist $\{x, y\} \in E$ und eine Färbung von $G - \{x, y\}$ gegeben, in der die Farbe α an x und β an y fehlt, so endet der α-β-Weg aus y in x: Sonst könnten wir auf dem Weg die beiden Farben vertauschen und $\{x, y\}$ mit α färben.

Sei $\{x, y_0\} \in E$. Sei c_0 eine Färbung von $G_0 = G - \{x, y_0\}$. Sei α die an x fehlende Farbe. Sei $y_0, \ldots, y_k$ eine maximale mit y_0 beginnende Folge verschiedener Nachbaren von x, so dass jeweils $c_0(\{x, y_i\})$ an y_{i-1} fehlt. Auf jedem der Graphen $G_i = G - \{x, y_i\}$ definieren wir eine Färbung c_i durch

$$c_i(e) = \begin{cases} c_0(\{x, y_{j+1}\}) & \text{falls } e = \{x, y_j\}, j = 0, \ldots, i-1 \\ c_0(e) & \text{sonst} \end{cases}$$

Die Menge der an x fehlenden Farben ändert sich dadurch nicht. Sei β eine in c_0 an y_k fehlende Farbe. β fehlt nicht an x, sonst könnten wir c_k zu einer Färbung von G ergänzen. Wegen der Maximalität von k existiert ein $i \in [k-1]$ mit $c_0(\{x, y_i\}) = \beta$. Sei P der α-β-Weg aus y_k in G_k bezüglich c_k. Wegen (*) endet P in x mit einer β-Kante. Wegen $\beta = c_0(\{x, y_i\}) = c_k(\{x, y_{i-1}\})$ ist dies die Kante $\{x, y_{i-1}\}$. Nach Wahl von y_i fehlt β an y_{i-1} in c_0 und damit auch in c_{i-1}. Sei P' der α-β-Weg aus y_{i-1} in G_{i-1} bezüglich c_{i-1}. Wegen Eindeutigkeit geht P' von y_{i-1} über P nach y_k und endet dort, da β in c_0 und c_{i-1} an y_k fehlt. Dies ist ein Widerspruch zu (*) angewandt auf $G_{i-1}, c_{i-1}, x, y_{i-1}$.

7.3 Perfekte Graphen

DEFINITION: Das *Komplement* eines Graphen G ist $\bar{G} = \left(V, \binom{V}{2} \setminus E\right)$.

DEFINITION: Ein Graph G heißt *perfekt*, wenn für alle induzierten Teilgraphen H gilt $\chi(H) = \omega(H)$

BEISPIELE:

- Bipartite Graphen sind perfekt.
- Komplemente von bipartiten Graphen sind perfekt.
- Ungerade Kreise der Länge größer 3 sind nicht perfekt.

SATZ: („Hauptsatz“ der Theorie der perfekten Graphen, Lovasz, 1972) G ist perfekt genau dann, wenn $\bar{G}$ es ist.

SATZ: (Strong Perfect Graph Conjecture, Berge, 1966[12]) G ist genau dann perfekt, wenn weder G noch $\bar{G}$ einen Kreis ungerader Länge größer gleich 5 als induzierten Teilgraphen enthält.

7.4 Ramsey-Theorie

Die Ramsey-Theorie beschäftigt sich mit 2-Färbungen von Hypergraphen, wo keine Hyperkante monochromatisch ist.

DEFINITIONEN:

- Ein Paar $\mathcal{H} = (V, \mathcal{E})$ mit V endliche Menge und $\mathcal{E} \subseteq \mathcal{P}(V)$ heißt *Hypergraph*.
- $\mathcal{H}$ heißt 2-*färbbar*, wenn es ein $f\colon V \to \{1,2\}$ gibt, so dass $|f(E)| = 2$ für alle $E \in \mathcal{E}$ ist.
- $\mathcal{H}$ heißt k-*uniform*, wenn $|E| = k$ für alle $E \in \mathcal{E}$ ist.

SATZ: Ist $\mathcal{H}$ k-uniform und $|\mathcal{E}| < 2^{k-1}$, so ist $\mathcal{H}$ 2-färbbar.

BEWEIS: Sei $f\colon V \to \{1,2\}$ eine zufällige Färbung, d.h. für alle $v \in V$ unabhängig gilt

$$P(f(v) = 1) = P(f(v) = 2) = \frac{1}{2}$$

Für $E \in \mathcal{E}$ gilt $P(|f(E)| = 1) = 2 \cdot 2^{-k} = 2^{1-k}$. Damit gilt

$$P(\exists E \in \mathcal{E}\colon |f(E)| = 1) \leq \sum_{E \in \mathcal{E}} P(|f(E)| = 1) = |\mathcal{E}| \cdot 2^{1-k} < 1$$

Also existiert $f\colon V \to \{1,2\}$ mit $|f(E)| = 2$ für alle $E \in \mathcal{E}$. □

BEMERKUNG: Die Frage nach 2-Färbbarkeit von beliebigen Hypergraphen ist NP-vollständig.

SATZ: (Ramsey 1930) Für alle $k \in \mathbb{N}$ existiert $n \in \mathbb{N}$, so dass jede (beliebige) 2-Färbung der Kanten des K_n einen monochromatischen K_k erzeugt.

BEISPIEL: Die Aussage gilt für $k = 3$ mit $n = 6$.

[12] Bewiesen 2002 von Maria Chudnovsky, Neil Robertson, Paul Seymour und Robin Thomas.

SATZ: (Van der Waerden, 1927) Jede 2-Färbung der natürlichen Zahlen erzeugt beliebig lange monochromatische arithmetische Progressionen.

7.5 Diskrepanztheorie

7.5.1 Geometrisches Diskrepanzproblem

ZIEL: Verteile Punkte gleichmäßig in einem geometrischen Setting. Hier: Finde n Punkte in $[0,1]^d$, so dass alle achsenparallele Rechtecke R (möglichst) $n \operatorname{vol}(R)$ Punkte enthalten.

DEFINITION: Sei $d \in \mathbb{N}$. Sind $x_i, y_i \in [0,1]$ mit $x_i \leq y_i$ für $i \in [d]$, so heißt $R = \prod[x_i, y_i]$ ein (hochdimensionales) *Rechteck*. Sei R_d die Menge aller solcher Rechtecke. Sei $P \subseteq [0,1]^d$ endlich und $n = |P|$. Setze

- $d(P,R) = ||P \cap R| - n \operatorname{vol}(R)|$
- $d(P, R_d) = \sup_{R \in R_d} d(P,R)$
- $d(n, R_d) = \inf_{P \subseteq [0,1]^d,\, |P|=n} d(P, R_d)$

SATZ: (Roth 1945)

- $d(n, R_d) = \Omega_d(\log(n)^{\frac{d-1}{2}})$
- $d(n, R_d) = O_d(\log(n)^{d-1})$
- $d(n, R_2) = \theta(\log(n))$

SATZ: Sei D_d die Menge der rechtwinkligen Dreiecke mit linker und unterer Seite parallel zu den Achsen. Dann gilt $d(n, D_d) = \Omega(\sqrt[4]{n})$

BEMERKUNG: Sei $f\colon [0,1]^d \to \mathbb{R}$. Dann gilt die *Koksma-Hlawka-Ungleichung*:

$$\left| \int_{[0,1]^d} f(x)dx - \frac{1}{|P|} \sum_{p \in P} f(p) \right| \leq \frac{1}{|P|} d(P, R_d) V(f)$$

Dabei ist $V(f)$ ein Maß für die Variation der Funktion (Variation im Sinne von Hardy und Krause).

7.5.2 Ausgewogenes Färben von Hypergraphen

ZIEL: Färbe $\mathcal{H}$ so, dass jede Hyperkante alle Farben gleich oft enthält.

DEFINITION: Sei $\mathcal{H} = (V, \mathcal{E})$ ein Hypergraph. Sei $c \in \mathbb{N}_{\geq 2}$. Eine Funktion $\chi \colon V \to [c]$ heißt *c-Färbung*. Setze

- $\operatorname{disc}(\mathcal{H}, \chi) = \max_{\mathcal{E} \in \varepsilon, i \in [c]} \left| |\chi^{-1}(i) \cap \mathcal{E}| - \frac{1}{c}|\mathcal{E}| \right|$
- $\operatorname{disc}(\mathcal{H}, c) = \min_{\chi \colon V \to [c]} \operatorname{disc}(\mathcal{H}, \chi)$

SATZ: $\operatorname{disc}(\mathcal{H}, c) = O\left(\sqrt{\frac{|V|}{c} \log(|\mathcal{E}|c)}\right)$

BEISPIELE:

- Betrachte den Graphen

  $$\mathcal{H} = ([n] \times [n], \{S \times T | S, T \subseteq [n]\}) = ([n], 2^{[n]}) \times ([n], 2^{[n]})$$

  Der Satz liefert $\operatorname{disc}(\mathcal{H}, 2) = O(n^{\frac{3}{2}})$, aber es ist unklar, wie die entsprechende Färbung aussieht.

- Ein berühmtes offenes Problem ist das 3-*Permutationen-Problem*. Seien $\sigma_1, \sigma_2, \sigma_3$ Permutationen von $[n]$. Betrachte den Graphen $\mathcal{H} = ([n], \mathcal{E})$, wobei $\mathcal{E}$ die Menge der Intervalle bezüglich mindestens einer der Ordnungen σ_i ist. Es gilt $\operatorname{disc}(\mathcal{H}, c) = O(\log(n))$, aber es ist offen, ob auch $\operatorname{disc}(\mathcal{H}, c) = O(1)$ gilt.

7.5.3 Ausgewogenes Färben von Hypergraphen mit 2 Farben

DEFINITIONEN: Sei $\mathcal{H} = (V, \mathcal{E})$ ein Hypergraph. Sei $\chi \colon V \to \{-1, 1\}$ eine 2-Färbung. Für $E \in \mathcal{E}$ sei $\chi(E) = \sum_{i \in E} \chi(i)$. Weiter sei definiert

- Die *Diskrepanz* von E sei $|\chi(E)|$
- Die *Diskrepanz* von $\mathcal{H}$ bezüglich χ sei $\operatorname{disc}(\mathcal{H}, \chi) = \max_{E \in \mathcal{E}} |\chi(E)|$
- Die *Diskrepanz* von $\mathcal{H}$ sei $\operatorname{disc}(\mathcal{H}) = \min_{\chi} |\operatorname{disc}(\mathcal{H}, \chi)|$

BEMERKUNG: Die Bestimmung von $\operatorname{disc}(\mathcal{H})$ ist ein NP-schweres Problem.

SATZ: Sei $\mathcal{H} = (V, \mathcal{E})$ ein Hypegraph und $n = |V|$, dann gilt

$$\operatorname{disc}(\mathcal{H}) = O(\sqrt{n \log(n)})$$

BEWEIS: Wir betrachten eine zufällige Färbung χ gegeben durch

$$P(\chi(i) = 1) = P(\chi(i) = -1) = \frac{1}{2} \quad \text{für alle } i = 1, \ldots, n \text{ unabhängig}$$

Wir zeigen: Für $\lambda = O(\sqrt{n \log n})$ gilt

$$P(\forall E\colon |\chi(E)| \leq \lambda) \geq \frac{1}{2} \qquad (*)$$

Dann sind wir fertig, denn daraus schon die Existenz einer entsprechenden Färbung folgt. *Beweis* von (*): Es gilt

$$P(\forall E\colon |\chi(E)| \leq \lambda) = 1 - P(\exists E\colon |\chi(E)| > \lambda)$$

Nun gilt aber mit $m = |\mathcal{E}|$

$$\begin{aligned} P(\exists E\colon |\chi(E)| > \lambda) &= P(\chi(E_1) > \lambda \vee \ldots \vee \chi(E_m) > \lambda) \\ &\leq \sum_{i=1}^{m} P(|\chi(E_i)| > \lambda) \leq \sum_{i=1}^{m} e^{-\frac{\lambda^2}{2n}} < m e^{-\frac{\lambda^2}{2n}} < \frac{1}{2} \end{aligned}$$

SATZ: (Roth 64, Spencer, Matoušek 94) Sei $AP = ([n], \mathcal{E})$, wobei $\mathcal{E}$ die Menge der arithmetischen Progressionen in $\mathcal{E}$ sei. Dann gilt

$$c_1 \sqrt[4]{n} \leq \operatorname{disc}(AP) \leq c_2 \sqrt[4]{n}$$

7.5.4 Spieltheoretische Aspekte

Wir betrachten hier die sog. *Vector-Balancing Games*. Das Spiel läuft wie folgt ab:

- Start: $P = 0$.
- Runde:
  - Spieler 1 wählt $v \in \mathbb{R}^d, ||v||_2 \leq 1$
  - Spieler 2 wählt $\epsilon \in \{-1, 1\}$
  - Es wird $P = P + \epsilon v$ gesetzt
- Ende: $\|P\|_2$ ist die Auszahlung an Spieler 1

Es wird eine feste Anzahl n von Runden gespielt.

SATZ: Der Spielwert (die minimale Auszahlung, die sich Spieler 1 garantieren kann) beträgt $\sqrt{n}$.

BEMERKUNG: Für dasselbe Spiel mit der ∞-Norm und n Runden beträgt der Wert etwa $\sqrt{n \log(n)}$.

8 Planare Graphen

DEFINITION: Sei $\gamma\colon [0,1] \to \mathbb{R}^2$ stetig. γ heißt *Jordanbogen*, falls es injektiv ist. γ heißt *Jordankurve*, falls $\gamma|_{[0,1)}$ injektiv ist und $\gamma(0) = \gamma(1)$. Wir nennen jeweils auch $\gamma([0,1])$ Jordanbogen bzw. Jordankurve.

SATZ: (Jordanscher Kurvensatz) Sei $J \subseteq \mathbb{R}^2$ eine Jordankurve. Dann hat $\mathbb{R}^2 \setminus J$ genau zwei Zusammenhangskomponenten K_1, K_2 und es gilt $\delta K_1 = J = \delta K_2$.[13]

DEFINITIONEN: Sei $G = (V, E)$ ein Graph.

- Sei $\varphi_v\colon V \to \mathbb{R}^2$ und φ_E eine Abbildung von E in die Menge der Jordanbogen mit:
  - φ_v ist injektiv
  - $\varphi_E(e) \cap \varphi_v(V) = \varphi_E(e)(\{0,1\})$ für alle $e \in E$
  - $\varphi_E(e)(\{0,1\}) = \varphi_v(e)$ für alle $e \in E$
  - $\varphi_E(e) \cap \varphi_E(f) = \varphi_v(e \cap f)$ für alle $e, f \in E, e \neq f$

  Dann heißt (φ_v, φ_E) *Einbettung* von G.
- Das Bild dieser Funktionen in $\mathbb{R}^2$ nennen wir *eingebetteten oder ebenen* Graphen.
- Ein Graph heißt *planar*, wenn er eine Einbettung besitzt. Im folgenden identifizieren wir planare Graphen und ihre Einbettungen.

DEFINITION: Sei G ein planarer Graph. Die Zusammenhangskomponenten von $\mathbb{R}^2$ heißen *Gebiete* von G.

LEMMA: Sei G planar. G hat genau ein Gebiet genau dann wenn G ein Wald ist.

SATZ: (Eulerformel) Sei G ein planarer, zusammenhängender Graph mit Gebietsmenge R. Dann gilt: $|V| - |E| + |R| = 2$.

BEWEIS: Induktion nach $|R|$: Ist $|R| = 1$, so ist G ein Baum, also $|E| = |V| - 1$ und damit gilt die Behauptung. Ist $|R| \geq 2$, so existiert ein Kreis C in G. Sei e eine Kante in C. Dann hat $G - e$ genau $|R| - 1$ Gebiete. Also gilt mit Induktion $|V| - |E \setminus \{e\}| + (|R| - 1) = 2$, somit gilt die Behauptung.

[13] Mit $\delta(K_i)$ wird der Rand von K_i gemeint.

DEFINITIONEN: Die *Taillenweite* $\rho(G)$ ist die Länge eines kürzesten Kreises in G oder ∞, falls G keine Kreise hat. Eine *Brücke* in einem zusammenhängenden Graphen ist eine Kante e, so dass $G - e$ nicht zusammenhängend ist. Ein *Dreieck* ist ein Kreis der Länge 3.

SATZ: Sei $G = (V, E)$ ein planarer zusammenhängender Graph. Setze $n = |V|, m = |E|, g = \rho(G)$. Dann gilt

$$m \leq \begin{cases} \max\left\{\frac{g}{g-2}(n-2), n-1\right\} & \text{falls } g < \infty \\ n-1 & \text{sonst} \end{cases}$$

BEWEIS: Sei $g < \infty$. Induktion nach n: für $n = 3$ ist G ein Dreieck und $m = 3$ wie behauptet. Für $n < 3$ betrachte zwei Fälle:

Fall 1 G besitzt eine Brücke e. Dann besitzt $G - e$ zwei Zusammenhangskomponenten: $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$. Mit $n_i = |V_i|, m_i = |E_i|$ für $i = 1, 2$ gilt:

$$\begin{aligned} m &= m_1 + m_2 + 1 \\ &\leq \max\left\{\frac{g}{g-2}(n_1-2), n_1-1\right\} + \max\left\{\frac{g}{g-2}(n_2-2), n_2-1\right\} + 1 \\ &\leq \max\left\{\frac{g}{g-2}(n-2), n-1\right\} \end{aligned}$$

Fall 2 G besitzt keine Brücken. Dann liegt jede Kante im Rand von genau zwei Gebieten von G. Sei r_i die Anzahl der Gebiete mit genau i begrenzenden Kanten. Dann ist

$$2m = \sum_{i=3}^{n} i r_i = \sum_{i=g}^{n} i r_i \geq g \sum_{i} r_i = g|R|$$

Also $m + 2 = n + |R| \leq n + \frac{2m}{g}$. Es folgt: $m \leq \frac{g}{g-2}(n-2)$

KOROLLAR: Sei G planar, $n = |V| \geq 3, m = |E|$. Dann ist $m \leq 3n - 6$. Ist G dreiecksfrei, so gilt: $m \leq 2n - 4$.

DEFINITION: Sei $K_{m,n} = (\{1, \ldots, m, \bar{1}, \ldots, \bar{n}\}, \{\{i, \bar{j}\} \mid i \in [m], \bar{j} \in [n]\})$ vollständiger bipartiter Graph auf $m + n$ Knoten.

KOROLLAR: K_5 und $K_{3,3}$ sind nicht planar.

KOROLLAR: Ein planarer Graph hat einen Knoten mit Grad höchstens 5.

8.1 Satz von Kuratowski und Wagner

DEFINITIONEN:

1. Seien v, w Knoten eines Graphen G. H geht aus G durch *Verschmelzen* der Knoten v, w hervor, wenn H statt v, w einen neuen Knoten enthält, der zu allen Nachbarn von v und w benachbart ist.

2. H gehe aus G durch Weglassen von Knoten und Kanten und anschließendem Verschmelzen benachbarter Knoten hervor. Dann heißt H *Minor* von G, oder G enthält H als Minor.

3. G gehe aus H hervor durch Ersetzen von Kanten durch Pfade (der Länge ≥ 1). Dann heißt G *Unterteilung* von H.

4. G enthält H als Unterteilung (oder als *topologischen Minor*), wenn G einen Teilgraphen besitzt, der eine Unterteilung von H ist.

SATZ: (Kuratowski, Wagner) Folgende Aussagen sind äquivalent:

1. G ist planar.

2. G enthält weder K_5 noch $K_{3,3}$ als Minor.

3. G enthält weder K_5 noch $K_{3,3}$ als Unterteilung.

8.2 Vier-Farben-Satz

Umgangssprache: Jede Landkarte lässt sich so mit vier Farben färben, dass Länder mit gemeinsamer Grenze positiver Länge verschiedene Farben enthalten.

SATZ: (Vier-Farben-Satz: Appel, Haker, Koch 1976) Die Färbungszahl eines planaren Graphen ist höchstens 4.

BEWEIS: durch Computerprogramm.

SATZ: (Sechs-Farben-Satz) Jeder planare Graph ist 6-färbbar.

BEWEIS: Induktion nach Knotenzahl. Der Induktionsanfang ist trivial. Sei G planar mit mehr als sechs Knoten. Dann existiert ein Knoten v mit $deg(v) \leq 5$. Mit Induktionsannahme finde eine 6-Färbung von $G - v$. Da v nun fünf Nachbaren hat, ist eine Farbe für v übrig.

SATZ: (Fünf-Farben-Satz) Jeder planare Graph ist 5-färbbar.

BEWEIS: Angenommen nicht. Sei G ein Gegenbeispiel mit minimaler Knotenzahl. Dann haben alle Knoten von G mindestens Grad 5. Also existiert ein Knoten v mit Grad genau 5. Wähle eine Einbettung von G. Seien $v_1, \ldots, v_5$ die Nachbaren von v im Uhrzeigersinn. Sei eine 5-Färbung von $G - v$ gegeben. Dann haben $v_1, \ldots, v_5$ verschiedene Farben. O.B.d.A. sei v_i mit Farbe i gefärbt. Sei H_{ij} der von den Knoten in Farbe i und j induzierte Teilgraph von G. Dann liegen v_i und v_j in der selben Zusammenhangskomponente von H_{ij} (anderenfalls können wir in einer Zusammenhangskomponente die Farben i und j vertauschen und so eine zulässige Farbe für v gewinnen). Insbesondere existiert ein Pfad von v_1 nach v_3 mit allen Knoten in Farbe 1 und 3 und einen von v_2 nach v_4 in Farbe 2 und 4 — Widerspruch zur Planarität von G.

9 Flüsse und Zusammenhang

9.1 Gerichtete Graphen und Flüsse

DEFINITIONEN:

- Ein *gerichteter Graph* ist ein Paar (V, E) mit $E \subseteq (V \times V) \setminus \mathrm{id}(V)$ (d.h. ohne Knoten (v, v)). Für $(x, y) \in E$ setze $(x, y)^{-1} = (y, x)$, ebenso $E^{-1} = \{e^{-1} \mid e \in E\}$.
- Zu $v \in V$ setze

$$\begin{aligned} N^+(v) &= \{w \in V \mid (v, w) \in E\} \\ N^-(v) &= \{w \in V \mid (w, v) \in E\} \end{aligned}$$

  die Menge von *out-Nachbaren* bzw. *in-Nachbaren* von v.
- Ein *Weg* ist eine Knotenfolge $x_0, \ldots, x_l$ mit $(x_{i-1}, x_i) \in E$ für alle $i \in [l]$
- *Pfade* sind Wege ohne doppelte Kanten
- Sei eine *Kapazitätsfunktion* $c \colon E \to \mathbb{R}_0^+$ gegeben. Seien $s, t \in V$. Eine Funktion $f \colon E \to \mathbb{R}_0^+$ heißt *s-t-Fluss* (in G bezüglich der Kapazität c), wenn folgende Bedingungen erfüllt sind:
  1. *Kirchhoff-Bedingung*: Für alle $v \in V \setminus \{s, t\}$ gilt
$$\sum_{z \in N^+(v)} f(v, z) = \sum_{z \in N^-(v)} f(z, v)$$
  2. Für alle $e \in E$ gilt $f(e) \leq c(e)$
- Für $x \in V$ sei der *Nettofluss* von x
$$\partial f(x) = \sum_{y \in N^+(x)} f(x, y) - \sum_{y \in N^-(x)} f(y, x)$$

LEMMA: $\partial f(s) = -\partial f(t)$

BEWEIS: In der Übung.

DEFINITIONEN: $|f| = \partial f(s)$ heißt *Wert* des Flusses. Zu einer Instanz $I = (V, E, c, s, t)$ heißt ein s-t-Fluss f *maximal*, wenn $|f|$ maximal ist unter allen s-t-Flüssen. Den Wert eines maximalen Flusses von I bezeichnen wir mit $f_{\max}(I)$ oder $f_{\max}$, wenn I klar ist.

PROBLEM **Maximaler Fluss**. Eingabe: $I = (V, E, c, s, t)$. Gesucht: s-t-Fluss mit $|f|$ maximal.

LEMMA: Es existieren maximale Flüsse.

BEWEIS: In der Übung. [14]

DEFINITION: Zu $S \subseteq V$ sei

$$\delta(S) = \{(x, y) \mid x \in S,\ y \in V \setminus S,\ (x, y) \in E\}$$

Ist $s \in S$ und $t \notin S$, so heißt $\delta(S)$ (oder auch S) ein *s-t-Schnitt*. Der *Nettofluss* von einem s-t-Schnitt ist

$$\partial f(\delta(S)) = \sum_{e \in \delta(S)} f(e) - \sum_{e \in \delta(V \setminus S)} f(e)$$

Die *Kapazität* eines solchen Schnittes ist $c(\delta(S)) = \sum_{e \in \delta(S)} c(e)$

LEMMA: Für alle s-t-Schnitte $S \subseteq V$ gilt $\partial f(\delta(S)) = |f|$

BEWEIS: In der Übung.

LEMMA: $f_{\max} \leq c_{\min} = \min_S c(\delta(S))$

BEWEIS: Sei f ein s-t-Fluss mit $|f| = f_{\max}$ und $\delta(S)$ ein s-t-Schnitt mit $c(\delta(S)) = c_{\min}$. Dann gilt

$$\begin{aligned} f_{\max} = |f| = \partial f(\delta(S)) &= \sum_{e \in \delta(S)} f(e) - \sum_{e \in \delta(V \setminus S)} f(e) \\ &\leq \sum_{e \in \delta(S)} f(e) \leq \sum_{e \in \delta(S)} c(e) \\ &= c(\delta(S)) = c_{\min} \end{aligned}$$

[14] Tipp: stetige Funktionen, kompakte Mengen

9.2 Algorithmus von Ford und Fulkerson

ANNAHME: Im folgenden gelte immer, dass aus $(x, y) \in E$ schon $(y, x) \notin E$ folgt.

DEFINITIONEN: Sei $P = x_1, \ldots, x_k$ ein (ungerichteter) s-t-Pfad in G. Dann nennen wir eine Kante $(x_i, x_{i+1}) \in E$ *Vorwärtskante* von P und eine Kante $(x_{i+1}, x_i) \in E$ *Rückwärtskante*. Gilt für alle Knoten von P

- $f(e) < c(e)$, falls e Vorwärtskante
- $f(e) > 0$, falls e Rückwärtskante

dann heißt P *f-augmentierender* Pfad. Für alle $e \in E$, die auf P liegen setze

$$\varepsilon_e = \begin{cases} c(e) - f(e) & \text{falls } e \text{ Vorwärtskante} \\ f(e) & \text{sonst} \end{cases}$$

Setze $\varepsilon_P = \min_{e \in E(P)} \varepsilon_e$.

BEMERKUNG: P ist f-augmentierend genau dann, wenn $\varepsilon_P > 0$ ist.

LEMMA: Sei $f_P \colon E \to \mathbb{R}$ definiert durch

$$f_P(e) = \begin{cases} 1 & \text{wenn } e \text{ Vorwärtskante von } P \\ -1 & \text{wenn } e \text{ Rückwärtskante von } P \\ 0 & \text{sonst} \end{cases}$$

Setze $f^* = f + \varepsilon_P f_P$. Dann ist f^* ein s-t-Fluss und es gilt $|f^*| = |f| + \varepsilon_P$.

DEFINITION: Zu einem s-t-Fluss f sei der *Residualgraph* $G_f = (V, E_f)$ mit $E_f = E_f^+ \cup E_f^-$, wobei

$$\begin{aligned} E_f^+ &= \{e \in E \mid f(e) < c(e)\} \\ E_f^- &= \left\{e \in E^{-1} \;\middle|\; f(e^{-1}) > 0\right\} \end{aligned}$$

BEMERKUNGEN:

- Ein (gerichteter) s-t-Pfad in G_f ist ein augmentierender Pfad in G und umgekehrt.
- In Zeit $O(m)$ kann ein augmentierender Pfad in G berechnet werden oder seine Nicht-Existenz bewiesen werden.

Algorithmus von Ford-Fulkerson. Eingabe: $I = (G = (V, E), c, s, t)$. Ausgabe: s-t-Fluss f mit $|f| = f_{\max}$

1. Setze $f \equiv 0$
2. Bestimme G_f. Führe in G_f von s ausgehend eine Graphensuche durch.
   - Gibt es keinen s-t-Pfad $\Rightarrow$ Abbruch.
   - Gibt es einen s-t-Pfad P, so setze $f = f + \varepsilon_P f_P$ und gehe zu 2.

LEMMA: Bei rationalen Kantenkapazitäten terminiert der Algorithmus von Ford-Fulkerson.

BEWEIS: Sind alle Kapazitäten rational, so existiert $N \in \mathbb{N}$ mit $N \cdot c(e) \in \mathbb{N}_0$ für alle $e \in E$. Im Algorithmus von Ford-Fulkerson gilt die Invariante, dass alle $f(e)$ auch Vielfache von $\frac{1}{N}$ sind. Dies ist klar zu Beginn des Algorithmus. Im Schritt 2 ist damit ε_p ein Vielfaches von $\frac{1}{N}$, also wieder $f(e)$ ein solches für alle $e \in E$. Damit erhöht sich der Flusswert im Schritt 2 jeweils um mindestens $\frac{1}{N}$. Also terminiert der Algorithmus nach höchstens $N \cdot f_{\max}$ Schritten.

SATZ: Der Algorithmus von Ford-Fulkerson liefert einen maximalen Fluss, falls er terminiert.

BEWEIS: Wenn der Algorithmus terminiert, gibt es keine f-augmentierende s-t-Pfade mehr. Sei $U \subseteq V$ die Menge aller Knoten x, die auf einem f-augmentierenden s-x-Pfad zu erreichen sind. Dann ist $s \in U$ und $t \notin U$. Für $e \in \delta(U)$ gilt $f(e) = c(e)$, ebenso gilt $f(e) = 0$ für alle $e \in \delta(V \setminus S)$. Damit gilt

$$\begin{aligned} |f| = \partial f(\delta(U)) &= \sum_{e \in \delta(U)} f(e) - \sum_{e \in \delta(V \setminus S)} f(e) \\ &= \sum_{e \in \delta(U)} c(e) = c(\delta(U)) \geq c_{\min} \end{aligned}$$

SATZ: (Max-Flow-Min-Cut-Theorem) $f_{\max} = c_{\min}$

BEMERKUNGEN:

- Ford-Fulkerson berechnet auch einen minimalen Schnitt, nämlich $\delta(U)$ von oben.
- Das Argument „c rational“ $\Rightarrow$ „Ford-Fulkerson terminiert“ liefert auch: Sind alle Kapazitäten ganzzahlig, so ist auch der von Ford-Fulkerson berechnete Fluss ganzzahlig.

9.3 Algorithmus von Edmonds und Karp

ZIEL: Ein Algorithmus für das Max-Flow-Problem mit der Laufzeit $O(nm^2)$, auch für irrationale Kapazitäten. Gesucht ist eine Strategie, nach der die f-augmentierenden Pfade zu wählen sind.

DEFINITION: Sei $I = (V, E, c, s, t)$ und $H = (V, F)$ ein Graph auf V.

- Sei $\mu(H) = d_H(s, t)$.
- Sei $E_\mu(H)$ die Menge aller Kanten $e \in F$, so dass es einen s-t-Pfad mit Länge $\mu(H)$ gibt, der e enthält.

LEMMA: Sei $I = (V, E, c, s, t)$; $G = (V, E), G' = (V, E \cup E_\mu(G)^{-1})$. Dann gilt:

- $\mu(G) = \mu(G')$.
- $E_\mu(G) = E_\mu(G')$.

BEWEIS: In der Übung.

Algorithmus von Edmonds und Karp, 1972. Eingabe: $I = (G, c, s, t)$. Ausgabe: s-t-Fluss f in I mit $|f| = f_{max}(I)$.

1. Initialisiere $f \equiv 0$.
2. Bestimme G_f. Führe in G_f von s ausgehend eine Breitensuche durch. Dies liefert einen kürzesten s-t-Pfad, falls einer existiert.
   (a) falls keiner existiert $\Rightarrow$ Abbruch
   (b) existiert ein kürzester s-t-Pfad P in G_f, so bestimme ε_P bezüglich des durch P induzierten f-augmentierenden Pfades.
3. Setze $f := f + \varepsilon_P f_P$ und gehe zu 2.

SATZ: Der Algorithmus von Edmonds und Karp bestimmt einen maximalen s-t-Fluss in Zeit $O(nm^2)$. Die Aussage bleibt richtig, auch wenn die Kapazitäten nicht alle rational sind.

BEWEIS: Wir zeigen:

(*) Es gibt nach $O(nm)$ Iterationen keinen f-augmentierenden Pfad mehr.

Wir wissen, dass f dann maximal ist. Eine Iteration kostet $O(m)$, daraus folgt die Laufzeit $O(nm^2)$.

Zum Beweis von (*): Sei G_f für irgendeine Iteration gegeben und P in G_f ein kürzester s-t-Pfad. Sei $G_f' = (V, E_f \cup E_\mu(G_f)^{-1})$. Sei f^* der entlang P augmentierte s-t-Fluss. *Behauptung:* G_{f^*} ist ein Subgraph von G_f'. *Beweis:* f^* unterscheidet sich von f nur entlang von P. Damit unterscheidet sich G_{f^*} von G_f nur in Kanten, die vorwärts oder rückwärts auf P liegen. Da $P \subseteq E_f$ und $P^{-1} \subseteq E_\mu(G_f)^{-1}$, folgt die Behauptung. Es folgt: $\mu(G_{f^*}) \geq \mu(G_f') = \mu(G_f)$.

Wir zeigen nun: solange die μ-Werte nicht strikt steigen, nehmen die E_μ-Mengen strikt ab. Setze also voraus, dass $\mu(G_{f^*}) = \mu(G_f)$. Dann ist auch $\mu(G_{f^*}) = \mu(G_f')$. Da G_{f^*} ein Subgraph von G_f' ist und wegen Lemma gilt somit: $E_\mu(G_{f^*}) \subseteq E_\mu(G_f') = E_\mu(G_f)$. Es gibt auf P eine Kante e mit $\varepsilon_e = \varepsilon_P$. Diese Kante kommt in G_{f^*} nicht mehr vor. Da $P \subseteq E_\mu(G_f)$ ist, enthält $E_\mu(G_f)$ eine Kante, die nicht in G_{f^*} ist. Es folgt: $E_\mu(G_{f^*}) \subsetneq E_\mu(G_f)$.

9.4 Zusammenhang

DEFINITIONEN:

- Ein Graph G heißt *k-kantenzusammenhängend*, falls $|V| \geq 2$ und $G - F$ für alle $F \subseteq E$ mit $|F| \leq k-1$ zusammenhängend ist.
- Sei die *Zusammenhangszahl*

$$\kappa(G) = \max\{\, k \in \mathbb{N}_0 \mid G \text{ ist } k\text{-zusammenhängend} \,\}$$

- Sei die *Kantenzusammenhangszahl*

$$\lambda(G) = \max\{\, k \in \mathbb{N}_0 \mid G \text{ ist } k\text{-kantenzusammenhängend} \,\}$$

SATZ: (Menger, 1932) Sei $G = (V, E)$ ein Graph, seien $s, t \in V$. Dann gilt:

- Die kleinste Zahl von Kanten, die s und t trennen, ist gleich der maximalen Zahl kantendisjunkter s-t-Pfade in G.
- Seien s und t nicht benachbart. Die kleinste Zahl von Knoten, die s und t trennen, ist gleich der maximalen Anzahl knotendisjunkter s-t-Pfade in G.

BEWEIS:

1. Sei λ die Kardinalität einer kardinalitätsminimalen s-t-trennenden Kantenmenge und p die Kardinalität einer kardinalitätsmaximalen Menge kantendisjunkter s-t-Pfade. Offenbar gilt $\lambda \geq p$. Wir definieren eine Instanz $I = (V, \vec{E}, c, s, t)$ des Flussproblems durch

$$\begin{aligned} \forall x \in V: &\quad (s,x) \in \vec{E} \iff \{s,x\} \in E \\ \forall x \in V: &\quad (x,t) \in \vec{E} \iff \{x,t\} \in E \\ \forall x,y \in V \setminus \{s,t\}: &\quad (x,y) \in \vec{E} \iff \{x,y\} \in E \\ \forall e \in \vec{E}: &\quad c(e) = 1 \end{aligned}$$

   Offenbar gilt $\lambda = c_{\min}(I)$. Nach Max-Flow-Min-Cut existiert ein ganzzahliger s-t-Fluss f mit $|f| = \lambda$. Nach Aufgabe 11.3 können wir annehmen, dass f Summe von Pfadflüssen ist. Da alle Kapazitäten 1 sind, tritt jeder Pfad in der Summe einmal auf. Folglich existieren λ kantendisjunkte s-t-Pfade.

2. *(Skizze)* Sei I wie in (1). Wir erstellen eine neue Instanz I' wie folgt: Wir ersetzen jeden Knoten $v \in V \setminus \{s,t\}$ durch zwei Knoten v^-, v^+. Eine Kante $(u,v) \in \vec{E}$ wird zu (u^+, v^-) in der neuen Instanz. Ferner fügen wir die Kante (v^-, v^+) für alle $v \in V \setminus \{s,t\}$ hinzu. Dann gilt

$$\begin{aligned} \text{Anzahl der } s\text{-}t\text{-trennenden Knoten} &= c_{\min}(I') = f_{\max}(I') \\ &= \text{Anzahl knotendisjunkter } s\text{-}t\text{-Pfade in } G \end{aligned}$$

SATZ: Die Bahnverbindung Hasselfelde → Gomadingen mit Abfahrt am 01.02.05 um 13:03 und der Voraussetzung, dass man nur Bahn fahren darf, hat die Dauer von 41:54.[15]

[15] Sie ist somit ein Kandidat für die schlechteste Bahnverbindung innerhalb Deutschlands.

10 Minimum-Kosten-Flüsse und Zirkulationen

DEFINITION: Sei $G = (V, E)$ und $k\colon E \to \mathbb{R}$. Für alle $f\colon E \to \mathbb{R}$ setze $k(f) = \sum_{e \in E} f(e)k(e)$. Wir nennen $k(f)$ die *Kosten* des Pseudoflusses f.

BEMERKUNG: Offenbar gilt für alle $f, g\colon E \to \mathbb{R}$ und $\lambda \in \mathbb{R}$:

$$\begin{aligned} k(f+g) &= k(f) + k(g) \\ k(\lambda f) &= \lambda k(f) \end{aligned}$$

PROBLEM **Min-Cost-Flow** (MCF)**.** Eingabe: Instanz $I = (V, E, c, s, t)$, Kostenfunktion $k\colon E \to \mathbb{R}$, Nachfrage $\phi \in \mathbb{R}$. Gesucht: Ein s-t-Fluss f für I mit $|f| = \phi$, der unter allen s-t-Flüssen mit Wert ϕ minimale Kosten hat.

Flüsse können nach Einführen einer Kante (t, s) mit unendlicher Kapazität auf Zirkulationen zurückgeführt werden. Deswegen betrachten wir im folgenden Minimum-Kosten-Zirkulationen.

DEFINITIONEN:

- Sei $G = (V, E)$ gerichteter Graph. Eine Funktion $f : E \to \mathbb{R}$ heißt eine *Zirkulation*, wenn $\partial f(v) = 0$ für alle $v \in V$.
- Seien $d\colon E \to \mathbb{R}$ und $c\colon E \to \mathbb{R}$. Eine Zirkulation $f\colon E \to \mathbb{R}$ heißt *zulässig* bezüglich d und c wenn für alle $e \in E$ gilt: $d(e) \leq f(e) \leq c(e)$.

(10.A) BEMERKUNG: Eine zulässige Zirkulation kann durch Lösen einer geeigneten Max-Flow-Instanz gefunden werden, oder deren Nichtexistenz bewiesen werden - hier wird das Verfahren nicht gezeigt.

PROBLEM **Min-Cost-Circulation** (MCC)**.** Eingabe: Eine Instanz $I = (V, E, d, c, k)$. Gesucht: Zulässige Zirkulation, die unter allen solchen minimale Kosten hat.

Eine Instanz sei von nun an ein Vektor $(G = (V, E), d, c, k)$. Wir nehmen an, dass wir irgendeine zulässige Zirkulation f kennen, und versuchen sie zu verbessern.

(10.5) LEMMA: Sei f eine nichtnegative Zirkulation. Sei $e_0 \in E$, so dass $f(e_0) > 0$. Dann gibt es in G einen Kreis C mit $e_0 \in C$ und $f(e_0) > 0$ für alle $e \in C$.

DEFINITION: Für eine Menge $F \subseteq E$ definiere deren charakteristische Funktion: $\chi^F\colon E \to \{0, 1\}$ durch $\chi(e) = 1$ falls $e \in F$ und $\chi(e) = 0$ sonst.

(10.7) LEMMA: Sei f eine nichtnegative Zirkulation. Dann gibt es nichtnegative reelle Zahlen $\lambda_1, \ldots, \lambda_r$ und Kreise $C_1, \ldots, C_r$, so dass gilt

$$f = \sum_{i=1}^{r} \lambda_i \chi^{C_i}$$

BEWEIS: Folgt induktiv aus dem Lemma (10.5). □

VORAUSSETZUNG: Wir nehmen an, dass unsere Instanz $E \cap E^{-1} = \emptyset$ erfüllt. Wir definieren $\bar{E} = E \cup E^{-1}$ und $\bar{G} = (V, \bar{E})$. Sei weiter die *Längenfunktion* $l\colon \bar{E} \to \mathbb{R}$ mit

$$l(e) = \begin{cases} k(e) & \text{falls } e \in E \\ -k(e^{-1}) & \text{falls } e \in E^{-1} \end{cases}$$

DEFINITION: Wir definieren den *Residualgraphen* $G_f = (V, E_f)$ zu G durch $E_f = E_f^+ \cup E_f^-$, wobei

$$\begin{aligned} E_f^+ &= \{ e \in E \mid f(e) < c(e) \} \\ E_f^- &= \left\{ e^{-1} \in E^{-1} \;\middle|\; d(e) < f(e) \right\} \end{aligned}$$

DEFINITION: Ein Kreis heißt *einfach*, wenn er keinen Knoten zweimal besucht und wenigstens 3 Kanten durchläuft.

DEFINITION: Sei C ein einfacher Kreis in G_f. Sei $f^C : E \to \{-1, 0, 1\}$ mit

$$f^C(e) = \begin{cases} 1 & \text{falls } e \in E_f^+ \cap C \\ -1 & \text{falls } e^{-1} \in E_f^- \cap C \\ 0 & \text{sonst} \end{cases}$$

Weiter sei

$$\begin{aligned} \varepsilon_f^C = \min \big(& \left\{ c(e) - f(e) \mid e \in E_f^+ \cap C \right\} \\ & \cup \left\{ f(e) - d(e) \mid e^{-1} \in E_f^- \cap C \right\} \big) \end{aligned}$$

BEMERKUNGEN:

- f^C ist wohldefiniert wegen der Einfachheit von C.
- Es ist $\varepsilon_f^C > 0$ aufgrund der Definition von G_f.

DEFINITION: Die *Länge* eines Kreises C sei $l(C) = \sum_{l \in C} l(e)$. Mit $|C|$ bezeichne die Anzahl der durchlaufenen Kanten, d.h. sind $v_1, ..., v_r$ die Knoten des Kreises, so ist $|C| = r$.

(10.12) LEMMA: Sei C ein einfacher Kreis in G_f. Dann gilt:

- $\widetilde{f} = f + \varepsilon_f^C f^C$ ist eine zulässige Zirkulation.
- $k(\widetilde{f}) = k(f) + \varepsilon_f^C l(C)$.

LEMMA: Es sind äquivalent:

1. f ist optimal.
2. G_f enthält keine Kreise negativer Länge.

BEWEIS:

(1) $\Rightarrow$ (2) folgt aus (10.12)

(2) $\Rightarrow$ (1) Sei g eine zulässige Zirkulation in der Instanz. Sei $h = g - f$. Dann ist h eine Zirkulation. Auf G_f definieren wir

$$h' \colon E_f \to \mathbb{R},\ e \mapsto \begin{cases} h(e) & \text{falls } e \in E_f^+, h(e) > 0 \\ -h(e) & \text{falls } e \in E_f^-, h(e) < 0 \\ 0 & \text{sonst} \end{cases}$$

Wir zeigen: h' ist eine Zirkulation in G_f. Seien

$$\begin{aligned} E^> &= \{e \in E \mid h(e) > 0\} \\ E^< &= \{e \in E \mid h(e) < 0\} \end{aligned}$$

Mit $\delta_f^+(v)$ bzw. $\delta_f^-(v)$ bezeichnen wir die Menge der Out- bzw. In-Kanten eines Knotens v in G_f. Es gelten folgende Beobachtungen:

1. $E^> \subseteq E_f^+$
2. $(E^<)^{-1} \subseteq E_f^-$
3. Für alle $A \subseteq E$ gilt

$$\sum_{e \in A \cap E^>} h(e) = \sum_{e \in A \cap E_f^+} h'(e)$$

4. Für alle $A \subseteq E$ gilt

$$\sum_{e \in A \cap E^<} h(e) = - \sum_{e \in A^{-1} \cap E_f^-} h'(e)$$

5. Für alle $v \in V$ gilt

$$\begin{aligned}
\delta^+(v) \cap E_f^+ &= \delta_f^+(v) \cap E_f^+ \\
\delta^-(v) \cap E_f^+ &= \delta_f^-(v) \cap E_f^+ \\
(\delta^+(v))^{-1} \cap E_f^- &= \delta_f^-(v) \cap E_f^- \\
(\delta^-(v))^{-1} \cap E_f^- &= \delta_f^+(v) \cap E_f^-
\end{aligned}$$

Sei nun $v \in V$. Es gilt dann

$$\begin{aligned}
0 &= \sum_{e \in \delta^+(v)} h(e) - \sum_{e \in \delta^-(v)} h(e) \\
&= \sum_{e \in \delta^+(v) \cap E^>} h(e) + \sum_{e \in \delta^+(v) \cap E^<} h(e) - \sum_{e \in \delta^-(v) \cap E^>} h(e) - \sum_{e \in \delta^-(v) \cap E^<} h(e) \\
&\overset{(3),(4)}{=} \sum_{e \in \delta^+(v) \cap E_f^+} h'(e) - \sum_{e \in (\delta^+(v))^{-1} \cap E_f^-} h'(e) - \sum_{e \in \delta^-(v) \cap E_f^+} h'(e) + \sum_{e \in (\delta^-(v))^{-1} \cap E_f^-} h'(e) \\
&\overset{(5)}{=} \sum_{e \in \delta_f^+(v) \cap E_f^+} h'(e) - \sum_{e \in \delta_f^-(v) \cap E_f^-} h'(e) - \sum_{e \in \delta_f^-(v) \cap E_f^+} h'(e) + \sum_{e \in \delta_f^+(v) \cap E_f^-} h'(e) \\
&= \sum_{e \in \delta_f^+(v)} h'(e) - \sum_{e \in \delta_f^-(v)} h'(e) \\
&= \partial h'(v)
\end{aligned}$$

Somit ist h' eine Zirkulation, sogar eine nichtnegative. Nach Lemma (10.7) findet man Kreise $C_1, \ldots, C_r$ in G_f sowie nichtnegative reelle Zahlen $\lambda_1, \ldots, \lambda_r$, so dass

$$h' = \sum_{i=1}^{r} \lambda_i \chi^{C_i}$$

Nach Definition von h' können $C_1, \ldots, C_r$ als einfache Kreise gewählt werden. Man kann zeigen, dass für alle $e \in E$ gilt

$$h(e) = \sum_{i=1}^{r} \lambda_i f^{C_i}(e)$$

Mit der Linearität von k folgt nun

$$\begin{aligned} k(g) - k(f) = k(g-f) &= k(h) \\ &= k\left(\sum_{i=1}^{r} \lambda_i f^{C_i}\right) = \sum_{i=1}^{r} \lambda_i l(C_i) \geq 0 \end{aligned}$$

Somit ist $k(g) \geq k(f)$, also f optimal. □

Um MCC zu lösen müssen wir also in geeigneter Reihenfolge entlang negativer Kreise augmentieren, bis es keine solche mehr gibt. Für die Effizienz brauchen wir die Strategie, nach der die Kreise zu wählen sind.

(10.16) LEMMA: Seien f und $\tilde{f}$ zulässig und $e_0 = (v_0, w_0)$ mit $f(e_0) > \tilde{f}(e_0)$. Dann liegt e_0 auf einem einfachen Kreis C in $G_{\tilde{f}}$. Außerdem ist dann C^{-1} ein Kreis in G_f.

BEWEIS: Sei $g = f - \tilde{f}$, dann ist g eine Zirkulation. Für alle $e \in E$ gilt:

$$g(e) > 0 \Rightarrow e \in E_{\tilde{f}}^{+} \quad \text{und} \quad g(e) < 0 \Rightarrow e^{-1} \in E_{\tilde{f}}^{-} \qquad (+)$$

Sei $G'_{\tilde{f}}$ der Subgraph von $G_{\tilde{f}}$, der nur aus Kanten besteht, für die $g(e) \neq 0$ bzw. $g(e^{-1}) \neq 0$ gilt und der e_0^{-1} nicht enthält. Sei T die Menge aller Knoten, die in $G'_{\tilde{f}}$ von w_0 aus erreichbar sind. Wegen der Kirchhoffbedingung gilt

$$g(\delta(T)) = g(\delta^{-1}(T)) \qquad (*)$$

In $G'_{\tilde{f}}$ verlässt keine Kante die Menge T. Mit $(+)$ ergibt das $g(e) \leq 0$ für alle $e \in \delta(T)$. Ebenso folgt aus $(+)$, dass $g(e) \geq 0$ für alle $e \in \delta^{-1}(T)$ ist mit $e \neq e_0$, außerdem gilt bereits $g(e_0) > 0$ nach Voraussetzung. Mit $(*)$ folgt $g(e) = 0$ für alle $e \in \delta^{-1}(T)$. Da $g(v_0, w_0) > 0$ und $w_0 \in T$, folgt $v_0 \in T$. Also gibt es in $G'_{\tilde{f}}$ einen Kreis, der (v_0, w_0) enthält. Dieser ist auch ein Kreis in $G_{\tilde{f}}$. Da $e^{-1} \notin G'_{\tilde{f}}$, kann C einfach gewählt werden.

Der Beweis dafür, dass C^{-1} ein Kreis in G_f ist, wird dem Leser als Übung überlassen.

10.1 Allgemeine Längen und Potentiale

In diesem Abschnitt sei G ein gerichteter Graph und $l\colon E \to \mathbb{R}$ eine Länge.

DEFINITION: Sei C ein Kreis in G. Die *Durchschnittslänge* von C sei $\frac{l(C)}{|C|}$.

(10.17) Lemma: — fehlt —

(10.18) Satz: (Resultat von (10.17)) Ein Kreis mit minimaler Durchschnittslänge kann in Zeit $O(mn)$ gefunden werden. Der Kreis kann als einfacher Kreis gewählt werden, wenn er drei Kanten oder mehr durchläuft.

(10.19) Definitionen:

- Eine Funktion $p\colon V \to \mathbb{R}$ heißt *Potential* (bezüglich l), wenn für alle $(u,v) \in E$ gilt $l(u,v) \geq p(v) - p(u)$
- Für eine Funktion $p\colon V \to \mathbb{R}$ heißt die durch

$$l_p(u,v) = l(u,v) - p(v) - p(u)$$

  definierte Funktion $E \to \mathbb{R}$ die (bezüglich p und l) *reduzierte Länge*.
- Für alle $\varepsilon \in \mathbb{R}$ definiere die Länge l^ε durch $l^\varepsilon(e) = l(e) + \varepsilon$.

(10.20) Lemma: Sei $p\colon V \to \mathbb{R}$. Dann gilt:

1. p ist Potential für l genau dann, wenn $l_p(e) \geq 0$ für alle $e \in E$
2. p ist Potential für l^ε genau dann, wenn $l_p(e) \geq -\varepsilon$ für alle $e \in E$
3. Für alle Kreise C gilt $l(C) = l_p(C)$
4. Ist l antisymmetrisch (d.h. $l(e) = -l(e^{-1})$ für alle $e \in E$ mit $e^{-1} \in E$), so ist auch l_p antisymmetrisch.

Beweis: Nachrechnen.

(10.21) Lemma: Es sind äquivalent:

1. G hat keine Kreise negativer Länge.
2. Es gibt ein Potential für l.

Beweis:

(1) $\Rightarrow$ (2) Übung

(2) $\Rightarrow$ (1) Folgt aus (2) und (3) in (10.20)

10.2 Algorithmus von Goldberg und Tarjan

Wir fassen die Optimalitätskriterien zusammen:

(10.22) SATZ: Sei f eine zulässige Zirkulation. Es sind äquivalent:

1. f ist optimal.
2. G_f hat keine Kreise negativer Länge.
3. G_f hat keine einfachen Kreise negativer Länge.
4. G_f hat ein Potential.

Algorithmus von Goldberg und Tarjan (1988-89)

1. Initialisiere f mit einer zulässigen Zirkulation. Gibt es keine $\Rightarrow$ Abbruch.
2. Berechne G_f
3. Finde in G_f einen einfachen Kreis C minimaler Durchschnittslänge.
4. Ist $l(C) \geq 0 \Rightarrow$ Abbruch, Ausgabe von f
   Ist $l(C) < 0$, so setze $f = f + \varepsilon_f^C f^C$
5. Gehe zu (2)

Nach (10.22) Liefert der Algorithmus bei Terminierung eine optimale Zirkulation. Jede Iteration kostet $O(mn)$. Unser Ziel wird es nun sein zu zeigen, dass der Algorithmus maximal $m^2 n \log(2n)$ Iterationen braucht.

(10.24) DEFINITIONEN:

- Sei $\varepsilon \geq 0$. Wir nennen f ε-optimal, wenn G_f für l^ε ein Potential hat.
- Sei
$$\varepsilon(f) = \inf \{ \varepsilon \geq 0 \mid f \text{ ist } \varepsilon\text{-optimal} \} \qquad (*)$$
- Sei
$$\mu(f) = \min \left\{ \frac{l(C)}{|C|} \;\middle|\; C \text{ ist einfacher Kreis in } G_f \right\} \cup \{0\}$$

(10.25) LEMMA:

1. f ist genau dann optimal, wenn $\mu(f) = 0$
2. f ist genau dann optimal, wenn $\varepsilon(f) = 0$
3. Es gilt $\mu(f) = -\varepsilon(f)$
4. f ist $\varepsilon(f)$-optimal, insbesondere ist das Infimum in (*) ein Minimum

BEWEIS:

(1, 2) Klar nach (10.22)

(3, 4) Als Übung

(10.26) LEMMA:

1. $\tilde{f}$ entstehe aus f durch eine Iteration des Algorithmus. Dann gilt
$$\varepsilon(\tilde{f}) \leq \varepsilon(f)$$
2. $\tilde{f}$ entstehe durch m Iterationen aus f. Dann gilt
$$\varepsilon(\tilde{f}) \leq \left(1 - \frac{1}{n}\right) \varepsilon(f)$$

BEWEIS:

1. Wir schreiben μ für $\mu(f)$ und ε für $\varepsilon(f)$. Sei C der Kreis, der beim Übergang von f zu $\tilde{f}$ benutzt wird. Sei p ein Potential für l^ε. Nach (3) in (10.20) gilt
$$0 = \mu + \varepsilon = \frac{l(C)}{|C|} + \varepsilon = \frac{l_p(C)}{|C|} + \varepsilon = \frac{1}{|C|} \sum_{e \in C} l_p^\varepsilon(e) \geq 0$$
Es folgt: $l_p^\varepsilon(e) = 0$ (also $l_p(e) = -\varepsilon$) für alle $e \in C$. Wir zeigen nun: p ist auch Potential für $G_{\tilde{f}}$ bezüglich l^ε. Für $e \in E_f \cap E_{\tilde{f}}$ ist nichts zu tun. Sei $e \in E_{\tilde{f}} \setminus E_f$. Dann ist $e^{-1} \in C$ und es gilt
$$l_p^\varepsilon(e) = l_p(e) + \varepsilon = -\underbrace{l_p(e^{-1})}_{=-\varepsilon} + \varepsilon = 2\varepsilon \geq 0$$

2. Sei p ein Potential für l^ε auf G_f und seien $C_1, \ldots, C_m$ die Kreise, die beim Übergang von f nach $\tilde{f}$ bearbeitet wurden. Sei j minimal unter der Bedingung, dass $e_0 \in C_j$ existiert mit $l_p(e_0) \geq 0$. So ein j existiert, denn: Angenommen $l_p(e) < 0$ für alle $e \in C_i, i \in [m]$. Dann kommt in jeder Iteration eine Kante mit positivem l_p hinzu (wegen der Antisymmetrie von l_p) und eine Kante mit negativem l_p wird verbraucht. Es gibt in G_f maximal m Kanten mit negativem l_p. Es folgt, dass C_m aus nur noch zwei Kanten besteht, ein Widerspruch, da C_m einfach ist.

   Für alle $e \in C_1$ gilt $l_p(e) = -\varepsilon$, folglich ist $1 < j \leq m$. Die Kreise $C_1, \ldots, C_{j-1}$ haben nur Kanten mit negativen l_p. Es kommen durch die entsprechenden Iterationen nur Kanten mit positiven l_p hinzu. Für alle $e \in C_j$ gilt daher $e \in E_f$, falls $l_p(e) < 0$. Nach der Wahl von p (siehe (10.20)(2)) folgt: für alle $e \in C_j$ ist $l_p(e) \geq -\varepsilon$. Betrachte die j-te Iteration. Sei f' durch die vorige Iteration entstanden. Es gilt

$$\begin{aligned} -\varepsilon(f') = \mu(f') &= \frac{l(C_j)}{|C_j|} = \frac{l_p(C_j)}{|C_j|} \\ &\geq \frac{1}{|C_j|} \sum_{e \in C_j \setminus \{e_0\}} l_p(e) \\ &\geq \frac{1}{|C_j|}(|C_j| - 1)(-\varepsilon) = -\left(1 - \frac{1}{|C_j|}\right)\varepsilon \\ &\geq -\left(1 - \frac{1}{n}\right)\varepsilon \end{aligned}$$

   Mit (1) gilt also

$$\varepsilon(\tilde{f}) \leq \varepsilon(f') \leq \left(1 - \frac{1}{n}\right)\varepsilon(f)$$

(10.27) DEFINITIONEN:

- Eine Kante $e \in E$ heißt ε-*fest*, wenn für alle ε-optimalen Zirkulationen $f, \tilde{f}$ gilt $f(e) = \tilde{f}(e)$.
- Eine Kante $e \in E^{-1}$ heißt ε-*fest*, wenn e^{-1} dies ist.
- Wir sagen, der Algorithmus *fixiert* eine Kante $e \in E$ im Schritt j, wenn der f-Wert von e nach der j-ten Iteration nicht mehr verändert wird.
- Wir sagen, der Algorithmus *fixiert* eine Kante $e \in E^{-1}$ im Schritt j, wenn er dies mit e^{-1} tut.

(10.28) LEMMA: Sei f zulässig, $\varepsilon = \varepsilon(f)$ und p ein Potential für l^ε auf G_f. Sei $e_0 \in \bar{E}$, so dass $l_p(e_0) < -2n\varepsilon$. Dann ist e_0 ε-fest.

BEWEIS: Angenommen e_0 ist nicht ε-fest. Dann gibt es eine zulässige ε-optimale Zirkulation $\tilde{f}$, so dass $f(e) \neq \tilde{f}(e_0)$. Betrachte zwei Fälle:

Fall 1. Es ist $e_0 \in E$. Wegen $l_p(e_0) < -2n\varepsilon$ ist insbesondere $l_p(e_0) < -\varepsilon$ und somit $e_0 \notin E_f$. Es folgt $f(e_0) = c(e_0)$, also $\tilde{f}(e_0) \neq c(e_0)$, woraus $\tilde{f}(e_0) < c(e_0) = f(e_0)$ folgt. Nach (10.16) liegt e_0 auf einem einfachen Kreis C in $G_{\tilde{f}}$. Es gilt

$$\mu(\tilde{f}) \leq \frac{l(C)}{|C|} \tag{1}$$

Ebenfalls nach (10.16) ist C^{-1} ein Kreis in G_f. Daher gilt $l_p(e) \geq -\varepsilon$ für alle $e \in C^{-1}$, wegen der Antisymmetrie also

$$\forall e \in C\colon \quad l_p(e) \leq \varepsilon \tag{2}$$

Fall 2. Es ist $e_0 \in E^{-1}$. Ähnlich wie im Fall 1 findet man einen Kreis C in $G_{\tilde{f}}$, für den (1) und (2) gelten.

Da $\tilde{f}$ ε-optimal ist, gilt $\varepsilon(\tilde{f}) \leq \varepsilon$. Setze $L = |C|$ und fasse zusammen:

$$\begin{aligned}
-\varepsilon \leq -\varepsilon(\tilde{f}) = \mu(\tilde{f}) \quad &\overset{(1)}{\leq} \quad \frac{l(C)}{L} = \frac{l_p(C)}{L} \\
&= \quad \frac{1}{L}\left(l_p(e_0) + \sum_{e \in C \setminus \{e_0\}} l_p(e)\right) \\
&\overset{(2)+\text{Vor.}}{<} \quad \frac{1}{L}\left(-2n\varepsilon + (L-1)\varepsilon\right) \leq \frac{1}{L}(-L\varepsilon) \\
&= \quad -\varepsilon
\end{aligned}$$

(10.29) LEMMA: $\tilde{f}$ gehe aus f durch $mn\log(2n)$ Iterationen hervor. Dann gibt es eine Kante $e_0 \in E$, die $\varepsilon(\tilde{f})$-fest, aber nicht $\varepsilon(f)$-fest ist. [16]

BEWEIS: Nach (10.26) (2) und elementarer Analysis ist

$$\varepsilon(\tilde{f}) \leq \left(1 - \frac{1}{n}\right)^{n\log(2n)} \varepsilon(f) < e^{-\log(2n)}\varepsilon(f) = \frac{1}{2n}\varepsilon(f)$$

[16] Mit anderen Worten: Spätestens alle $mn\log(2n)$ Iterationen wird eine weitere Kante fixiert.

Also gilt $2n\varepsilon(\tilde{f}) < \varepsilon(f)$. Sei nun C der Kreis, der ausgehend von f als erstes betrachtet wird. Sei $\tilde{p}$ ein Potential für $l^{\varepsilon(\tilde{f})}$ auf $G_{\tilde{f}}$. Dann gilt nach (10.20)(3) und (10.25)(3):

$$\frac{l_{\tilde{p}}(C)}{|C|} = \frac{l(C)}{|C|} = -\varepsilon(f) < -2n\varepsilon(\tilde{f})$$

Ein Mittelwertargument liefert, dass $e_0 \in C$ existiert, so dass $l_{\tilde{p}}(e_0) < -2n\varepsilon(\tilde{f})$ ist. Nach (10.28) ist daher e_0 $\varepsilon(\tilde{f})$-fest. Da $e_0 \in C$ und C in der Iteration ausgehend von f betrachtet wurde, ist e_0 nicht $\varepsilon(f)$-fest.

(10.30) SATZ: Der Algorithmus von Goldberg und Tarjan liefert in Zeit $O(m^3n^2\log(n))$ eine optimale zulässige Zirkulation, falls eine solche existiert. Anderenfalls liefert er nach $O(m^2n)$ eine Meldung über die Nichtexistenz einer zulässigen Zirkulation.

BEWEIS:

- Eine initiale Zirkulation kann durch Lösen einer geeigneten Max-Flow-Instanz gefunden, oder ihre Nichtexistenz nachgewiesen werden (10.A).
- Der Algorithmus benötigt nach (10.29) maximal $m^2n\log(2n)$ Iterationen. Jede Iteration kostet $O(mn)$.
- Die Korrektheit folgt aus (10.22)

11 Selfish Routing

[17] Es gibt zwei Modellen für selfish routing:

- Mit unendlich vielen Spielern (Users). Quellen: Dissertation von Tim Roughgarden (die neueste Homepage ist in Stanford) und andere Arbeiten von Roughgarden und Tardos.
- Mit endlich vielen Spielern. Quellen: Artur Czumaj, B. Vöcking.

DEFINITIONEN: Sei $G = (V, E)$ ein gerichteter Graph, auf dem es N Quelle/Senke Paare $d_i = (s_i, t_i)$ gibt. Solche Paaren nennen wir auch *Demands*. Zu jedem solchen Paar d_i gehört eine Menge n_i von *Users* oder *Spielern*. Für alle $e \in E$ gibt es eine *Latenzfunktion* $l_e \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Üblicherweise sind alle l_e monoton steigend und stetig. Eine Instanz ist $I = (G, d, n, l)$.

DEFINITIONEN: Sei $\mathcal{P}_i$ die Menge aller s_i-t_i-Pfade und $\mathcal{P} = \bigcup_{i \in [N]} \mathcal{P}_i$. Für $e \in E$ sei $\mathcal{P}(e) = \{ P \in \mathcal{P} \mid e \in P \}$

- Eine Funktion $x \colon \mathcal{P} \to \mathbb{R}_{\geq 0}$ heißt ein *Fluss*, wenn

$$\sum_{P \in \mathcal{P}_i} x_p = n_i \quad \forall i \in [N]$$

- Die *Congestion* von e sei

$$x_e = \sum_{P \in \mathcal{P}_e} x_p$$

- Die *Latenz* vom Link e ist $l_e(x_e)$, d.h. es dauert die Zeit $l_e(x_e)x_e$ um alle Daten durch e zu schicken.
- Die *sozialen Kosten* eines Flusses x sind

$$\mathrm{sc}(x) = \sum_{e \in E} l_e(x_e)x_e = \sum_{i \in [N]} \sum_{P \in \mathcal{P}_i} l_p(x)x_p, \quad \text{wobei } l_p(x) = \sum_{e \in P} l_e(x_e)$$

[17] Die Vorlesung vom 08.02.2005 wurde als Ergänzung zur Diskrepanztheorie hinzugefügt, siehe 7.5.3

DEFINITION: x heißt *Nash-Fluss* oder *Nash-Equilibrium (NE)*, wenn gilt:

$$\forall e \in [N]\ \forall Q, R \in \mathcal{P}_i\ \forall \delta \in (0, x_Q]\colon\ l_Q(x) \leq l_R(\tilde{x})$$

wobei

$$\tilde{x}_P = \begin{cases} x_Q - \delta & P = Q \\ x_R + \delta & P = R \\ x_P & \text{sonst} \end{cases}$$

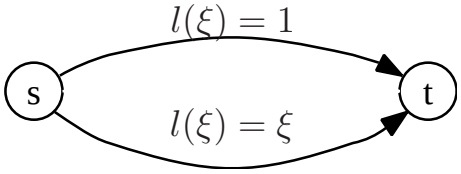
SATZ: x ist ein Nash-Equilibrium genau dann, wenn

$$\forall i \in [N]\ \forall Q, R \in \mathcal{P}_i\colon\ x_Q > 0 \Rightarrow l_Q(x) \leq l_R(x)$$

Man sieht: In einem Nash-Equilibrium haben alle flussführenden Pfade eines Demands dieselbe Latenz, etwa $l_i(x)$. Die Sozialkosten lassen sich dann schreiben als

$$\mathrm{sc}(x) = \sum_{i \in N} l_i(x) n_i$$

BEISPIEL: Betrachte das folgende Netz mit einem Demand und $n = 1$:



Ein Nash-Equilibrium ist $x = (0, 1)$, also 1 durch den unteren Pfad, die Sozialkosten davon sind $\mathrm{sc}(x) = 1$. Wäre $x^* = \left(\frac{1}{2}, \frac{1}{2}\right)$, so wäre $\mathrm{sc}(x^*) = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$. Somit ist ein Nash-Fluss nicht immer optimal.

DEFINITION: Den Unterschied $\mathrm{sc}(x) - \mathrm{sc}(x^*)$, wobei x ein schlechtestes Nash-Equilibrium ist und x^* optimal, nennt man *Price of Anarchy*.

SATZ: Sind die Latenzfunktionen stetig und monoton wachsend, so folgt: Nash-Equilibria existieren und sind essentiell eindeutig, d.h. $l_e(x_e) = l_e(\tilde{x}_e)$ für Nash-Equilibria $x, \tilde{x}$.

BEMERKUNG: Insbesondere ist $l_i(x)$ von x unabhängig, etwa $L_i = l_i(x)$ für irgendein Nash-Equilibrium x. Der Vektor $(L_i)_i$ ist eine für die gegebene Instanz charakteristische Größe.

11.1 Charakterisierung optimaler Flüsse

SATZ: Ein Fluss x^* ist optimal genau dann, wenn x^* ein Nash-Equilibrium ist für $l_e^*(\xi) = l_e(\xi) + \xi l_e'(\xi)$, vorausgesetzt l_e ist differenzierbar und $\xi \to \xi l_e'(\xi)$ ist convex für alle $e \in E$.

BEWEIS: Folgt mit dem KKT-Theorem.

BEISPIEL: Für das Beispiel oben sei auf dem unteren Pfad $l(\xi) = \xi^p$. Es ist oben $l^*(\xi) = 1$, unten ist $l^*(\xi) = (p+1)\xi^p$. Wir rechnen dafür ein Nash-Equilibrium $x^* = (x_1^*, x_2^*)$ aus. Es müsste gelten $x_1^* + x_2^* = 1$. Sei o.B.d.A. $x_2^* > 0$, dann gilt

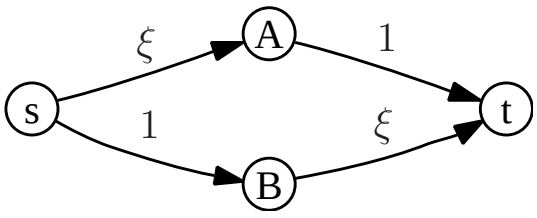
$$\begin{aligned} x_2^* > 0 \implies (p+1)(x_2^*)^p \leq 1 &\implies x_2^* < 1 \implies x_1^* > 0 \\ &\implies (p+1)(x_2^*)^p = 1 \implies x_2^* = (p+1)^{-\frac{1}{p}} \end{aligned}$$

Die erste Implikation gilt, da sonst x^* kein Nash-Equilibrium wäre. Die vierte Implikation gilt, weil alle Latenzen auf Flussführenden Pfaden in einem Nash-Equilibrium gleich sind. Für den Price of Anarchy gilt

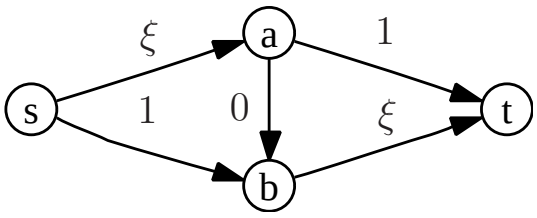
$$\begin{aligned} \rho &= \left(1 - (p+1)^{-\frac{1}{p}} + (p+1)^{-\frac{1}{p}}(p+1)^{-1}\right)^{-1} \\ &= \left(1 + (p+1)^{-\frac{1}{p}}\left((p+1)^{-1} - 1\right)\right)^{-1} \\ &= \left(1 + (p+1)^{-\frac{1}{p}}\left(-(p+1)^{-1}p\right)\right)^{-1} \\ &= \left(1 - (p+1)^{-\frac{1}{p}-1}\right)^{-1} \\ &= \theta\left(\frac{p}{\ln p}\right) \text{ für } p \to \infty \end{aligned}$$

11.2 Paradox von Braess

Betrachte das folgende Netz mit $n = 1$:



Ein Nash-Equilibrium wäre z.B. $x = \left(\frac{1}{2}, \frac{1}{2}\right)$. Es ist $L = \frac{3}{2}$ und $\mathrm{sc}(x) = \frac{3}{2}$. Wir bauen nun eine zusätzliche schnelle Verbindung ein:



Hier wäre $x_p = 1$ auf dem Pfad $P = (s, a, b, t)$ ein Nash-Equilibrium. Dann ist $L = 2$ und $\mathrm{sc}(x) = 2$, es ist also für alle Teilnehmer schlechter geworden.

SATZ: Es ist NP-schwer problematische Kanten zu bestimmen (d.h. solche Kanten, deren Streichen das Nash-Equilibrium verbessert).

BEWEIS: Man kann das Problem 2DDP darauf reduzieren: Finden, ob es disjunkte Pfade in einem Graphen zwischen s_1 und t_1 bzw. s_2 und t_2 gibt.

Index