Näherungsalgorithmen
Vorlesungsskript
entstanden ab 2001, zunächst in Tübingen

Henning Fernau
LS Theoretische Informatik
Universität Trier
email: fernau@informatik.uni-trier.de

LaTeX-nisch in Teilen betreut von
Thomas Bitschnau

Inhaltsverzeichnis

Kurze Einführung

Dies ist das Skriptum zu einer im Wintersemester 2001/2002 zunächst an der Universität Tübingen gehaltenen Vorlesung, die dann wiederholt gehalten wurde, schließlich auch an der Universität Trier. Der Kurs stützt sich — sofern nicht anderes im laufenden Text angegeben wird— auf das Buch

> G. Ausiello and P. Creczenzi and G. Gambosi and V. Kann and A. Marchetti-Spaccamela and M. Protasi: *Complexity and Approximation; Combinatorial Optimization Problems and Their Approximability Properties*, Springer, 1999.

Aufgrund des enzyklopädischen Umfangs dieses Werkes kann in einer zweistündigen Vorlesung natürlich nur ein Bruchteil der Thematik abgehandelt werden. Als ergänzende Lekttüre ist das Buch aber jederzeit empfohlen.

Die Vorlesung gliedert sich in drei große Teile:

- In dem die drei ersten Kapitel umfassenden Einführungsteil wird Grundsätzliches zur P–NP-Problematik und diesbezüglichen Lösungsansätzen gesagt und am Knotenüberdeckungsproblem durchgesprochen. Ein solcher Lösungsansatz sind Approximationsalgorithmen.
- Die Kapitel 4 bis 9 behandeln verschiedene Techniken zum Entwurf von Näherungsalgorithmen.
- Die Kapitel 10 bis 15 betrachten Optimierungsprobleme mehr phänomenologisch: Welche Typen von Näherungsalgorithmen gibt es, und wie "gute" Approximationen sind zu erwarten (sofern man einige komplexitätstheoretische Annahmen trifft)?

Eine Einführung in die Thematik

Der Einführungsteil gliedert sich in drei Abschnitte:

- Im Einführungsteil (im engeren Sinne) werden wir Querbezüge zu anderen, verwandten Themen herstellen und diese dann anhand von einfachen Algorithmen für das Knotenüberdeckungsproblem erläutern.
- Das Knotenüberdeckungsproblem soll uns veranlassen, uns zum Einstieg Überdeckungsprobleme allgemein genauer anzusehen. Dabei werden wir den Satz über lokale Verhältnisse als einen ersten recht allgemeinen Ansatz zum Nachweis der Qualität von Näherungsalgorithmen kennen lernen.
- Im dritten Abschnitt werden wir den eigentlichen Gegenstand der Vorlesung, nämlich Optimierungsprobleme, formal näher untersuchen. Dabei werden wir auch Reduktionen zwischen verschiedenen Problemvarianten besprechen.

1 Einführung

1.1 Motivation

Viele interessante Probleme (aus der Praxis!) sind NP-hart
$\rightsquigarrow$ Es sind wohl keine Polynomzeitalgorithmen sind zu erwarten.

Mögliche Auswege:

- Heuristische Verfahren
  - Ziel: schnelle Laufzeit
  - „hoffentlich“ wird „gute“ Lösung gefunden.
    $\hookrightarrow$ keine „mathematische“ Garantie, nur „Empirie“.
  - typische Beispiele: Greedy-Verfahren
- Randomisierte Verfahren
  - finden optimale Lösung „mit großer Wahrscheinlichkeit“.
    Hinweis: Spezialvorlesung „Randomisierte Algorithmen“ (wird manchmal angeboten)
- Parameterisierte Verfahren
  - finden stets optimale Lösung.
  - versuchen, den nicht-polymiellen Laufzeitanteil auf einen (als klein angenommenen) sogenannten Parameter zu beschränken.
    Hinweis: Spezialvorlesung „Parameterisierte Algorithmen“
- Näherungsverfahren
  - sind „Heuristiken mit Leistungsgarantie“.
  - Güte von Näherungsverfahren kann
    * absolut oder
    * relativ zum Optimum gemessen werden

Da bis auf wenige Ausnahmen (z.B. Knotenfärbung in planaren Graphen wegen des 4-Farbensatzes) das Auffinden von Lösungen, die absoluten Fehlerschranken genügen, meist ebenfalls NP-hart ist, hier zumeist: Beschränkung auf relative Näherungen.

1.2 Das Knotenüberdeckungsproblem — ein einführendes Beispiel

Eine *Knotenüberdeckung* (engl: vertex cover) eines Graphen $G = (V, E)$ ist eine Menge $C \subseteq V$ derart, dass für jede Kante $e = \{v_1, v_2\} \in E$ gilt: $e \cap C \neq \emptyset$, d.h., e wird durch einen Knoten aus C abgedeckt.
Das Problem, eine kleinstmögliche Knotenüberdeckung zu finden, ist eines der „Kernprobleme“ im Buch von Garey/Johnson, d.h. insbesondere, es ist NP-vollständig.

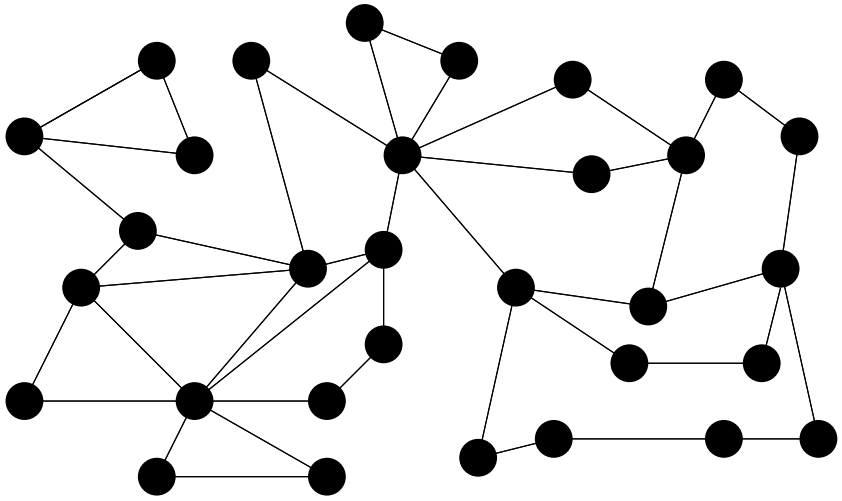
Wo findet sich das Knotenüberdeckungsproblem in der Praxis?

a) Im Mittelalter: Errichtung von Raubritterburgen an wichtigen Handelswegen
Besonderheit: planarer Graph⇒1.5-Approximation und besser!

b) In der Neuzeit: Eine internationale Firma will in Duty-Free-Shops auf internationalen Flughäfen so vertreten sein, dass jeder Passagier, der auf einer „Hauptroute“ fliegt, beim Start- oder beim Zielflughafen die Gelegenheit hat, bei einer Filiale der Firma einzukaufen.
Hier: i.a. nichtplanarer Graph!

c) „Fehlergraphen“

Beispiel: Wie groß ist ein kleinstmögliches VC?

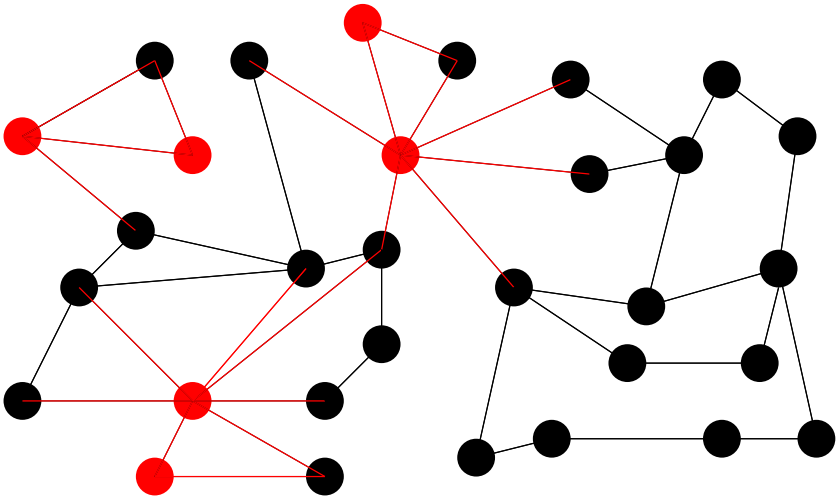


Einfache Beobachtungen und Regeln

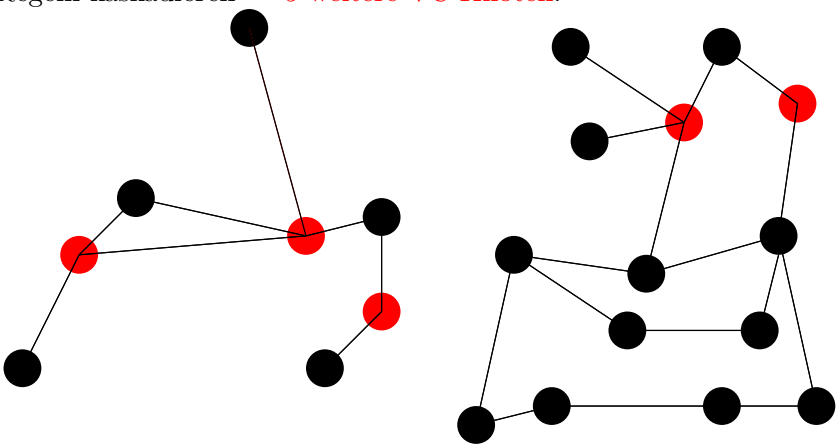
- Zwei Knoten in einem Dreieck gehören in ein VC.

- Wenn Wahlmöglichkeit besteht, nimm solche Knoten, die "noch mehr" abdecken können.
  $\rightsquigarrow$ Nimm Nachbarn eines Grad-1-Knotens ins VC.
  $\rightsquigarrow$ Gibt es in einem Dreieck einen Knoten vom Grad zwei, so nimm dessen beide Nachbarn ins VC.

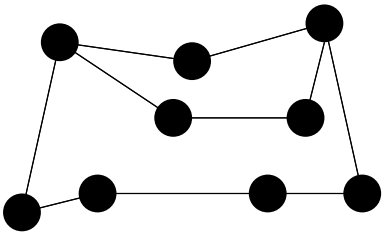
Unser Beispiel: Wir wenden die Regeln an und finden 6 VC-Knoten.



Die Regeln kaskadieren $\rightsquigarrow$ 5 weitere VC-Knoten.



Was bleibt übrig?



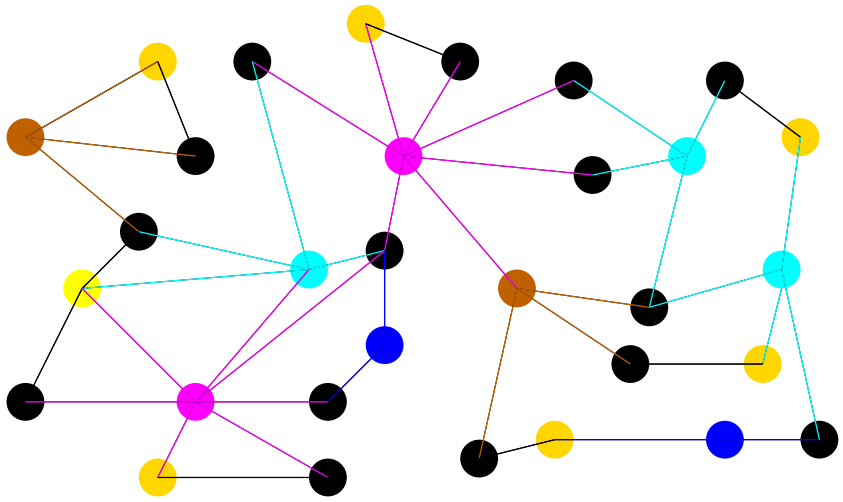
Eine einfache Greedy-Strategie

als Beispiel für ein heuristisches Verfahren ohne Gütegarantie
Aufruf: GreedyVC$(G, \emptyset)$
GreedyVC (G, C)

- Falls G leer, gib C aus. exit.
- Suche Knoten v mit maximalem Grad in G.
- Berechne GreedyVC $(G - v, C \cup \{v\})$
  Hinweis: $G - v$ entsteht aus G, indem v mit anliegenden Kanten aus G gelöscht wird.

Idee hierbei: ein Knoten hohen Grades „deckt viel ab“.
Leider gibt es keine Gütegarantie!

Wie arbeitet diese Strategie bei unserem Beispiel? Die farbig hervorgehobenen Knoten würden (beispielsweise) durch die Greedy-Vorgehensweise ausgewählt. Knoten gleichen Grades tragen dabei die gleiche Farbe.



Eine einfache Suchbaumstrategie

als Beispiel für ein parametrisiertes Verfahren
Aufruf: SuchbaumVC$(G, \emptyset, k)$
Hinweis: k ist der Parameter, der die Überdeckungsmengengröße beschränkt.

SuchbaumVC(G, C, k)

- Falls G leer, gib C aus; exit.
- Falls $K = 0$: exit.
- Nimm irgendeine Kante $e = \{v_1, v_2\}$ aus G und verzweige:
  1. Berechne SuchbaumVC$(G - v_1, C \cup \{v_1\}, k - 1)$
  2. Berechne SuchbaumVC$(G - v_2, C \cup \{v_2\}, k - 1)$

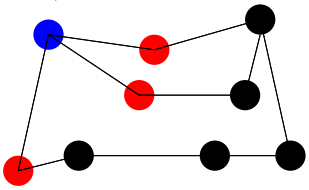
Die Laufzeit hängt nur exponentiell vom Parameter ab.
$\Rightarrow$ parameterisierter Algorithmus.

Man kann (und sollte) Suchbäume mit einfachen Regel kombinieren, welche in Polynomzeit einfache Fälle lösen können. Zwei solche Regeln hatten wir für das Knotenüberdeckungsproblem entwickelt. Für unser Beispiel bedeutet dies:

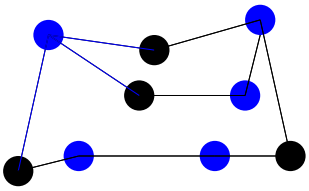
- Der ursprünglich ziemlich große Graph wird durch die *Reduktionsregeln* stark verkleinert.
- Nach einmaligem Verzweigen wird der übrigbleibende Graph vollständig durch die Reduktionsregeln erledigt.

Genauer zeigen die folgenden Bilder zunächst (farbkodiert) die beiden Verzweigungsfälle (rot und blau) und sodann, welche Knoten die Regeln noch jeweils hinzugenommen hätten, um schließlich zu einer gültigen Knotenüberdeckung zu gelangen.

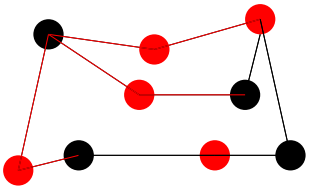
Knotenorientierter Suchbaum, kombiniert mit den Regeln $\rightsquigarrow$ zwei Zweige



Der blaue Fall



Der rote Fall



Ergebnis für unser Beispiel:

Die kleinste Knotenüberdeckung enthält 16 Knoten.
Eine kleinste Knotenüberdeckung wurde von der Heuristik gefunden.
Die Verwendung von Reduktionsregeln hilft, die Verzweigungszahl bei Suchbaumalgorithmen zu verringern.

Ein einfacher Näherungsalgorithmus

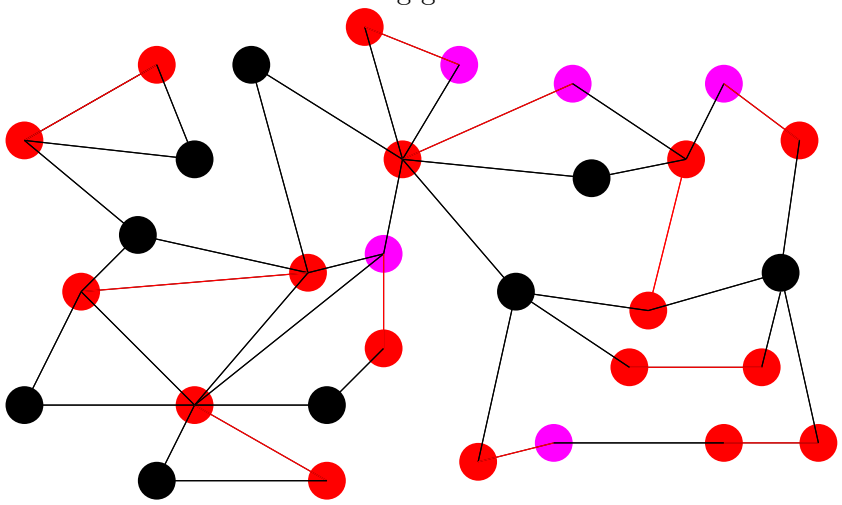
Aufruf: NeinfachVC$(G, \emptyset)$

NeinfachVC$(G = (V, E), C)$

- Falls E leer, gib C aus. exit.
- Nimm irgendeine Kante $e = \{v_1, v_2\}$ aus G
  und berechne NeinfachVC$(G - \{v_1, v_2\}, C \cup \{v_1, v_2\})$

Dies ist ein 2-Approximations-Verfahren.
Warum? Für jede von NeinfachVC ausgewählte Kante gilt: einer der beiden Eckpunkte **muss** in einer optimalen Lösung vorkommen, NeinfachVC ist aber höchstens um Faktor 2 schlechter.

In unserem Beispiel könnten wir die rot gekennzeichneten Kanten auswählen. Dies würde uns zunächst alle roten und magentafarbenen Knoten in die Überdeckung bringen. Natürlich ist es möglich (für die Güteschranke allerdings unwesentlich), daraus eine inklusionsminimale, ebenfalls zulässige Lösung (also Knotenüberdeckung) auszurechnen, indem solange wie mögliche mit einer Greedy-Strategie wiederum Knoten aus der bislang konstruierten Lösung entfernt werden. Dies könnte dazu führen, dass die mangentafarbenen Knoten nicht zu so einer Lösung gehören.



2 Überdeckungsprobleme

Nach: R.Bar-Yehuda: „One for the price of two: a unified approach for approximating covering problems“, Algorithmica 27, 131–144, 2000

Motivation

- Der einfache 2-Approximationsalgorithmus klappt so nicht im wichtigen Fall von Graphen mit Knotengewichten.
- Klappt das intuitive Greedy-Verfahren „leicht modifiziert“?

2.1 Ein allgemeines Überdeckungsproblem

g ist gegeben durch ein Tripel (X, f, w), wobei

- X eine endliche Menge ist
- $f : 2^X \to \{0, 1\}$ eine monotone Abbildung ist, d.h. $A \subseteq B \to f(A) \leq f(B)$, und es gelte $f(X) = 1$,
- $w : X \to \mathbb{R}^+$ ist die Gewichtsfunktion

Für eine Menge $Y \subseteq X$ definieren wir $w(Y) = \sum_{x \in y} w(x)$ als Gewicht von Y.

Bemerkung 2.1 *Damit ist auch w monoton. Außerdem sind Gewichtsfunktionen **linear** in dem Sinne, dass $(w_1 + w_2)(C) = w_1(C) + w_2(C)$ gilt.*

$C \subseteq X$ ist Überdeckung gdw. $f(C) = 1$. Ziel ist es, eine kleinstmögliche Überdeckung (minimum cover) $C^* \subseteq X$ zu finden, die also

$$w(C^*) = \min\{w(C) \mid C \subseteq X, f(C) = 1\}$$

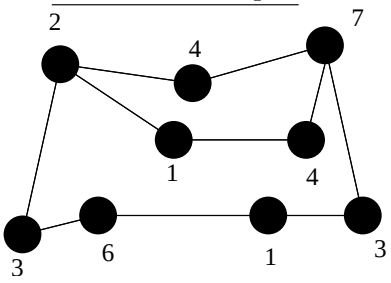
erfüllt. Eine Überdeckung heißt r-**Approximation**, falls $w(C) \leq r \cdot w(C^*)$.

2.2 Gewichtetes Knotenüberdeckungsproblem

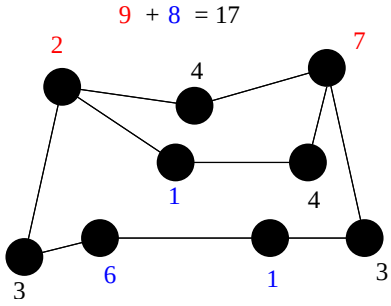
Ggb: gewichteter Graph, i.Z.: $G = (V, E), w : V \to \mathbb{R}^+$
In obiger Terminologie:

$$X = V, f : V' \to \begin{cases} 0, & V' \text{ ist keine Knotenüberdeckung} \\ 1, & V' \text{ ist eine Knotenüberdeckung} \end{cases}$$

f ist hier durch E gegeben und muss **nicht** explizit gespeichert werden bzw. gehört nicht zur Eingabe. Ein kleines Beispiel:



Ein kleines Beispiel: mit möglicher Lösung
Dabei wurden rote Knoten in die Überdeckung mit Hilfe der Große-Grad-Heuristik aufgenommen, während sich die blauen Knoten aus der folgenden Heuristik ergaben: Ist Knoten vom Grad Eins, so nimm Nachbarn! Bitte überprüfen Sie die Güte der Heuristiken in dem gegebenen Beispiel und in selbst gewählten.



2.3 Gewichtsreduktion

Zerlegungsbeobachtung: Sind w_1, w_2 Gewichtsfunktionen, so gilt für die Gewichte $OPT(w_i)$ der jeweiligen kleinstmöglichen Überdeckungen:

$$OPT(w_1) + OPT(w_2) \leq OPT(w_1 + w_2).$$

Beweis: Ist C^* optimal für $w_1 + w_2$, so gilt:

$$\begin{aligned} OPT(w_1 + w_2) &= (w_1 + w_2)(C^*) \\ &= w_1(C^*) + w_2(C^*) \\ &\geq OPT(w_1) + OPT(w_2) \end{aligned}$$

Betrachte nun *Gewichtsreduktionsfunktion* $\delta : X \to \mathbb{R}^+$
mit $\forall x \in X : 0 \leq \delta(x) \leq w(x)$, d.h. δ und $w - \delta$ sind Gewichtsfunktionen.
Wir definieren: $\Delta OPT := OPT(w) - OPT(w - \delta)$.
Zerlegungsbeobachtung $\rightsquigarrow$

$$\begin{aligned} \Delta OPT &= OPT(w) - OPT(w - \delta) \\ &\geq OPT(\delta) + OPT(w - \delta) - OPT(w - \delta) \end{aligned}$$

Eine Gewichtsreduktion führt daher zu einer Optimumsreduktion um wenigstens $OPT(\delta)$.

Wir nennen δ *r-effektiv*, falls $\delta(X) \leq r \cdot OPT(\delta)$.

Betrachte folgenden Grundalgorithmus:

$A(X, f, w)$:

- Wähle r-effektive Gewichtsreduktion δ .
- Berechne durch $B(X, f, w - \delta)$ eine Überdeckung C .
- Gib C an.

Der erwähnte Algorithmus B wird häufig A selbst wieder sein (oder eine leichte Modifikation).

Satz 2.2 *Satz über lokale Verhältnisse*
Liefert B eine Überdeckung C mit $(w - \delta)(C) \leq r \cdot OPT(w - \delta)$, [1] *dann ist $w(C) \leq r \cdot OPT(w)$, d.h. A ist r-Approximation.*

Beweis:

$$\begin{aligned} w(C) &= (w - \delta)(C) + \delta(C) \text{ [Linearität]} \\ &\leq (w - \delta)(C) + \delta(X) \text{ [Monotonie von } \delta\text{]} \\ &\leq r \cdot OPT(w - \delta) + r \cdot OPT(\delta) \text{ [B's Eigenschaft und } \delta \text{ } r\text{-effektiv]} \\ &\leq r \cdot OPT(w) \text{ [Zerlegungsbeobachtung]} \end{aligned}$$

[1] Ist also B eine r-Approximation

2.4 2-Approximationen für das gewichtete Knotenüberdeckungsproblem

Der Algorithmus von Bar-Yehuda und Even

Kante e definiert Gewichtsreduktionsfunktion δ_e durch

$$\delta_e(v) = \begin{cases} min\{w(v_1), w(v_2)\} & , \quad v \in e \\ 0 & , \quad v \notin e \end{cases}$$

Aus $\forall e \in E : \delta_e = 0$ folgt: $\forall \{v_1, v_2\} \in E : w(v_1) = 0 \vee w(v_2) = 0$; d.h., die ausgegebene Menge C ist eine Knotenüberdeckung.

1. Rekursive Variante

   BErec $G = (V, E), w)$

   - Falls $\forall e \in E : \delta_e = 0$, gib $C = \{v \in V \mid w(v) = 0\}$ aus; exit.
   - Nimm irgendeine Kante $e = \{v_1, v_2\}$ aus G (mit $\delta_e \neq 0$);
   - Berechne BErec $(G, w - \delta_e)$

2. Iterative Variante

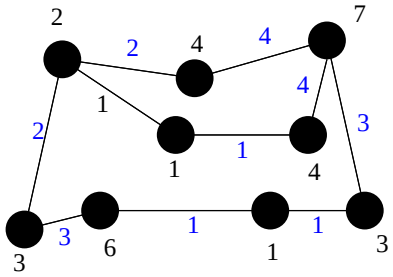
   BEiter $G = (V, E), w)$

   - Für jedes $e \in E$
     - Bestimme $\varepsilon := \min\{w(v) \mid v \in e\}$.
     - Für jedes $v \in e$ setze $w(v) := w(v) - \varepsilon$.
   - Gib $C = \{c \in V \mid w(v) = 0\}$ aus.

Es ist leicht einzusehen, dass die iterative Variante dasselbe wie die rekursive berechnet.

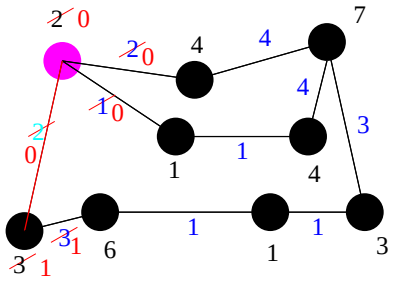
In dem hier diskutierten Fall von Graphen ist für $e = \{v_1, v_2\}$ $\varepsilon = \min\{w(v_1), w(v_2)\}$ und die Gewichtsreduktion ist dann $w(v_1) := w(v_1) - \varepsilon$ sowie $w(v_2)) := w(v_2) - \varepsilon$. Die etwas umständlich-mathematisch wirkende Formulierung wurde gewählt, um auch Hypergraphen mit demselben Algorithmus weiter unten abhandeln zu können.

Wir zeigen die Arbeitsweise der rekursiven Variante des Algorithmus an unserem kleinen Beispiel; die Gewichtsreduktionsfunktion δ_e ist an den Kanten notiert:

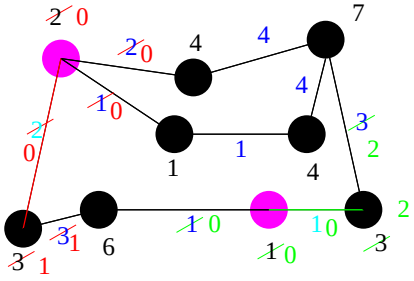


Die folgenden Bilder zeigen, wie sich nach und nach die Gewichtsfunktion verändert durch fortgesetzte rekursive Aufrufe. Die ausgewählte Kante ist durch eine neue Farbe hervorgehoben. Die Knoten, die unser Algorithmus schließlich in die Knotenüberdeckung hineinfügen wird, sind in Magenta herausgestellt.

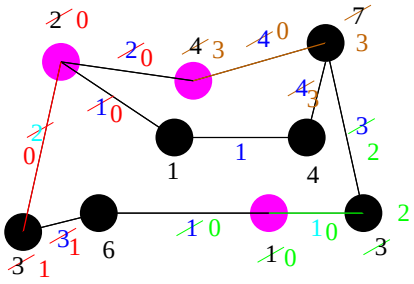
Erste Kantenwahl:



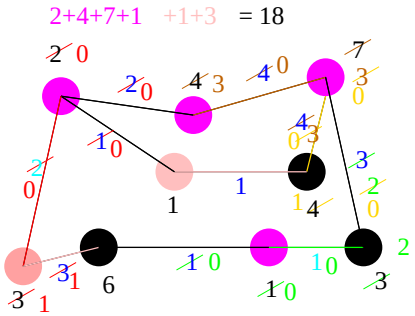
Zweite Kantenwahl:



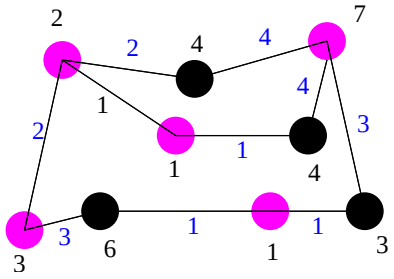
Dritte Kantenwahl:



Vierte Kantenwahl:



Die gewonne Lösung enthält eine inklusions-minimale mit Gewicht 14; diese kann mit einem Greedy-Algorithmus (als Nachverbesserung) gefunden werden.



Satz 2.3 *Der Algorithmus von Bar-Yehuda und Even ist eine 2-Approximation.*

Beweis: Wir beweisen jetzt die Behauptung durch Induktion über die Rekursionstiefe t von BErec:

- Ist $t = 0$, so liefert BErec sogar eine optimale Lösung.
- Ist die Behauptung, BErec liefere 2-Approximationen, für $t = 0, .., T$ gezeigt, so liefert der Satz über die lokalen Verhältnisse den Induktionsschritt, denn δ_e ist 2-effektiv. Ist nämlich C_{δ_e} eine optimal Überdeckung für δ_e, so wird insbesondere $e = \{v_1, v_2\}$ abgedeckt, d.h.

  $$\delta_e(C^*_{\delta_e}) \geq \min\{w(v_1), w(v_2)\};$$

  andererseits ist natürlich

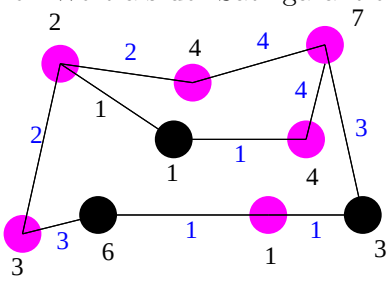
  $$\delta_e(V) = \delta_e(v_1) + \delta_e(v_2) = 2 \cdot \min\{w(v_1), w(v_2)\},$$

  also $\delta_e(V) \leq 2 \cdot OPT(\delta_e)$.

Wie gut ist die gefundene Lösung "wirklich" ?

Diese Frage enthält ein wichtiges Problem beim "*Benchmarking*" (Leistungsvergleich) heuristischer Algorithmen für NP-harte Probleme. Hierbei sind auch Approximationen hilfreich, nämlich durch "möglichst ungeschickte Wahl" im Algorithmusablauf.

In unserem Beispiel könnte man am Schluss auch eine Lösung vom Gewicht 21 erhalten haben (s.u.); eine kleinstmögliche Überdeckung C^* hat also Gewicht $w(C^*) \geq 11$. Der von uns zunächst verfolgte Ablauf des Algorithmus liefert also einen besseren Wert als der Satz garantiert.



Versuchen Sie einmal nachzuvollziehen, in welcher Reihenfolge der Algorithmus von Bar-Yehuda/Evens hier gearbeitet hat.

Der Algorithmus von Clarkson

Wir schicken einige Hilfsdefinitionen voraus:

- δ_e aus dem vorigen Algorithmus BErec
- $N(v)$: *Menge der Nachbarn* von Knoten v (*offene Nachbarschaft*).
- $d(v)$ sei der *Grad* (engl.: degree) des Knoten v, also: $d(v) = |N(v)|$. $d'(v) = |N'(v)|$ mit $N'(v) = \{u \in N(v) \mid w(u) > 0\}$ für Graph mit Knotengewichten w
- $\varepsilon(v) = \frac{w(v)}{d(v)}, \quad \varepsilon'(v) = \frac{w(v)}{d'(v)}$
- Gewichtsreduktionsfunktionen:

$$\delta_v^{(\prime)}(u) = \begin{cases} w(v) & , \quad u = v \\ \varepsilon^{(\prime)}(v) & , \quad u \in N^{(\prime)}(v) \\ 0 & , \quad \text{sonst} \end{cases}$$

1. Rekursive Variante

   Crec $(G = (V, E), w)$

- Falls $\forall e \in E,\ \delta_e = 0,$ gib $C = \{c \in V \mid w(v) = 0\}$ aus; exit.
- Suche Knoten $v \in V$, der $\varepsilon'(v)$ minimiert[2].
- Dieses v definiert Gewichtsreduktionsfunktion $\delta_v(u)$.
- Berechne $\mathrm{Crec}(G, w - \delta'_v)$.

2. Iterative Variante

   Citer $(G, (V, E), w)$

   - $C := \emptyset$
   - Solange $E \neq \emptyset$, tue
     - Suche $v \in V$, der $\varepsilon(v)$ minimiert.
     - Für jeden Nachbarn $u \in V$ von v setze $w(u) := w(u) - \varepsilon(v)$.
     - Setze $G := G - v$.
     - Setze $C := C \cup \{v\}$.
   - Gib C aus.

Man überlege sich, dass die iterative Variante das selbe wie die rekursive liefert. Beachte hierzu: Bei Crec bleiben nullgewichtete Knoten unbeachtet (ε'), bei Citer werden sie bevorzugt gelöscht.

Dass die rekursive eine 2-Approximation ist, folgt wiederum durch Induktion aus dem Satz über lokale Verhältnisse. Wesentlich ist noch zu zeigen, dass δ_v eine 2-effektive Gewichtsfunktion ist.

a) Gewichtsfunktion: Klar für $u \notin N^{(\prime)}[v] = N^{(\prime)}(v) \cup \{v\}$ (*abgeschlossene Nachbarschaft*).
   Ist $u \in N^{(\prime)}(v)$, so ist $\delta_v^{(\prime)}(u) = \frac{w(v)}{d^{(\prime)}(v)} \leq \frac{w(u)}{d^{(\prime)}(u)} \leq w(u)$ aufgrund der minimalen Wahl von v.

b) 2-effektiv:

$$\begin{aligned} \delta_v^{(\prime)}(V) &= \delta_v^{(\prime)}(v) + \delta_v^{(\prime)}(N^{(\prime)}(v)) + \delta_v(V - N^{(\prime)}[v]) \\ &= w(v) + \left|N^{(\prime)}(v)\right| \frac{w(v)}{d^{(\prime)}(v)} + 0 \\ &= 2 \cdot w(v) = 2 \cdot OPT(\delta_v^{(\prime)}) \end{aligned}$$

Überlegen Sie sich, wie der Algorithmus von Clarkson auf unserem Beispiel arbeitet.

[2] $d(v)$ sei der Grad des Knoten v

2.5 Randomisierte r-Approximation

Eine Wahrscheinlichkeitsverteilung auf dem Raum der Gewichtsreduktionsfunktionen zu (X, f, w) heißt r-**effektiv**, falls

$$E(\delta(X)) \leq r \cdot E(OPT(\delta))^3$$

Unter der Ausnutzung der Linearität des Erwartungswertfunktionals lässt sich in völliger Analogie zum deterministischen Fall zeigen:

Satz 2.4 *Satz über lokale Verhältnisse - Randomisierte Version*

Betrachte dazu folgenden Grundalgorithmus (bei Verteilung F)
$A(X, f, w)$

- *Wähle $\delta : X \to \mathbb{R}^+$ gemäß r-effektiver Verteilung F.*
- *Berechne durch $B(X, f, w - \delta)$ eine Überdeckung C.*
- *Gib C aus.*

Falls B eine Überdeckung C liefert mit

$$E((w - \delta)(C)) \leq r \cdot E(OPT(w - \delta)),$$

dann gilt:

$$E(w(C)) \leq r \cdot OPT(w),$$

d.h. A liefert eine randomisierte r-Approximation.

Beispiel 2.5 *RandomGreedyVC* $(G = (V, E), w)$

- *Setze $C := \{v \in V \mid w(v) = 0\}$ und $G := G - C$.*
- *Solange $E \neq \emptyset$:*
  - $\overline{w} := \left(\sum_{v \in V} \frac{d(v)}{w(v)}\right)^{-1}$
  - *Wähle zufällig $v \in V$ mit Wahrscheinlichkeit $p(v) = \frac{d(d)}{w(v)} \cdot \overline{w}$.*
  - *Setze $C := C \cup \{v\}$ und $G := G - v$.*
- *Gib C aus.*

[3] E ist Erwartungswert (bzgl. der angenommenen Verteilung)

Ergebnis unserer Analyse soll sein nachzuweisen, dass RandomGreedyVC eine randomisierte 2-Approximation für v ist. Dazu wären die im folgenden skizzierten Schritte im einzelnen durchzuführen:

1. Überführe RandomGreedyVC in eine äquivalente rekursive Form.
2. Zeige die Behauptung durch vollständige Induktion über die Rekursionstiefe unter Benutzung der randomisierten Version des Satzes über lokale Verhältnisse.
3. Dazu ist zu klären: Wie sieht die Wahrscheinlichkeitsverteilung auf dem Raum der Gewichtsreduktionsfunktion aus?
   Die Reduktionsfunktion[4]
   $$\delta^v(u) = w(u)\delta_{uv} = \begin{cases} w(v), & u = v \\ 0, & u \neq v \end{cases}$$
   wird mit Wahrscheinlichkeit $p(v)$ gezogen. $\sum_{v \in V} p(v) = 1$ gilt nach Definition von $p(v)$.[5]
4. Zu zeigen bleibt: die Wahrscheinlichkeitsverteilung ist 2-effektiv.

   a)
   $$E(OPT(\delta)) = E(\delta(C^*)) = \sum_{v \in V} \delta^v(C^*) \cdot p(v)$$
   $$= \sum_{v \in C^*} w(v)p(v) = \sum_{v \in C^*} d(v) \cdot \overline{w} \geq |E| \cdot \overline{w}$$

   b)
   $$E(\delta(V)) = \sum_{v \in V} w(v)p(v) = \sum_{v \in V} d(v)\overline{w} \leq 2 \cdot |E| \cdot \overline{w}$$

2.6 (Δ-)Hitting-Set — eine weitere Anwendung der Sätze über lokale Verhältinisse

Ggb: Grundmenge U („Universum“) von n Elementen $e_1, \ldots, e_n$ und Mengen $S_1, \ldots, S_m \subseteq U$

Ges: Kleinste Anzahl j von Elementen $e_{i_1}, \ldots, e_{i_j}$, die alle Hyperkanten treffen, d.h. $\forall\, 1 \leq k \leq m\ \exists\, 1 \leq r \leq j : e_{i_r} \in S_k$

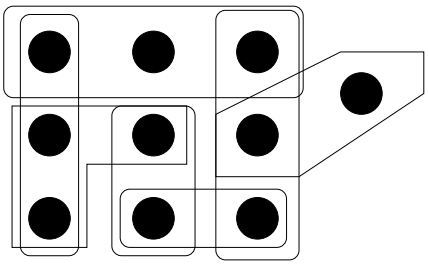
Ist bekannt, dass $\forall 1 \leq i \leq m : |S_i| \leq \Delta$ für eine Konstante Δ, so lassen sich die Algorithmen von Bar-Yahuda, Clarkson und Random-Greedy

[4] Wie üblich bezeichnet δ_{uv} die Kroneckerfunktion.
[5] Alle übrigen Gewichtsfunktionen werden mit Wahrscheinlichkeit 0 gezogen.

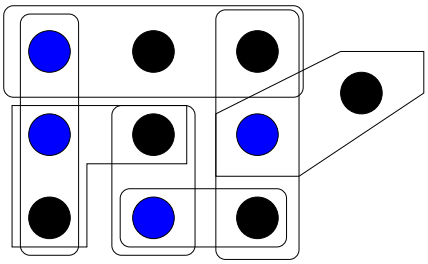
auch für Δ-Hitting-Set lesen.
Die genannten Algorithmen funktionieren natürlich auch für die entsprechenden gewichteten Varianten und liefern jeweils Δ-Approximationen.

Eine mögliche Anwendung sind automatische Diagnosesysteme. Dies erläutern wir näher auf den Folien.

Ein abstrakteres Beispiel:



Eine kleinste Überdeckung in dem Beispiel wäre:



Neben Approximationsalgorithmen mit Gütegarantien sind auch hier wieder Heuristiken hilfreich, insbesondere solche, die in manchen Situationen beweisbar eine bestmögliche Wahl treffen (Datenreduktionsregeln).

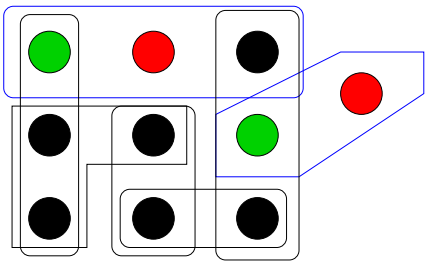
Datenreduktionsregeln

1. Kantendominierung: $f \subset e$. $\rightsquigarrow$ entferne e
2. Kleine Kanten: $e = \{v\} \rightsquigarrow v$ kommt ins HS; entferne e
3. Knotendominierung: Ein Knoten x heiße *dominiert* durch einen Knoten y, falls $\{e \in E \mid x \in e\} \subseteq \{e \in E \mid y \in e\} \rightsquigarrow$ entferne x

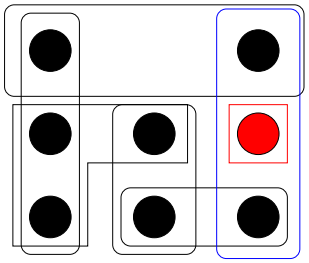
R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. John Wiley & Sons, 1972.

Oft wiederentdeckt: K. Weihe (Zugnetzoptimierung), R. Niedermeier & P. Rossmanith (param. HS, 2003)
Auch: R. Reiter (Theory of Diagnosis $\rightsquigarrow$ HS Bäume, 1987)

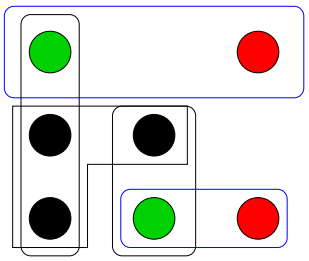
Knotendomination



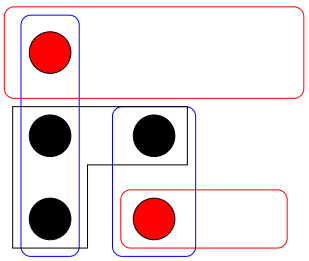
Kantenregeln



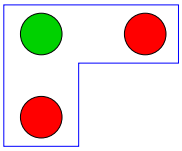
Knotendomination



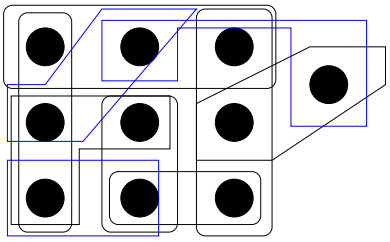
Kantenregeln



Knotendomination



Ein irreduzibles Beispiel

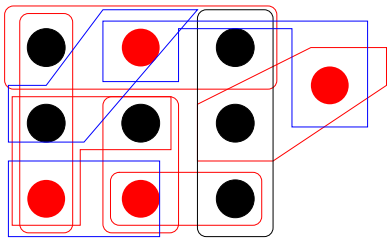


Ein Näherungsverfahren — übersetzt von VC: $N1 - 3HS(G = (V, E), C)$

- (Wende Reduktionsregeln an.)
- Falls E leer, gib C aus; exit.
- Nimm irgendeine "kleine Kante" e aus G
  und berechne $N1 - 3HS(G - e, C \cup e)$ (e aufgefasst als Knotenmenge)

Dies ist ein **3**-Approximations-Verfahren.

Ein irreduzibles Beispiel wird approximiert:



Wie gut ist die Approximation "wirklich" ?

Faktor 3 ist nur schlimmster Fall! Wir erreichen 5 Knoten-Lösung mit dem "banalen" Algorithmus.
Da unser irreduzibles Beispiel das frühere als Teilfall enthält, wissen wir: 4 ist eine untere Schranke. Am Lauf des Algorithmus sehen wir: zweimal haben wir zwei Knoten statt möglicherweise einem genommen, beim dritten Schritt waren wir optimal
$\rightsquigarrow$ untere Schranke 3 (auch ohne das früher behandelte Teilproblem).
$\rightsquigarrow$ Der Satz über lokale Verhältnisse gestattet das Auffinden besserer Schranken im konkreten Beispiel.
Tatsächlich gibt es Lösung mit 4 Knoten, und die findet unser Algorithmus "fast".

1.3 Zusammenfassung und Ausblick

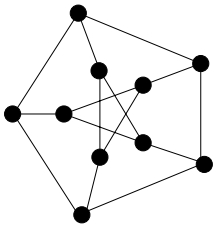
- Bisher kennen gelernte Näherungsalgorithmen sind oft sehr kurze Programmstücke
- Schwierigkeit: Analyse, betreffend die **Güte** der Näherung!
- Warum sind die Gütegarantien schwierig zu beweisen?
  Grundsätzlich (und intuitiv) lässt sich dies genau so begründen wie die vermutete Ungleichheit von P und NP.

Das sehen wir am Beispiel des Knotenüberdeckungsproblems:

+ Es ist leicht, eine vorgegebene Lösung insofern zu verifizieren, als dass die Knotenüberdeckungseigenschaft überprüft wird.

- Es ist schwierig nachzuweisen, dass eine gefundene Knotenüberdeckung kleinstmöglich (optimal) ist.

Konkret: Betrachten Sie den Petersen-Graph.



Können Sie (mit Hilfe unserer bisherigen Betrachtungen) begründen, weshalb eine kleinstmögliche Knotenüberdeckung wenigstens 5 Knoten enthält?
Schwieriger noch: Benutzen Sie den Satz über lokale Verhältnisse, um nachzuweisen, dass wenigstens 6 Knoten in jeder Knotenüberdeckung des Petersen-Graphen zu finden sind. (Hinweis: $\frac{5}{3}$-Approximation für Kreise der Länge 5!)

In den folgenden Kapiteln erwartet Sie ein systematisches Kennenlernen verschiedener Techniken zum Entwurf und zur Analyse von Näherungsalgorithmen.

3 Grundlegende Definitionen

3.1 Spezifikation von Optimierungsproblemen

Wir wollen zunächst einen einfachen Formalismus zur Spezifizierung von Optimierungsproblemen kennen lernen.

Definition 3.1 *Ein* Optimierungsproblem $\mathcal{P}$ *wird beschrieben durch ein Quadrupel* $(I_\mathcal{P}, S_\mathcal{P}, m_\mathcal{P}, opt_\mathcal{P})$*, wobei*

1. $I_\mathcal{P}$ *die Menge der möglichen Eingaben (Instanzen),*
2. $S_\mathcal{P} : I_\mathcal{P} \to$ *Menge der zulässigen Lösungen (engl.: feasible solutions),*
3. $m_\mathcal{P} : (x, y) \mapsto m_\mathcal{P}(x, y) \in \mathbb{N}$ *(oder* $\mathbb{Q}, ...$*) für* $x \in I_\mathcal{P}, y \in S_\mathcal{P}(x)$ *liefert den Wert der zulässigen Lösung* y *und*
4. $opt_\mathcal{P} \in \{\min, \max\}$ *gibt an, ob* $\mathcal{P}$ *ein Minimierungs- oder Maximierungsproblem ist.*

$I_\mathcal{P}$ *und* $S_\mathcal{P}(x)$ *sind –geeignet codierte– formale Sprachen über dem Alphabet* $\{0, 1\}$.

Weitere Bezeichnungen

$S^*_\mathcal{P} : I_\mathcal{P} \to$ Menge der optimalen Lösungen, d.h.

$$\forall x \in I_\mathcal{P}\ \forall y^*(x) \in S^*_\mathcal{P}(x) : m_\mathcal{P}(x, y^*(x)) = \mathrm{opt}_\mathcal{P}\{m_\mathcal{P}(x, z) \mid z \in S_\mathcal{P}(x)\}.$$

Der Wert einer optimalen Lösung wird auch $m^*_\mathcal{P}(x)$ notiert.

Ist $\mathcal{P}$ aus dem Zusammenhang klar, so schreiben wir kurz –unter Fortlassung des Indexes $\mathcal{P}$— $I, S, m, \mathrm{opt}, S^*, m^*$.

3.2 Problemvarianten

Konstruktionsproblem (construction problem):

$$\mathcal{P}_C : \text{Ggb. } x \in I_\mathcal{P}, \text{liefere ein } y^*(x) \in S^*_\mathcal{P}(x) \text{ sowie ihren Wert } m^*_\mathcal{P}(x)$$

Auswertungsproblem (evaluation problem):

$$\mathcal{P}_E : \text{Ggb. } x \in I_\mathcal{P},\ \text{liefere } m^*_\mathcal{P}(x)$$

Entscheidungsproblem (decision problem):

$$\mathcal{P}_D : \text{Ggb. } x \in I_\mathcal{P} \text{ und Parameter } k \in \mathbb{N},\ \text{entscheide ob}$$
$$m^*_\mathcal{P}(x) \geq k, \quad (\text{falls } \mathrm{opt}_\mathcal{P} = \max)$$
$$\text{bzw. ob}$$
$$m^*_\mathcal{P}(x) \leq k, \quad (\text{falls } \mathrm{opt}_\mathcal{P} = \min).$$

Beispiel 3.2 *Knotenüberdeckungsproblem VC*

1. $I = \{G = (V, E) \mid G \text{ ist Graph } \}$
2. $S(G) = \{U \subseteq V \mid \forall \{x, y\} \in E : x \in U \vee y \in U\}$
   (Knotenüberdeckungseigenschaft)
3. $m = |U|$
4. $opt = \min$

VC_D ist dann das entsprechende „parametrisierte Problem", d.h. ggb. Graph $G = (V, E)$ und Parameter k, gibt es eine Knoteüberdeckung $U \subseteq V$ mit $|U| \leq k$?

3.3 Die Klassen PO und NPO

Definition 3.3 *$\mathcal{P} = (I, S, m, opt)$ gehört zu NPO, gdw:*

1. *$x \overset{?}{\in} I$ ist in Polynomzeit entscheidbar.*
2. *Es gibt ein Polynom q derart, dass $\forall x \in I \; \forall y \in S(x) : |y| \leq q(|x|)$ und für alle y mit $|y| \leq q(|x|)$ ist die Frage $y \overset{?}{\in} S(x)$ in Polynomzeit entscheidbar.*
3. *m ist in Polynomzeit berechenbar.*

Beispiel VC: zu Punkt 2: Jede Knotenüberdeckung ist in ihrer Größe trivialerweise durch die Mächtigkeit der Knotenmenge beschränkt.

Satz 3.4 *Ist $\mathcal{P} \in NPO$, so ist $\mathcal{P}_D \in NP$.*

Beweis: (nur für $\text{opt}_{\mathcal{P}} = \max$), $\mathcal{P}_D$ lässt sich bei Eingabe von $x \in I$ und k wie folgt lösen (in nichtdeterministischer Weise):

1. Rate y mit $|y| \leq q(|x|)$ in Zeit $O(q(|x|))$. ($q(|x|)$ Bits sind zu raten)
   Dieser Schritt ist polynomiell wegen Eigenschaft 2 von NPO-Problemen.
2. Teste $y \in S(x)$ in Polynomzeit.
   Dieser Schritt ist polynomiell wegen Eigenschaft 1 von NPO-Problemen.
3. Falls $y \in S(x)$, berechne $m(x, y)$ in Polynomzeit.
   Dieser Schritt ist polynomiell wegen Eigenschaft 3 von NPO-Problemen.
4. Falls $y \in S(x)$ und $m(x, y) \leq k$, antworte JA.
5. Falls $y \notin S(x)$ oder $m(x, y) > k$, antworte NEIN. □

Definition 3.5 *Ein Optimierungsproblem $\mathcal{P}$ gehört zur Klasse PO gdw. $\mathcal{P}_C$ in Polynomzeit gelöst werden kann.*

Der nachstehende Satz 3.7 liefert eine Begründung dafür, dass PO über das zugehörige Konstruktionsproblem definiert ist: Das Konstruktionsproblem ist nämlich das Schwierigste unter den diskutierten Problemvarianten. D.h., wenn das Konstruktionsproblem „einfach" ist, so sind erst recht die anderen Varianten einfach.

Erinnerung: (polynomielle) Turing-Reduzierbarkeit
$\mathcal{P}_1 \leq_T^{(p)} \mathcal{P}_2$ (in Worten: $\mathcal{P}_1$ ist reduzierbar auf $\mathcal{P}_2$) gdw.
es gibt eine Turing-Maschine, die eine Instanz x von $\mathcal{P}_1$ (in Polynomzeit) mit Hilfe von Orakelanfragen (das sind generierte Instanzen von $\mathcal{P}_2$, die auf ein „Orakelband" geschrieben werden und gedachter Weise in konstanter Zeit beantwortet werden) bearbeiten kann.
Hinweis: In einer einführenden Vorlesung zur Komplexitätstheorie wird vornehmlich auf den Spezialfall der many-one-Reduzierbarkeit eingegangen, bei der insbesondere nur ein Orakelaufruf möglich ist.

Wir werden uns in dieser Vorlesung vornehmlich um kombinatorisch harte Probleme kümmern, für die kein (deterministischer) Polynomzeitalgorithmus bekannt ist. Daher ist die polynomielle Turing-Reduzierbarkeit $\leq_T^p$ in diesem Abschnitt der für uns adäquate Begriff. Wir schreiben $\mathcal{P}_1 \equiv_T^p \mathcal{P}_2$, falls sowohl $\mathcal{P}_1 \leq_T^p \mathcal{P}_2$ als auch $\mathcal{P}_2 \leq_T^p \mathcal{P}_1$ gelten.

Da $VC \in$ NPO und VC_D NP-hart ist, erhalten wir:

Folgerung 3.6 *$P \neq NP \rightarrow PO \neq NPO$.*

Satz 3.7 *$\forall \mathcal{P} \in NPO : \mathcal{P}_D \equiv_T^p \mathcal{P}_E \leq_T^p \mathcal{P}_C$*

Beweis: Klar: $\mathcal{P}_D \leq_T^p \mathcal{P}_E \leq_T^p \mathcal{P}_C$.
Z.z: $\mathcal{P}_E \leq_T^p \mathcal{P}_D$.
Dazu überlegen wir uns:

$$\{m_{\mathcal{P}}(x, y) \mid y \in S_{\mathcal{P}}(x)\} \subseteq 0 \ldots 2^{p(|x|)} \tag{1}$$

für ein Polynom p.
Dann kann $\mathcal{P}_E$ durch binäre Suche auf dem Intervall $0 \ldots 2^{p(|x|)}$ mit $p(|x|)$ vielen Orakelanfragen an $\mathcal{P}_D$ gelöst werden.

Da $\mathcal{P} \in$ NPO, ist $m_{\mathcal{P}}(x, y)$ in Zeit $r(|x|, |y|)$ für ein Polynom r berechenbar. Das heißt insbesondere, dass

$$0 \leq m_{\mathcal{P}}(x, y) \leq 2^{r(|x|,|y|)}$$

gilt. Da $\mathcal{P} \in$ NPO, ist $|y| \leq q(|x|)$ für alle $y \in S_{\mathcal{P}}(x)$ für ein Polynom q. Daher gilt für das Polynom $p(n) := r(n, q(n))$ die Beziehung (1).

Im Folgenden wollen wir klären, inwiefern bzw. wann auch $\mathcal{P}_E$ und $\mathcal{P}_C$ polynomiell Turing-äquivalent sind. Dazu diskutieren wir zunächst ein Beispiel.

3.4 MAXCLIQUE als Beispielproblem

Ein *vollständiger Graph* mit n Knoten ist (isomorph zu, i.Z. $\cong$)

$$K_n = (\{1 \ldots n\}, \{\{i, j\} \mid 1 \leq i < j \leq n\}).$$

Eine *Clique* in einem Graphen ist eine Knotenteilmenge, die einen vollständigen Graphen induziert. Das MAXCLIQUE-Problem fragt in einem Graphen nach größtmöglichen (maximum) Cliquen.

Unter Benutzung des oben eingeführten Formalismus zur Spezifizierung von Optimierungsproblemen stellt sich MAXCLIQUE wie folgt dar:

I : alle Graphen $G = (V, E)$
S : alle Cliquen in G, d.h. $U \subseteq V : G(U) \cong K_{|U|}$ (vollständiger Graph mit $|U|$ Knoten)
$m = |U|$
opt = max

Algorithmus für $MAXCLIQUE_C \leq_T^p MAXCLIQUE_E$

Wir stellen einen einfachen Algorithmus, $MAXCLIQUE_C^E(G)$, vor, der $MAXCLIQUE_C \leq_T^p MAXCLIQUE_E$ zeigt:

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

Ausgabe: eine größtmögliche Clique in G

begin

$k := MAXCLIQUE_E(G)$;

Falls $k = 1$ **liefere** irgendeinen Knoten in V

sonst finde Knoten $v \in V$ mit $k = MAXCLIQUE_E(G(N[v]))$

und **liefere** $\{v\} \cup MAXCLIQUE_C^E(G(N(v)))$.[6]

Warum ist das Programm korrekt?

- Die Auswahl „$v \in V$ mit $k = MC_E(G(N[v]))$“ garantiert, dass v in einer größtmöglichen Clique liegt.
- Es ist klar, dass $G(N(v))$ eine Clique (evtl. mehrere) der Größe $k-1$ enthält (und auch keine größere); eine dieser wird rekursiv gefunden.

Zeitkomplexität des Algorithmus ($n = \#$ Knoten):

$$\begin{aligned} T(1) &= O(1) \\ T(n) &= (n+1) + T(n-1) \\ &\quad \hookrightarrow \text{Durchsuchen des Graphen} \\ &= (n+1) + n + \ldots + n + O(1) = O(n^2) \end{aligned}$$

Gilt „das“ auch allgemein? Der Schlüssel dazu ist die NP-Härte des Entscheidungsproblems MAXCLIQUE$_D$, wie der folgende Satz lehrt.

Satz 3.8 *Ist $\mathcal{P} \in$ NPO und ist $\mathcal{P}_D$ NP-hart, so gilt: $\mathcal{P}_C \leq_T \mathcal{P}_D$.*

Beweis: (für $\text{opt}_{\mathcal{P}} = \max$).
Im Beweisgang werden wir ein anderes NPO-Problem $\mathcal{P}'$ konstruieren mit $\mathcal{P}_C \leq_T^p \mathcal{P}'_E$. Wegen Satz 3.4 liegt $\mathcal{P}'_D$ in NP, und $\mathcal{P}_D$ ist nach Voraussetzung NP-hart, also gilt

$$\mathcal{P}'_E \leq_T^p \mathcal{P}'_D \leq_T^p \mathcal{P}_D$$

wegen Satz 3.7, und die Transitivität der Reduktionsrelation liefert die Behauptung.
$\mathcal{P}'$ ist wie $\mathcal{P}$ definiert mit Ausnahme des Messfunktion $m_{\mathcal{P}'}$. Betrachte dazu ein Polynom q mit $\forall x \in I_{\mathcal{P}}\ \forall y \in S_{\mathcal{P}}(x), |y| \leq q(|x|)$ (s. Definition NPO). Mit anderen Worten: $S_{\mathcal{P}}(x) \subseteq \{0,1\}^{\leq q(|x|)}$.

Setze nun $\tau(y) := 2^{q(|x|)-|y|}y$ (Konkatenation!). Es gilt $|\tau(y)| = q(|x|)$. Für jedes $y \in S_{\mathcal{P}}(x)$ bezeichne $\lambda(y)$ den Wert von $\tau(y)$, interpretiert als Ternärzahl. Daraus folgt: $0 \leq \lambda(y) \leq 3^{q(|x|)}$.
Außerdem gilt:
$\forall y, y' \in S_{\mathcal{P}}(x) : y = y' \iff \tau(y) = \tau(y') \iff \lambda(y) = \lambda(y')$.
Setze für jedes $x \in I_{\mathcal{P}'} = I_{\mathcal{P}}$ und jedes $y \in S_{\mathcal{P}'}(x) = S_{\mathcal{P}}(x)$:

$$m_{\mathcal{P}'}(x,y) = 3^{q(|x|)+1} m_{\mathcal{P}}(x,y) + \lambda(y).$$

Beachte: $\mathcal{P}' \in$ NPO, denn $m_{\mathcal{P}'}$ ist in Polynomzeit berechenbar, weil insbesondere die Exponentiation $3^{q(|x|)}$ nur $O(q(|x|))$ viele Bits benötigt.
Damit: $\forall x\ \forall y_1, y_2 \in S_{\mathcal{P}'}(x) : y_1 \neq y_2 \rightarrow m_{\mathcal{P}'}(x,y_1) \neq m_{\mathcal{P}'}(x,y_2)$.
Also gibt es eine eindeutig bestimmte maximale zulässige Lösung $y^*_{\mathcal{P}'}(x)$ in $S^*_{\mathcal{P}'}(x)$.
Nach Definition von $m_{\mathcal{P}'}$ gilt ferner:

$$(m_{\mathcal{P}'}(x,y_1) > m_{\mathcal{P}'}(x,y_2) \Rightarrow m_{\mathcal{P}}(x,y_1) \geq m_{\mathcal{P}}(x,y_2)).$$

Daraus folgt: $y^*_{\mathcal{P}'}(x) \in S^*_{\mathcal{P}}(x)$.
$y^*_{\mathcal{P}'}(x)$ kann wie folgt mit einem Orakel für $\mathcal{P}'_E$ in Polynozeit berechnet werden.

1. Bestimme $m^*_{\mathcal{P}'}(x)$ (Orakel!)
2. Berechne $\lambda(y^*_{\mathcal{P}'}(x)) = m^*_{\mathcal{P}'}(x) \mod (3^{q(|x|)+1} \cdot m_{\mathcal{P}}(x,y))$.
   Dies kann durch wiederholte Subtraktion von $3^{q(|x|)+1}$ geschehen; höchstens polynomiell oft ist zu subtrahieren, da $\mathcal{P}$ in NPO liegt und deshalb —wie wir uns schon früher überlegten— $m_{\mathcal{P}}(x,y)$ durch ein Polynom in $|x|$ beschränkt ist.
3. Bestimme y aus $\lambda(y)$.

Daher kann $\mathcal{P}_C$ mit einem Orakel für $\mathcal{P}'_E$ in Polynomzeit berechnet werden, denn $m^*_{\mathcal{P}}(x) = m_{\mathcal{P}}(x,y)$ für das mit oben stehendem Algorithmus berechnete y.

Grundtechniken

In diesem Teil des Skriptums soll es darum gehen, gewisse Grundtechniken des Entwurfs von Näherungsalgorithmen systematisch zu studieren.

Man beachte wiederum:

- Die dikutierten Verfahren sind sehr einfach (zu implementieren).
- Es ist leicht einzusehen, dass sie stets eine zulässige Lösung liefern.
- **Schwierig** bleibt der Nachweis der Güte der Näherung.

Die einzelnen Abschnitte widmen sich dabei:

- Greedy-Verfahren
- Partitionsprobleme
- Lokale Suche
- Lineares Programmieren
- Dynamisches Programmieren

4 Greedy-Verfahren

Allgemeines zu Greedy-Verfahren; hier speziell bei Maximierungsverfahren:
Allgemeine Aufgabe ist hierbei: Aus einer Grundmenge X ist eine maximale zulässige Lösung S_{max} zu finden.
Greedy lässt sich (eventuell) einsetzen, falls die Menge der zulässigen Lösungen monoton ist in dem Sinne, dass, falls S zulässige Lösung ist, so auch $S' \subseteq S$ für alle $S' \subseteq S$.
Damit ist auch $\emptyset$ eine zulässige Lösung, mit der das Verfahren starten kann. Greedy-Verfahren sortieren häufig zuerst die vorgebene Grundmenge nach einem geeigneten Kriterium und gehen die so sortierte Liste sukzessive durch.
Für jedes Element x der Liste wird dabei für die bislang gefundene zulässige Lösung S' geprüft, ob $S' \cup \{x\}$ ebenfalls zulässig ist. Wenn ja, wird $S' := S' \cup \{x\}$ gebildet. Auf diese Weise ist garantiert, dass stets nur zulässige Lösungen ausgegeben werden. Außerdem ist klar, dass eine so gefundene zulässige Lösung S' nicht mehr durch Hinzunahme eines weiteren Elements $x \notin S'$ erweitert werden kann, denn wenn $\{x\} \cup S'$ zulässig wäre, dann auch $\{x\} \cup S''$ für alle $S'' \subseteq S'$ aufgrund der Monotonie; speziell gälte dies für ein S'', das als „bisherige Lösung“ an dem Punkt, als x untersucht wurde, gebildet worden war.

Analog kann man sich Geedy-Verfahren überlegen für die in Kapitel 1 besprochenen Überdeckungsprobleme, wobei man jetzt mit der Gesamtmenge X startet und sukzessive Elemente entfernt, sodass immer noch die Zulässigkeit (modelliert durch $f(X') = 1$ für $X' \subseteq X$) gewährleistet ist. Beispiel 4.15 (s.u.) zeigt eine alternative Greedy-Strategie für Minimierungsprobleme.

Wegen des Sortierschritts haben solche Greedy-Verfahren meist eine Zeitkomplexität von $O(n \log n)$.

4.1 Das Rucksackproblem

Beispiel 4.1 *Maximum Knapsack (Rucksackproblem)*

I:	*endliche Menge* X	*bzw. „Liste“*	$X = \{x_1, \ldots, x_n\}$
	Profitfunktion $p : X \to \mathbb{N}$	*bzw. „Liste“*	$\{p_1, \ldots, p_n\}$
	Größenfunktion $a : X \to \mathbb{N}$	*bzw. „Liste“*	$\{a_1, \ldots a_n\}$
	Fassungsvermögen $b \in \mathbb{N}$ *des Rucksacks*		
	[O.E.: $\forall 1 \leq i \leq n : a_i \leq b$ *im Folgenden]*		
S:	$\{Y \subseteq X \mid a(Y) = \sum_{x_i \in Y} a_i \leq b\}$		
m:	$p(Y) = \sum_{x_i \in Y} p_i$		
opt:	*max*		

Mitteilung 4.2 *Knapsack ist NP-vollständig.*

Heuristische Idee:
Es ist günstig, möglichst solche Sachen x_i in den Rucksack zu tun, die möglichst viel Profit versprechen, gemessen am in Anspruch genommenen Platz.

$\rightsquigarrow$ GreedyKnapsack (X, p, a, b)

1. Sortiere X absteigend nach dem Verhältnis $\frac{p_i}{a_i}$.
   ($\{x_1, \ldots, x_n\}$ sei die erhaltene sortierte Grundmenge)
2. $Y := \emptyset$ (die leere Menge ist zulässig)
3. Für $i := 1$ bis n tue:
   wenn $a_i \leq b$, dann $Y := Y \cup \{x_i\}$; $b := b - a_i$.
4. Liefere Y zurück.

Leider kann GreedyKnapsack beliebig schlechte Ergebnisse liefern, wie folgendes Beispiel lehrt:

$p : \{p_1 = 1, \ldots, p_{n-1} = 1, p_n = b - 1\}$
$a : \{a_1 = 1, \ldots, a_{n-1} = 1, a_n = b = k \cdot n\}$ für ein beliebig großes $k \in \mathbb{N}$

Hier ist für $x = (X, p, a, b) : m^*(x) = b - 1 = k \cdot n - 1 > kn - k = k(n-1)$ (durch Wahl des letzten Elements), aber GreedyKnapsack liefert, da

$$\frac{p_1}{a_1} = 1 = \ldots = \frac{p_{n-1}}{a_{n-1}} > \frac{p_n}{a_n} = \frac{b-1}{b}$$

$$m_{Greedy}(x) = \sum_{i=1}^{n-1} p_i = n - 1, \text{ d.h. } \frac{m^*(x)}{m_{Greedy}(x)} > k$$

Eine andere heuristische Idee fürs Rucksackfüllen wäre, X einfach absteigend nach dem Profit zu sortieren. Nahezu dasselbe Beispiel wie eben (nur mit $a_n = \frac{n}{k}$) lehrt, dass auch hier beliebig schlechte Lösungen herauskommen können.

Der entsprechende Algorithmus GreedyKnapsack'(X, p, a, b) arbeitet genau so wie GreedyKnapsack, nur dass die erste Zeile durch

1. Sortiere X absteigend nach dem Profit p_i.

zu ändern ist.
Erstaunlicherweise hat der folgende Kombinationsalgorithmus eine Gütegarantie:

$\rightsquigarrow$ GreedyKnapsackKombi(X, p, a, b)

1. $Y_1 :=$ GreedyKnapsack (X, p, a, b)
2. $Y_2 :=$ GreedyKnapsack'(X, p, a, b)
3. Wenn $p(Y_1) \geq p(Y_2)$, dann liefere Y_1, sonst liefere Y_2.

Tatsächlich werden wir im Folgenden auf eine leicht variierte Version der Kombination der beiden heuristischen Ideen eingehen, nämlich auf:

GreedyKnapsack"(X, p, a, b)
Dieser Algorithmus arbeitet wie GreedyKnapsack, nur dass Schritt 4 ersetzt wird durch

4." Suche x_{max} mit $\forall i : p_i \leq p_{max}$ (maximaler Profit!)

5." Wenn $p(Y) \geq p_{max}$, dann liefere Y sonst liefere $\{x_{max}\}$.

Lemma 4.3 *Für alle Instanzen x gilt:*

$$m_{GreedyKnapsack''}(x) \leq m_{GreedyKnapsackKombi}(x)$$

Damit überträgt man die im folgenden Satz gezeigte Gütegarantie von Greedy-Knapsack" auf GreedyKnapsackKombi.
Abkürzend schreiben wir $m_G, m_{G''}$ für $m_{GreedyKnapsack}$ bzw. $m_{GreedyKnapsack''}$.

Satz 4.4 *Ist x eine MaximumKnapsack-Instanz, so ist*

$$\frac{m^*(x)}{m_{G''}(x)} < 2.$$

Beweis: Es sei j der Index des ersten Elements, welches der Greedy-Algorithmus GreedyKnapsack *nicht* in den Rucksack tut.
Es gilt:

$$\overline{p_j} := \sum_{i=1}^{j-1} p_i \leq m_G(x) \leq m_{G''}(x).$$

Ferner setzen wir:

$$\overline{a_j} := \sum_{i=1}^{j-1} a_i \leq b.$$

Behauptung: $m^*(x) < \overline{p_j} + p_j$.

Aus der Behauptung folgt die Aussage des Satzes, denn:

- Gilt $p_j \leq \overline{p_j}$, so ist $m^*(x) < 2\overline{p_j} \leq 2m_G(x) \leq 2m_{G''}(x)$.
- Gilt $p_j > \overline{p_j}$, so ist $m^*(x) < 2p_j \leq 2p_{max} \leq 2m_{G''}(x)$.

Warum gilt die Behauptung? Betrachte (kurzfristig) die folgende Verallgemeinerung des Rucksackproblems: Es sei nun erlaubt, „Bruchstücke" der x_i (mit entsprechend der Größe skaliertem Gewinn) in den Rucksack zu tun. Der Wert einer optimalen Lösung für das neue Problem ist sicher eine obere Schranke für $m^*(x)$. Wie man leicht einsieht, ist

$$\overline{p_j} + (b - \overline{a_j}) \cdot \frac{p_j}{a_j}$$

der maximale Wert einer Lösung des variierten Problems, wenn die x_i nach $\frac{p_i}{a_i}$ absteigend sortiert vorliegen und j wie oben definiert ist.

$$\Rightarrow \quad m^*(x) \leq \overline{p_j} + \underbrace{(b - \overline{a_j})}_{<a_j} \cdot \frac{p_j}{a_j} < \overline{p_j} + p_j.$$

Bemerkung 4.5 *Tatsächlich liefert Satz 4.4, dass der folgende Algorithmus eine $\frac{1}{2}$-Approximation von Maximum Knapsack ist:*

1. Sortiere X absteigend nach $\frac{p_i}{a_i}$.
2. $Y := \emptyset$; $i := 1$;
3. Solange $i \leq n$ und $a_i \leq b$, tue:
   $Y := Y \cup \{x_i\}$; $b := b - a_i$; $i := i + 1$.
4. Wenn $a_i \leq b$,[7] dann liefere Y,
   sonst, wenn $p_i \leq p(Y)$, dann liefere Y,
   sonst liefere $\{x_i\}$.

Hinweis: Verallgemeinerung von Problemen ist oft ein Ansatz, um Schranken für Approximationsgüte zu gewinnen.

Bemerkung 4.6 *Es gibt eigene Bücher zum Thema „Rucksackprobleme“, z.B. S. Martello, P. Toth: Knapsack Problems; Algorithms and Computer Implementations, Wiley, 1990.*

4.2 Unabhängige Mengen in Graphen

Beispiel 4.7 *Maximum Independent Set (MIS)*

I: *Graph* $G = (V, E)$
S: $\{U \subseteq V \mid \underbrace{E(G[U]) = \emptyset}_{\text{„unabh. Menge“}}\}$
m: $|U|$
opt: max

Mitteilung 4.8 *MIS_D ist NP-vollständig.*

Heuristische Idee:
Es ist günstig, möglichst kleingradige Knoten zu wählen.

$\rightsquigarrow$ GreedyIndependantSet($G = (V, E)$):

1. Setze $U := \emptyset$; $V' := V$.
2. Solange $V' \neq \emptyset$, tue:

   2a: x sei Knoten kleinsten Grades in $G(V')$.

   2b: $U := U \cup \{x\}$

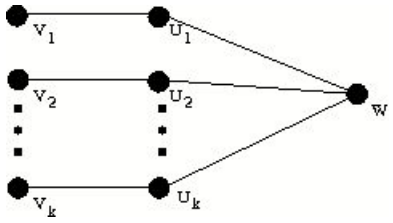
[7] D.h., wenn $i = n + 1$, also $Y = X$.

2c: $V' := V' \setminus N[x]$.

3. Liefere U

Der Greedy-Algorithmus kann beliebig schlechte Ergebnisse liefern, wie folgendes Beispiel lehrt.

Beispiel 4.9 *Betrachte die folgenden Graphenschar: G_k*



$V(G_k) = \{v_1, \ldots, v_k, u_1, \ldots, u_k, w\}$.
$G(\{v_1, \ldots, v_k\}) = K_k$. *Weitere Kanten:* $\{\{v_i, u_j\}, \{u_i, w\} \mid 1 \leq i, j \leq k\}$
$\Rightarrow \quad E(G(\{u_1, \ldots, u_k\})) = \emptyset$.
GreedyIndependentSet liefert aber w als kleinstgradigen Knoten und einen der Knoten v_i, also nur zwei Knoten als unabhängige Menge, während es doch (wie beschrieben) unabhängige Mengen der Größe k gibt.

Bemerkung 4.10 *Falls $P \neq NP$, so gibt es nachweislich keinen guten Näherungsalgorithmus für alle Graphen.*
Daher betrachten wir im folgenden speziellere Graphklassen, nämlich solche von beschränkter Dichte (dem Verhältinis von Kanten- zu Knotenzahl). Aus der Eulerformel folgt, dass planare Graphen eine durch 6 beschränkte Dichte haben.

Mathematischer Exkurs

Lemma 4.11 *Betrachte $x_1, \ldots, x_n \geq 0$; setze $\overline{x} = \frac{\sum x_i}{n}$. Dann gilt:*

$$\sum {x_i}^2 \geq n \cdot \overline{x}^2$$

Beweis: Für $\Delta_i := x_i - \overline{x}$ gilt

$$\sum \Delta_i = \sum (x_i - \overline{x}) = \sum x_i - n \cdot \overline{x} = 0$$

$$\rightsquigarrow \sum {x_i}^2 = \sum (\overline{x} + \Delta_i)^2 = \sum \overline{x}^2 + 2\overline{x} \underbrace{\sum \Delta_i}_{=0} + \sum \Delta_i^2 \geq \sum \overline{x}^2 = n \cdot \overline{x}^2$$

Lemma 4.12 $\forall x, y > 0 : \frac{x}{y} + \frac{y}{x} \geq 2$

Beweis:

$$\frac{x}{y} + \frac{y}{x} = \frac{x^2 + y^2}{x \cdot y} \overset{L\ 4.11}{\geq} \frac{2 \cdot (\frac{x+y}{2})^2}{x \cdot y} = \frac{(x+y)^2}{2xy} = \frac{x^2 + 2xy + y^2}{2xy} = \frac{1}{2}\left(\frac{x}{y} + \frac{y}{x}\right) + 1$$

Satz 4.13 *Es sei G ein Graph mit n Knoten und m Kanten. Es bezeichne $\delta = \frac{m}{n}$ seine Dichte. Der Wert $m_{Gr}(G)$, den GreedyIndependentSet(G) liefert, ist wenigstens $\frac{n}{2\delta+1}$.*

Beweis: Es sei x_i der in der i-ten Iteration der Solange-Schleife in Schritt 2a ausgewählten Knoten und d_i bezeichne den Grad von x_i. In Schritt 2c werden x_i und seine Nachbarn entfernt, d.h. insgesamt $d_i + 1$ viele Knoten. Dadurch werden wenigstens $\frac{d_i(d_i+1)}{2}$ viele Kanten entfernt, denn

- x_i ist ein Knoten minimalen Grades im aktuellen Graphen und
- von jedem der insgesamt $d_i + 1$ vielen entfernten Knoten gehen insgesamt wenigstens $d_i(d_i + 1)$ viele (nicht notwendig verschiedene) Kanten aus; im schlimmsten Fall —wenn nämlich jeder der $d_i + 1$ Knoten aus der Nachbarschaft $N[x_i]$ mit jedem anderen Knoten aus $N[x_i]$ verbunden ist und sonst mit keinem anderen Knoten außerhalb von $N[x_i]$— sind es $\frac{d_i(d_i+1)}{2}$ viele verschiedene Kanten.

Wenn wir über alle $m_{Gr}(G)$ viele Iterationen des Programmes aufsummieren, erhalten wir

(1) $\sum_{i=1}^{m_{Gr}(G)} \frac{d_i(d_i+1)}{2} \leq m = \delta \cdot n.$
Beachte: Es gilt i.a. „$\leq$“ nicht „$=$“, da $\frac{d_i(d_i+1)}{2}$ nur eine untere Schranke der Anzahl der im i-ten Schritt entfernten Kanten ist.

Da der Algorithmus hält, wenn alle Knoten entfernt worden sind, gilt außerdem:

(2) $\sum_{i=1}^{m_{Gr}(G)} (d_i + 1) = n.$

Addieren wir (2) und (zweimal) (1), so bekommen wir:

$$\sum_{i=1}^{m_{Gr}(G)} (d_i + 1)^2 \leq n(2\delta + 1).$$

Der Mittelwert der $(d_i + 1)$ ist

$$\frac{\sum (d_i + 1)}{m_{Gr}(G)} \overset{(2)}{=} \frac{n}{m_{Gr}(G)}$$

und Lemma 4.11 liefert damit

$$n(2\delta+1) \geq \sum (d_i+1)^2 \geq m_{Gr}(G) \cdot \frac{n^2}{(m_{Gr}(G))^2} = \frac{n^2}{m_{Gr}(G)}.$$

Also gilt:

$$m_{Gr}(G) \geq \frac{n}{2\delta+1}.$$

□

Satz 4.14 *Es sei G ein Graph mit n Knoten und m Kanten. Setze $\delta = \frac{m}{n}$. Der Wert $m_{Gr}(G)$, den GreedyIndependentSet(G) liefert, genügt*

$$\frac{m^*(G)}{m_{Gr}(G)} \leq \delta + 1$$

Die folgenden Bezeichnungen lehnen sich an die des vorigen Beweises an.
Beweis: Es sei V^* eine größtmögliche unabhängige Menge. Es bezeichne k_i die Anzahl der Knoten am V^*, die unter den $d_i + 1$ im i-ten Schritt des Algorithmus entfernten Knoten sind. Natürlich gilt jetzt:

(3) $\sum_{i=1}^{m_{Gr}(G)} k_i = |V^*| = m^*(G)$

Wir wollen jetzt Abschätzung (1) verbessern.
Per Definition sind alle Knoten einer unabhängigen Menge nicht untereinander verbunden. Daher gehen nicht nur $\frac{d_i(d_i+1)}{2}$ viele paarweise verschiedene Kanten mindestens von Knoten in $N[x_i]$ aus, sondern noch „zusätzlich“ wenigstens $\frac{k_i(k_i-1)}{2}$ viele, was

(1') $\sum_{i=1}^{m_{Gr}(G)} \frac{d_i(d_i+1)+k_i(k_i-1)}{2} \leq \delta n$

liefert.

Jetzt addieren wir (2), (3) und (zweimal) (1'), um

$$\sum_{i=1}^{m_{gr}(G)} (d_i+1)^2 + \sum_{i=1}^{m_{Gr}(G)} k_i^2 \leq n(2\delta+1) + m^*(G)$$

zu erhalten. Der Mittelwert der $(d_i + 1)$ ist $\frac{n}{m_{Gr}(G)}$ und der der k_i ist

$$\frac{\sum k_i}{m_{Gr}(G)} = \frac{m^*(G)}{m_{Gr}(G)},$$

sodass jetzt Lemma 4.11

$$n(2\delta+1) + m^*(G) \geq \frac{n^2 + (m^*(G))^2}{m_{Gr}(G)}$$

ergibt. Also ist:

$$m_{Gr}(G) \geq \frac{n^2 + (m^*(G))^2}{n(2\delta+1) + m^*(G)} = m^*(G) \cdot \frac{(\frac{n}{m^*(G)}) + (\frac{m^*(G)}{n})}{2\delta + 1 + (\frac{m^*(G)}{n})}$$

$$\overset{m^*(G)\leq n \text{ u. } L\ 4.12}{\geq} m^*(G) \cdot \frac{2}{2\delta + 1 + \frac{n}{n}} = m^*(G) \cdot \frac{1}{\delta + 1}.$$

□

Bemerkungen zu MIS und verwandten Problemen

1. Für manche Graphenklassen, z.B. die der Bäume, liefert GreedyIndependentSet stets eine optimale Lösung. Man sieht also, dass die genaue Spezifikation der Klasse I zulässiger Instanzen wichtig ist.
2. GreedyIndependentSet enthält in Schritt 2 einen gewissen (durch beliebige Wahl aufgelösten) Nichtdeterminismus. Selbst, wenn man für Graphen weiß, dass es eine Wahlmöglichkeit in Schritt 2a gibt, sodass seine Variante von GreedyIndependentSet für diese Graphen die optimale Lösung findet, ist das auf solche Graphen eingeschränkte MIS_D-Problem (immer noch) NP-vollständig.

   Literatur für 1. und 2.: H. L. Bodlaender, D. M. Thilikos, K. Yamazaki: It is hard to know when greedy is good for finding independent sets. Information Processing Letters 61 (1997), 101–106.
3. Das Auffinden maximaler Cliquen in einem Graphen (MaxClique) ist genauso schwer —auch im approximativen Sinne— wie das Auffinden größtmöglicher unabhängiger Mengen, denn $C \subseteq V$ ist maximale Clique in $G = (V, E)$ genau dann, wenn C größtmögliche unabhängige Menge in $G^c = (V, E^c)$ ist, $E^c = \{\{v, v'\} \mid v, v', v \in V, v \neq v', \{v, v'\} \notin E\}$. Direkt implementiert ist die heuristische Idee für MaxClique, möglichst großgradige Knoten zu wählen.
4. Unser Approximationsalgorithmus liefert für die Klasse G_d der Graphen mit Maximalgrad d eine unabhängige Menge der Mindestgröße $\frac{n}{d+1}$. Der „Restgraph“ hat Maximalgrad $d - 1$ (denn sonst könnte man den Knoten mit Grad d noch in die unabhängige Menge nehmen, da keiner seiner Nachbarn in diese Menge hinzugenommen wurde!), sodass sich auf ihn wiederum der Approximationsalgorithmus anwenden ließe. Auf diese Weise kann man eine $(d + 1)$-Färbung des Ursprungsgraphen erzeugen.

4.3 Das Handelsreisendenproblem

Bislang haben wir Beispiele für Greedy-Algorithmen zu Maximierungsproblemen untersucht. Jetzt wollen wir die Technik auch auf Minimierungsprobleme anwenden.

Beispiel 4.15 *Minimum Traveling Salesperson MTS. (Handelsreisendenproblem)*

I : vollständiger gerichteter Graph $G = (V, E), V = \{v_1, \ldots, v_n\}$ mit positiven Kantengewichten. (Abstandsfunktion; $D(v_i, v_j)$-Matrix über $\mathbb{Q}^+$)
S : $\{(v_{i_1}, \ldots v_{i_n}) \in V^n \mid [\forall j = 1, \ldots, n-1 : (v_{i_j}, v_{i_{j+1}}) \in E \wedge]$
$(i_1, \ldots, i_n)$ ist Permutation von $(1, \ldots, n)\}$
$\hookrightarrow$ alternative Schreibweise: Wort der Länge n über V
m : $D(v_{i_1}, \ldots v_{i_n}) = \sum_{j=1}^{n-1} D(v_{i_j}, v_{i_{j+1}}) + D(v_{i_n}, v_{i_1})$
(„Länge der Tour“)
opt: min

MTS ist sehr beliebt und populär. *G.Reinelt, The Traveling Salesman, Computational Solutions for TSP Applications, LNCS 840, 1994,* ist ganz MTS „gewidmet“.

Ein kleines Beispiel:

Ein optimaler Reiseweg eines Handlungsreisenden durch die 15 größten Städte Deutschlands ist gesucht. Der angegebene Weg ist der kürzeste unter 43 589 145 600 möglichen.

Spezialfall: metrisches MTS (kurz: MMTS), d.h. D ist Metrik, also:

- $\forall u, v : D(u, v) = D(v, u)$ (also „ungerichteter Graph“)
- $\forall u, v, w : D(u, w) \leq D(u, v) + D(v, w)$ ($\triangle$-Ungleichung)
- $\forall u, v : D(u, v) = 0 \Leftrightarrow u = v$

Bemerkung 4.16 *Aus der $\triangle$-Ungleichung folgt unmittelbar:*
Ist $f' = (v_{i_{j_1}}, \ldots, v_{i_{j_k}})$ eine Teilfolge von $f = (v_{i_1}, \ldots, v_{i_n})$,
so gilt $D(f') \leq D(f)$.

Mitteilung 4.17 *MTS_D und $MMTS_D$ sind NP-vollständig.*

Für diesen Spezialfall analysieren wir folgenden Algorithmus MMTS_{NN}:[8]

$\text{MMTS}_{NN}(G = (V, E), D)$

1. Setze $P := v$ für irgendein $v \in V$ und $P_{end} := v$.

[8] NN steht für „nächster Nachbar“.

2. Setze $V' := V \setminus \{v\}$.
3. Solange $V' \neq \emptyset$ tue:

   3a. v sei ein Knoten aus V', sodass
   $\forall u \in V' : D(P_{end}, v) \leq D(P_{end}, u)$

   3b. $P_{end} := v$; $P := P \cdot P_{end}$.
   [$\cdot$ steht für die Konkatenation.]

   3c. $V' := V' \setminus \{v\}$.
4. Liefere P.

Bezeichnung: Für eine Tour $f \in V^*$ sei

$$E(f) = \{(u, v) \in V^2 \mid \exists x, z \in V^* : f = xu \cdot vz \text{ oder } f = vxu\}.$$

Dann ist $D(f) = D(E(f))$.

Durch die Verwaltung der Menge V' ist klar, dass P ein zu keinem Zeitpunkt zweimal den selben Knoten beinhaltet. M.a.W: das zurückgelieferte P ist in jedem Fall zulässig. Als Gütegarantie wollen wir den folgenden Satz beweisen:

Satz 4.18 *Es sei $(G = (V, E), D)$ eine Instanz von MMTS, $|V| = n$. Ist $m_{NN}(G, D)$ die Lösung der von MMTS$_{NN}(G, D)$ gelieferten Tour, so gilt:*

$$\frac{m_{NN}(G, D)}{m^*(G, D)} \leq \frac{1}{2}(\lceil \log n \rceil + 1).$$

Satz 4.18 folgt sofort aus nachfolgendem Lemma (wegen der $\triangle$-Ungleichung), wenn wir für die von MMTS$_{NN}$ gelieferte Tour $P = v_{i_1}, \ldots v_{i_n}$ festlegen,

$$l(v_{i_j}) = \begin{cases} D(v_{i_j}, v_{i_{j+1}}), & 1 \leq j < n, \\ D(v_{i_n}, v_{i_1}), & j = n. \end{cases}$$

Augenscheinlich ist

$$m_{NN}(G, D) = \sum_{j=1}^{n} l(v_j)$$

Lemma 4.19 *Es sei (G, D) eine Instanz von MMTS, und sei $G = (V, E)$, $n = |V|$. Existiert eine Abbildung $l : V \to \mathbb{Q}$ mit*

1. $\forall u, v \in V, u \neq v : D(u, v) \geq \min(l(u), l(v))$
2. $\forall v \in V : l(v) \leq \frac{1}{2} m^*(G, D)$,

so gilt:

$$\sum_{v \in V} l(v) \leq \frac{1}{2}(\lceil \log n \rceil + 1) \cdot m^*(G, D)$$

Beweis: Es seien $v_1, \ldots, v_n$ absteigend gemäß ihrer l-Werte sortiert. Wir beweisen zunächst, dass $m^*(G,D) \geq 2 \cdot \sum_{i=k+1}^{\min(2k,n)} l(v_i)$ für alle $1 \leq k \leq n$ gilt. Dazu sei P^* eine optimale Tour der Länge $m^*(G,D)$. Betrachte die Teilmenge $V_k = \{v_i \mid 1 \leq i \leq \min(2k,n)\}$ von V. Bezeichne P_k diejenige Teiltour von P^*, die entsteht, wenn man nur die „Städte“ aus V_k in der durch P^* vorgegebenen Reihenfolge durchläuft. Wegen obiger Bemerkung ist $D(P_k) \leq D(P^*) = m^*(G,D)$. Wegen der ersten Voraussetzung gilt ferner:

$$D(P_k) \geq \sum_{(u,v) \in E(P_k)} \min(l(u), l(v)) = \sum_{v_i \in V_k} \alpha_i l(v_i)$$

für gewisse $\alpha_i \in \{0,1,2\}$. Genauer gilt:

$$\alpha_i = |\{v_j \in V_k \mid \{v_i, v_j\} \in E \wedge j < i\}|,$$

denn die V_i werden als absteigend sortiert angenommen.
Außerdem ist

$$\sum_{v_i \in V_k} \alpha_i = |V_k| \leq 2k.$$

Wegen der absteigenden Sortierung der v_i können wir

$$\sum_{v_i \in V_k} \alpha_i l(v_i) \geq 2 \cdot \sum_{i=k+1}^{\min(2k,n)} l(v_i)$$

abschätzen, indem wir

$$\alpha_1 = \ldots = \alpha_k = 0 \text{ und } \alpha_{k+1} = \ldots = \alpha_{\min(2k,n)} = 2$$

annehmen. Damit ist

$$m^*(G,D) \geq D(P_k) \geq 2 \cdot \sum_{i=k+1}^{\min(2k,n)} l(v_i)$$

gezeigt.
Summieren wir diese Ungleichung für $k = 2^j$, $j = 0, \ldots, \lceil \log n \rceil - 1$, so erhalten wir:

$$\lceil \log n \rceil m^*(G,D) = \sum_{j=0}^{\lceil \log n \rceil - 1} m^*(G,D) \geq \sum_{j=0}^{\lceil \log n \rceil - 1} \left(2 \cdot \sum_{i=2^j+1}^{\min(2^{j+1},n)} l(v_i) \right)$$

$$= 2 \cdot \sum_{i=2}^{n} l(v_i)$$

Mit der zweiten Voraussetzung, aus der $m^*(G,D) \geq 2 \cdot l(v_1)$ folgt, ergibt sich die Behauptung des Lemmas. □

Warum dürfen wir das Lemma anwenden, um Satz 4.18 zu zeigen?
Zu Voraussetzung 1: Es seien v_{i_j}, v_{i_k} zwei Städte der von MMTS_{NN} gelieferten Tour. Ist $j < k$; dann ist $l(v_{i_j}) = D(v_{i_j}, v_{i_{j+1}}) \leq D(v_{i_j}, v_{i_k})$, denn sonst wäre v_{i_k} statt $v_{i_{j+1}}$ vom Algorithmus MMTS_{NN} gewählt worden. Ist $j > k$, dann gilt analog

$$l(v_{i_k}) = D(v_{i_k}, v_{i_{k+1}}) \leq D(v_{i_k}, v_{i_j}).$$

Zu Voraussetzung 2: Betrachte einen Knoten $v \in V$. Nach Definition von l gibt es einen „benachbarten“ Knoten $u \in V$ auf der MMTS$_{NN}$-Tour mit $l(v) = D(u, v)$. Gemäß einer optimalen Tour P^* gibt es zwischen u und v zwei disjunkte Wege; jeder dieser Wege hat wegen der $\triangle$-Ungleichung mindestens die Länge $D(u, v)$. Daher ist

$$m^*(G, D) = D(P^*) \geq 2 \cdot D(u, v) = 2 \cdot l(v).$$

Bemerkung 4.20 *Logarithmische, nicht konstante Approximationsgüte!*
Später (Kap. 11 besser!

Die Popularität von TSP ermisst sich leicht im Internet.
Oft werden hierfür auch *Metaheuristiken* getestet, wie *genetische Algorithmen*.

5 Partitionsprobleme

Bei einem Partitionsproblem ist im Allgemeinen eine Liste von Gegenständen $L = \{x_1, \ldots, x_n\}$ gegeben, die Menge L ist so zu unterteilen, dass den problemspezifischen Einschränkungen Genüge getan ist; die entsprechende Partition wird ausgegeben. Ein allgemeines Schema einer (Greedy-)Heuristik für Partitionsprobleme ist das Folgende:

1. Sortiere L (gemäß eines problemspezifischen Kriteriums)
   Sei $(x_1, \ldots, x_n)$ die sortierte Liste.
2. $P := \{\{x_1\}\}$;
3. Für $i = 2$ bis n tue
   Falls x_i zu einer Menge p aus P hinzugefügt werden kann
   (gemäß einem problemabhängigen Kriterium),
   so füge x_i zu p hinzu
   sonst $P := P \cup \{\{x_i\}\}$.
4. Liefere P zurück.

In obigem Schema werden die Gegenstände der Liste der Reihe nach abgearbeitet; einmal getroffene Entscheidungen werden später nie revidiert.

5.1 Scheduling

Beispiel 5.1 *Scheduling auf identischen Maschinen (MSIM)*

I : *Menge von Tasks T, Anzahl p der identischen Maschinen,*
Dauer l_j der Ausführung von Task $t_j \in T$
(Tasks sind nicht unterbrechbar!)
S : $\{f : T \to \{1, \ldots, p\}\}$ *(oder: Partition $P = \{m_1, \ldots, m_p\}, \bigcup_{i=1}^{p} m_i = T$)*
m : *Gesamtausführungszeit von T:* $m(f) = max\{\sum_{t_j \in T, f(t_j)=i} l_j \mid 1 \leq i \leq p\}$
opt : *min.*

Bemerkung 5.2 *Historischer Ausgangspunkt für Näherungsalgorithmen: Graham 1969, vgl. das 1. und 2. Kapitel im Buch von Hochbaum!*

List-Scheduling:

Heuristische Idee: Gib den nächsten Task an die Maschine, die mit ihrer bisherigen Arbeit am schnellsten fertig ist.

Satz 5.3 *Ist $x = (T, p, l)$ eine Instanz von MSIM, so findet LS eine Näherungslösung vom Wert $m_{LS}(x)$ (unabhängig von der Ordnung von T), welche*

$$m_{LS}(x)/m^*(x) \leq (2 - \frac{1}{p})$$

erfüllt.

Beweis: Es sei $T = \{t_1, \ldots, t_n\}$.
Setze $W := \sum_{k=1}^{n} l_k$. Klar: $m^*(x) \geq W/p$.
Wir nehmen an, LS habe bereits j Tasks zugewiesen. Bezeichne jetzt $A_i(j)$ die Zeit, die notwendig ist, um die davor der Maschine i zugewiesenen Tasks zu beenden, d.h.

$$A_i(j) = \sum_{1 \leq k < j, f(t_k) = i} l_k$$

für das von LS konstruierte Schedule f.
Sei h eine Maschine mit $A_h(n+1) = m_{LS}(x)$ und sei j der Index des Tasks, der als letztes h zugewiesen wurde. t_j wurde von LS einer Maschine mit minimaler Auslastung zugewiesen, d.h.

$$\forall 1 \leq i \leq p : A_i(n+1) \geq A_i(j) \geq A_h(j) = A_h(n+1) - l_j.$$

Damit gilt:

$$W = \sum_{i=1}^{p} A_i(n+1) \geq (p-1) \cdot (A_h(n+1) - l_j) + A_h(n+1) = p \cdot (A_h(n+1) - l_j) + l_j.$$

Also gilt:

$$m_{LS}(x) = A_h(n+1) \leq \frac{W}{p} + \frac{(p-1)l_j}{p}.$$

Wegen $m^*(x) \geq W/p$ und $m^*(x) \geq l_j$ folgt

$$m_{LS}(x) \leq m^*(x) + \frac{p-1}{p} m^*(x) = \left(2 - \frac{1}{p}\right) m^*(x), \quad \text{denn } m^*(x) \geq \max\{W/p, l_j\} \quad \square$$

Die in Satz 5.3 angegebene Schranke ist scharf, wie das folgende Beispiel lehrt!

Beispiel 5.4 *Ggb: p. Betrachte die Instanz von MSIM mit $p^2 - p$ Tasks der Länge 1 und einem Task der Länge p, zu verteilen auf p Maschinen. Optimal wäre es, die $p(p-1)$ kurzen Tasks auf $p-1$ Maschinen zu verteilen und den großen Task auf die verbleibende. Der optimale Wert wäre also gleich p, denn ein Gesamtarbeitsvolumen von p^2 benötigt auf p Maschinen wenigstens p Schritte.*

Wenn der große Task jedoch der letzte der Liste ist, werden zunächst die $p(p-1)$ kleinen Tasks gleichmäßig auf die p Maschinen verteilt, und dann bekommt irgendeine der Maschinen noch weitere p Schritte Arbeit, nämlich durch den großen Task. Daher liefert List Scheduling dann eine Lösung vom Wert $2p - 1$.

Bemerkung 5.5 *List Scheduling lieferte im vorigen Beispiel 5.4 (nur) bei einer „bösen“ Eingabereihenfolge (in der nämlich erst die kleinen Tasks verteilt wurden) ein schlechtes Ergebnis.*

Positiv formuliert ist List Scheduling per se ein Online-Algorithmus.

Im Folenden versuchen wir, die Eingabereihenfolge durch geeignetes Sortieren zu verbessern.

Schließlich möchten wir noch auf den Aspekt der *Online-Algorithmen* hinweisen, bei denen der Algorithmus nicht den Zugriff auf die gesamte Instanz zu jeder Zeit hat, sondern ihn nur “bruchstückweise” sieht und dann nach dem Bekanntwerden einer neuen Einzelheit auf unabänderliche Weise eine neue Teilentscheidung fällen muss. Viele der hier besprochenen Algorithmen tragen diesen Wesenszug.

Neue Heuristische Idee: LPT (Largest Processing Time)

Sortiere eingangs die Tasks in absteigender Folge gemäß ihrer Laufzeit, d.h. es gilt dann: $l_1 \geq l_2 \geq \ldots \geq l_n$. Danach wird der LIST Scheduling Algorithmus auf die sortierte Liste angewendet. $\rightsquigarrow$ Algorithmus LPTLS

Satz 5.6 *Ist $x = (T = \{t_1, \ldots t_n\}, p, l)$ eine Instanz von MSIM, so findet LPTLS eine Näherungslösung vom Wert $m_{LPTLS}(x)$, welche*

$$\frac{m_{LPTLS}(x)}{m^*(x)} \leq \left(\frac{4}{3} - \frac{1}{3p}\right)$$

erfüllt.

Beweis: Nach der Sortierung ist l_n eine der kürzesten Tasklängen.
Wir unterscheiden zwei Fälle:

1. Fall: $l_n > m^*(x)/3$. Dann können höchstens 2 Tasks auf eine Maschine gelegt werden. LPTLS liefert hier sogar eine optimale Lösung, wie man wie folgt einsieht.

   a) Zunächst kommen $\tilde{p}$ Tasks der Länge $> m^*(x)/2$. Da $m^*(x)$ die optimale Länge ist, werden auch in jeder optimalen Lösung nie zwei solcher Tasks auf einer Maschine abgearbeitet werden, und es stehen auf jeden Fall mehr Maschinen zur Verfügung als es Tasks der Länge $> m^*/2$ gibt (also $\tilde{p}$). LPTLS macht also „nichts falsch“.

   b) Dann kommen Tasks der Länge l, $m^*(x)/2 \geq l > m^*(x)/3$. Die ersten $\min(2(p - \tilde{p}), n - \tilde{p})$ dieser Tasks werden von LPTLS auf die $p - \tilde{p}$ Maschinen verteilt, die unter a) noch keine Arbeit bekamen. Wegen $l_n > m^*(x)/3$ sind diese $\tilde{p}$ Maschinen somit soweit ausgefüllt, dass sie keine weiteren Tasks mehr aufnehmen können. Wenn $n - \tilde{p} = \min(2(p - \tilde{p}), n - \tilde{p})$, so ist die Zuweisung optimal. Es sei nun $n > 2p - \tilde{p}$. Zwar

könnten auch andere Maschinenzuweisungen zu optimalen Lösungen führen, aber dann wäre eine Lösung, die durch „Tausch“ von einem der $2p - \tilde{p}$ betrachteten Tasks, die einer der ersten belasteten $\tilde{p}$ Maschinen zugeordnet wurde, mit einer der $n - 2p - \tilde{p}$ „letzten“ Tasks, die dann evtl. einer der $p-\tilde{p}$ anderen Maschinen zugeordnet worden wäre, immer noch optimal; sollten alle $p - \tilde{p}$ nach Schritt a) verfügbaren Maschinen nur einen Task bekommen haben, durch den betreffenden optimalen Algorithmus, so ist die „Umverteilbarkeit“ ebenso klar. Mithin gibt es eine optimale Lösung, die die ersten $n - 2(p - \tilde{p})$ Tasks genau so wie LPTLS verteilt.
Sollten noch Tasks verbleiben, so müssen sie den $\tilde{p}$ Maschinen zugewiesen werden, denen LPTLS zuerst Arbeit unter a) zugewiesen hatte. Ein „Tauschargument“ ähnlich dem obigen zeigt, dass die LPTLS-Lösung optimal ist.

Für das abschließende Argument übernehmen wir die Bezeichnungsweise vom Beweis von Satz 5.3, insbesondere betreffend h und j.

2. Fall: $l_n \leq m^*(x)/3$.

Fall 2a: $l_j > m^*(x)/3$: Gilt $l_j > l_n$, so betrachte die „abgeschnittene“ Instanz $x' = (T' = \{t_1, \ldots, t_j\}, p, l' = (l_1, \ldots, l_j))$, wobei wir die $\{t_1, \ldots, t_n\}$ als absteigend sortiert annehmen. Es gilt:

$$\frac{m_{LPTLS}(x)}{m^*(x)} = \frac{m_{LPTLS}(x')}{m^*(x)} \leq \frac{m_{LPTLS}(x')}{m^*(x')} = 1$$

nach Fall 1, denn $m^*(x) \geq m^*(x')$.

Fall 2b: $l_j \leq m^*(x)/3$. Das Argument vom Beweis von Satz 5.3 zeigt:

$$\begin{aligned} m_{LPTLS}(x) &\leq \frac{W}{p} + \frac{(p-1)}{p} \cdot l_j \leq m^*(x) + \frac{p-1}{p} \cdot \frac{1}{3} m^*(x) \\ &\leq \left(\frac{3p + p - 1}{3p}\right) m^*(x) \\ &\leq \left(\frac{4}{3} - \frac{1}{3p}\right) m^*(x) \quad \square \end{aligned}$$

Über Scheduling gibt es eigene Bücher wie:
M. Pinedo: Scheduling: Theory, Algorithms and System, Prentice-Hall, 1995.
P. Brucker: Scheduling Algorithms: 2nd. ed. Springer, 1998.
Siehe auch Kap. 1 im Buch von Hochbaum!

5.2 Bin Packing

Beispiel 5.7 *Auffüllen von Behältern gleicher Kapazität (Minimum Bin Packing MBP)*

I : $I = \{a_1, \ldots, a_n\}, a_i \in (0,1] \cap \mathbb{Q}, 1 \leq i \leq n$
S : *Partition* $P = \{B_1, \ldots, B_k\}$ *von* I *mit* $\forall 1 \leq j \leq k : \sum_{a_i \in B_j} a_i \leq 1$
m : $k = |P|$
opt : *min.*

Einfachste Heuristik: NextFit.
I wird in der vorgegebenen Reihenfolge abgearbeitet. a_1 wird in B_1 platziert. Ist im Verlauf des Algorithmus B_j der zuletzt benutzte Behälter und wird a_i betrachtet, so wird a_i an B_j zugewiesen, falls es noch passt, und widrigenfalls an B_{j+1}.

Satz 5.8 *Ist* $I = \{a_1, \ldots, a_n\}$ *eine Instanz von MBP, so findet NextFit eine Lösung vom Wert* $m_{NF}(I)$ *mit* $m_{NF}(I)/m^*(I) \leq 2$.

Beweis: Sei $A = \sum_{i=1}^{n} a_i$. Da für je zwei aufeinanderfolgende Behälter B_j, B_{j+1} nach Ablauf von NextFit gilt, dass die Summe der Größen $a(B_j)$ und $a(B_{j+1})$ der sich in B_j und B_{j+1} befindenden Gegenstände größer als Eins ist (sonst wären sie ja alle in B_j gelandet), gilt (mit $k = m_{NF}(I)$):

$$2A = a(B_1) + \sum_{l=1}^{k-1} (a(B_l) + a(B_{l+1})) + a(B_k) > a(B_1) + a(B_k) + k - 1$$

$$\rightsquigarrow 2A > k - 1 = m_{NF}(I) - 1$$

Andererseits ist $m^*(I) \geq \lceil A \rceil$. (Einheitsvolumen der Behälter!)

$$\rightsquigarrow m_{NF}(I) < 2m^*(I) + 1.$$

Da die Werte ganzzahlig sind, folgt $m_{NF}(I) \leq 2m^*(I)$. □

Bemerkung 5.9 *Die Schranke lässt sich verbessern, wenn bekannt ist, dass die Gegenstände eine Maximalgröße von* $\alpha \leq \frac{1}{2}$ *haben; dann gilt:*

$$m_{NF}(I)/m^*(I) \leq \frac{1}{1-\alpha}.$$

Bemerkung 5.10 *Die für NextFit angegebene Schranke ist asymptotisch nicht verbesserbar.*
Betrachte dazu $I = \{a_1, \ldots, a_{4n}\}$ *mit*

$$a_i = \begin{cases} \frac{1}{2}, & i \text{ ungerade} \\ \frac{1}{2n}, & i \text{ gerade} \end{cases}$$

Offenbar ist

$$m_{NF}(I) = 2n$$

und

$$m^*(I) = n + 1.$$

NextFit ist in doppeltem Sinne ein Online-Algorithmus (und lässt sich daher auch in Linearzeit mit konstantem Speicher implementieren):

1. Die Gegenstände werden „der Reihe nach“ (so wie sie in der Eingabe stehen) betrachtet. Insbesondere werden sie nicht sortiert.
2. Ein Behälter, in den einmal ein Gegenstand *nicht* hineingetan wurde, wird nie wieder betrachtet. Er wird sozusagen „geschlossen“.

Wenn man Punkt 2 fallenlässt, braucht man evtl. mehr Speicher, um sich alle möglichen „geöffneten“ Behälter merken zu können. Der Algorithmus FirstFit öffnet erst dann einen neuen Behälter, wenn der betrachtete Gegenstand a_i nicht in einen der bereits geöffneten Behälter passt. Passt a_i in einen der geöffneten Behälter, so tut FirstFit ihn in den ersten passenden (daher der Name).

Mitteilung 5.11 *Ist $I = \{a_1, \ldots, a_n\}$ eine Instanz von MBP, so findet FirstFit eine Lösung vom Wert $m_{FF}(I)$ mit*

$$m_{FF}(I) \leq \lceil 1,7 \cdot m^*(I) \rceil \text{ [siehe Kapitel 2 im Buch von Hochbaum]}$$

Noch besser steht „FirstFitDecreasing“ da; dieser Algorithmus lässt auch noch Punkt 1 fallen und sortiert zunächst die Eingabe, bevor der FirstFit-Algorithmus angewendet wird. In seiner Doktorarbeit hat D. S. Johnson 1973 auf über 70 Seiten folgendes bewiesen:

Mitteilung 5.12 *Ist $I = \{a_1, \ldots a_n\}$ eine Instanz von MBP, so findet FirstFitDecreasing eine Lösung vom Wert $m_{FFD}(I)$ mit*

$$m_{FFD}(I) \leq \frac{11}{9} m^*(I) + 4$$

B. S. Baker hat 1985 im *Journal of Algorithms 6: 49–70* einen kürzeren Beweis für

$$m_{FFD} \leq \frac{11}{9} m^*(I) + 3$$

gegeben.
Wir beschränken uns im folgenden darauf, eine schlechtere Schranke zu beweisen.

Satz 5.13 *Ist $I = \{a_1, \ldots, a_n\}$ eine Instanz von MBP, so findet FirstFitDecreasing eine Lösung vom Wert $m_{FFD}(I)$ mit*

$$m_{FFD}(I) \leq 1,5 \cdot m^*(I) + 1.$$

Beweis: Betrachte $I = A \cup B \cup C \cup D$ mit

$$\begin{aligned} A &= \left\{ a_i \;\middle|\; a_i > \frac{2}{3} \right\}, \\ B &= \left\{ a_i \;\middle|\; \frac{2}{3} \geq a_i > \frac{1}{2} \right\}, \\ C &= \left\{ a_i \;\middle|\; \frac{1}{2} \geq a_i > \frac{1}{3} \right\}, \\ D &= \left\{ a_i \;\middle|\; \frac{1}{3} \geq a_i \right\}. \end{aligned}$$

Gibt es einen Behälter, der ausschließlich mit Gegenständen aus D gefüllt wurde, so sind alle Behälter —mit möglicher Ausnahme des zuletzt geöffneten— zu wenigstens 2/3 gefüllt, woraus die Behauptung hier folgt, denn

$$m^*(I) \geq \left\lceil \sum a_i \right\rceil \geq \frac{2}{3} m_{FFD}(I).$$

Gibt es keinen Behälter, der nur Gegenstände aus D enthält, so gilt:

$$m_{FFD}(I) = m_{FFD}(I \setminus D),$$

da alle Behälter auch Gegenstände aus $I \setminus D$ enthalten und die Vorsortierung bewirkt, dass Gegenstände aus $I \setminus D$ bei der Instanz I nicht anders als bei der Instanz $I \setminus D$ behandelt werden. Wir zeigen nun $m_{FFD}(I \setminus D) = m^*(I \setminus D)$, woraus $m_{FFD}(I) \leq m^*(I)$ folgt. Jedenfalls ist $m(I \setminus (D \cup A)) + |A| = m(I \setminus D)$ für $m \in \{m^*, m_{FFD}\}$. Gegenstände von B und C werden nun günstigenfalls „paarweise“ gegliedert, und zwar „kleine“ aus C und mit „großen“ aus B. Das genau macht m_{FDD}, woraus $m^*(I \setminus (D \cup A)) = m_{FFD}(I \setminus (D \cup A))$ folgt. □

Bemerkung 5.14 *Um die asymptotische Schärfe der Mitteilung einzusehen, betrachte für $n > 0$ (n durch 6 teilbar), die Instanz I_{5n}, die folgende Gegenstände enthält.*

n	*Gegenstände der Größe* $1/2 + \epsilon$	*(Typ 1)*
n	*Gegenstände der Größe* $1/4 + 2\epsilon$	*(Typ 2)*
n	*Gegenstände der Größe* $1/4 + \epsilon$	*(Typ 3)*
$2n$	*Gegenstände der Größe* $1/4 - 2\epsilon$	*(Typ 4)*

Bestmöglicherweise braucht man $\frac{3}{2}n$ Behälter:

n Behälter sind mit je einem Gegenstand vom Typ 1, 3 und 4 gefüllt,
$\frac{n}{2}$ Behälter sind mit je zwei Gegenständen vom Typ 2 und zweien vom Typ 4 gefüllt.
Dagegen würde FFD wie folgt füllen:

n Behälter mit je einem Gegenstand vom Typ 1 und 2,
$\frac{n}{3}$ Behälter mit je drei Gegenständen vom Typ 3,
$\frac{n}{2}$ Behälter mit je vier Gegenständen vom Typ 4.

Also gilt:

$$\frac{m_{FFD}(I_{5n})}{m^*(I_{5n})} = \frac{\frac{11}{6}n}{\frac{3}{2}n} = \frac{11}{9}.$$

Aufgrund der Paxisrelevanz von MBP (z.B. Containerbeladung, Zuschnittprobleme aus Standardstücken) und seiner (oft mehrdimensionaler) Varianten ist die Literatur zu MBP immens, vgl. auch das 2. Kapitel im Buch von Hochbaum.

5.3 Graphenfärben

ist ein Problem, das nicht sofort als Partitionsproblem „vorliegt“.

I : Graph $G = (V, E)$
S : $\{f : V \to \mathbb{N} \mid f$ „Färbung“, d.h. $\forall\{v_1, v_2\} \in E : f(v_1) \neq f(v_2)\}$
m : $|f(V)|$
opt : min.

Graphenfärben ist insofern ein Partitionierungsproblem, als dass V in „gleichfarbige Mengen“ unterteilt werden soll.
Die entsprechende Partitionierungsheuristik würde also einen neuen „Knoten“ v_i zu einer „alten“ Farbmenge V_c (Partitionsmenge) hinzufügen, sofern kein Nachbar von v_i bereits in V_c liegt. Widrigenfalls würde eine neue Farbe gewählt.

Satz 5.15 *Betrachte die Knotenmenge des Eingabegraphen $G = (V, E)$ als geordnet: $(v_1, \ldots, v_n)$. (Dies ist auch die Reihenfolge, in der die Partitionsheuristik arbeitet.)*
Es sei $G_i = G(\{v_1, \ldots, v_i\})$; $d_i(v)$ bezeichne den Grad von v in G_i.
Die Partitionsheuristik benötigt dann höchstens

$$\max_{1 \leq i \leq n} \min(d_n(v_i), i - 1) + 1$$

viele Farben zum Färben von G.

Beweis: Es sei k_i die Farbanzahl, die der Algorithmus zum Färben von G_i benötigt. Wird keine neue Farbe eingeführt, so gilt in dieser Notation $k_i = k_{i-1}$.
Sonst gilt: $k_i = k_{i-1} + 1$ und $d_i(v_i) \geq k_{i-1}$ (sonst bräuchte man keine neue Farbe).
Wir wollen zeigen:

$$k_j \leq \max_{1 \leq i \leq j} (d_i(v_i)) + 1$$

$j = 0, 1$: $\surd$
$j \to j + 1$: Ist $k_{j+1} = k_j$, so ist die Behauptung per Induktionsannahme wahr.
Ist $k_{j+1} = k_j + 1$, so gilt:
$k_{j+1} = k_j + 1 \leq d_{j+1}(v_{j+1}) + 1 \leq \max_{1 \leq i \leq j+1} d_i(v_i) + 1.$

Wegen $d_i(v_i) \leq d_n(v_i)$ und $d_i(v_i) \leq i-1$ folgt die Behauptung. □

Folgerung 5.16 *Ist Δ der Maximalgrad von G, so liefert die Partitionsheuristik eine Färbung von G mit höchstens $\Delta + 1$ vielen Farben bei beliebiger Anordnung der Knoten.*

Um den Ausdruck

$$\max_{1 \leq i \leq n} \min(d_n(v_i), i-1)$$

möglichst klein zu bekommen, erscheint das Anordnen der Knoten des Graphen nach absteigendem Grad sinnvoll.
Leider gestattet ein solcher Algorithmus im Allgemeinen keine vernünftigen Güteabschätzungen (und dies ist kein Zufall, wie wir noch später sehen werden).

Betrachte nämlich $G = (V, E)$ mit

$$V = \{x_1, \ldots, x_n, y_1, \ldots, y_n\}, \quad E = \{\{x_i, y_j\} \mid i \neq j\}$$

sowie die Knotenanordnung

$$(x_1, y_1, x_2, y_2, \ldots)$$

Obwohl der Graph zweifärbbar ist (in „x“- und „y“-Farben) benötigt (auch die vorsortierte) Partitionsheuristik hier n Farben (gemäß den Knotenindizes).

6 Lokale Suche

Die lokale Suche ist ein beliebte Heuristik, die nach folgendem Schema verfährt:

1. Initialisiere y mit (irgendeiner) zulässigen Lösung
2. Solange es in der „Nachbarschaft“ von y eine Lösung z gibt, die besser als y ist, setze $y := z$.
3. Liefere (lokales Optimum) y zurück.

Um für ein spezielles Problem einen auf lokaler Suche fußenden Algorithmus zu entwerfen, benötigen wir daher zweierlei:

a) ein Verfahren, um den in Schritt 1 erforderlichen Startwert (d.h., um irgendeine zulässige Lösung) zu finden sowie

b) einen geeigneten „Nachbarschaftsbegriff“ auf dem Raum der zulässigen Lösungen.

Weiterhin muss folgendes gewährleistet sein, damit „lokale Suche“ als Näherungsalgorithmusstrategie taugt:

(i) Schritt 1 (siehe a)) geht polynomiell.

(ii) Die gemäß b) definierte Nachbarschaft lässt sich in Polynomzeit auf bessere Lösungen hin untersuchen (Schritt 2).

(iii) Der Gesamtalgorithmus findet nach polynomiell vielen Schritten ein lokales Optimum.

6.1 Größtmöglicher (Kanten-)Schnitt (Maximum Cut MC)

I : Graph $G = (V, E)$
$S : \{\{V_1, V_2\} \mid V_1, V_2 \subseteq V, V_1 \cap V_2 = \emptyset, V_1 \cup V_2 = V\}$
$m : |C(V_2, V_2)|$, mit $C(V_1, V_2) = \{e \in E \mid e \cap V_1 \neq \emptyset \wedge e \cap V_2 \neq \emptyset\}$
opt : max
Wir wollen lokale Suche für MC anwenden.

zu a): Nimm $V_1 = \emptyset, V_2 = V$ (ist also trivial)

zu b) $\mathcal{N}(\{V_1, V_2\}) = \{\{U_1, U_2\} \in S \mid \exists v \in V : U_1 \triangle V_1 = U_2 \triangle V_2 = \{v\}\}$
(Hamming-Nachbarschaft)
$\triangle$ bezeichnet die symmetrische Differenz von Mengen, d.h.:
$A \triangle B = A \setminus B \cup B \setminus A$.

Satz 6.1 *Ist $G = (V, E)$ eine Instanz von MC und ist $\{V_1, V_2\}$ ein lokales Optimum bzgl. der obigen Nachbarschaft $\mathcal{N}(\{V_1, V_2\})$ mit dem Wert $m_{\mathcal{N}}(G)$, dann gilt:*

$$m^*(G)/m_{\mathcal{N}}(G) \leq 2.$$

Daher liefert die lokale Suche eine polynomielle 2-Approximation.

Beweis: Wir betrachten die Eingabe $G = (V, E)$. "Lokale Suche" ist sicher polynomial, da mit der Suche nur weitergemacht wird, wenn der Kantenschnitt vergrößert werden kann und daher höchstens $|E|$ viele lokale Suchen durchgeführt werden. Jeder lokale Suchschritt ist sicher auch in Polynomzeit durchzuführen.

Ist $m := |E|$, so gilt sicher $m^*(G) \leq m$. Wir zeigen jetzt:

$$m_{\mathcal{N}}(G) \geq \frac{m}{2}.$$

Ist $m_i := |E(G(V_i))|, i = 1, 2$, so gilt:

$$m = m_1 + m_2 + m_{\mathcal{N}}(G) \qquad (*)$$

Ist $V_{i,u} = \{v \in V_i \mid \{u, v\} \in E\}$, $i = 1, 2$, und ist $m_{i,u} := |V_{i,u}|$, so gilt wegen der lokalen Optimalität von $\{V_1, V_2\}$:

$$m_{1,u} \leq m_{2,u}, \quad \text{falls } u \in V_1 \text{ (sonst } \{V_1 \setminus \{u\}, V_2 \cup \{u\}\} \text{ „besser“)}$$

$$m_{1,u} \geq m_{2,u}, \quad \text{falls } u \in V_2 \text{ (sonst } \{V_1 \cup \{u\}, V_2 \setminus \{u\}\} \text{ „besser“)}$$

Daraus folgt:

$$\sum_{u \in V_1} (m_{1,u} - m_{2,u}) = 2[!] \cdot m_1 - m_{\mathcal{N}}(G) \leq 0$$

sowie

$$\sum_{u \in V_2} (m_{2,u} - m_{1,u}) = 2 \quad \cdot m_2 - m_{\mathcal{N}}(G) \leq 0;$$

zusammen

$$\rightsquigarrow m_1 + m_2 - m_{\mathcal{N}}(G) \leq 0$$

$$\overset{(*)}{\rightsquigarrow} m \leq 2m_{\mathcal{N}}(G) \qquad \square$$

Im Allgemeinen ist lokale Suche eine beliebte Heuristik, die Schwierigkeit besteht darin, Gütegarantien nachzuweisen. Auch wenn es solche Garantien nicht geben sollte, sind lokale Suchalgorithmen häufig in der Praxis wertvoll, z.B. bei TSP.

7 Lineares Programmieren

7.1 Allgemeines

Lineare Optimierungsaufgaben wurden erstmal sustematisch in den vierziger Jahren des 20. Jahrhunderts im Zusammenhang der „Mangelverwaltung“ der Nachkriegszeit untersucht. Aus dieser Zeit stammt auch der Terminus „Lineares Programmieren “ (LP); heute würden wir eher von „linearem Optimieren“ sprechen, denn mit Programmieren im Sinne von Informatik hat LP (zunächst) nichts zu tun.
Eine LP-Aufgabe ist gegeben durch n **reellwertige** Variablen, zusammen gefasst als Vektor $\vec{x} = (x_1, \ldots, x_n)$, eine Zielfunktion (d.h. eine Linearkombination aus den Variablen), die es zu minimieren oder zu maximieren gilt unter der Einschränkung, dass die Lösung m gegebenen linearen Ungleichungen genügt.
Wie wir noch zeigen werden, lässt sich jedes lineare Programm wie folgt formulieren:

$$\boxed{\begin{array}{l}\text{Ggb: } \vec{c} \in \mathbb{R}^n, \vec{b} \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n} \\ \text{minimiere } \vec{c}^T \cdot \vec{x} \\ \text{unter den Bedingungen } A\vec{x} \geq \vec{b}, \\ \qquad\qquad\qquad\qquad \vec{x} \geq \vec{0}.\end{array}} \quad (*)$$

Das immer noch beliebteste Verfahren zur Lösung von LPs stammt von Dantzig, das so genannte **Simplexverfahren**. Dieses Verfahren durchläuft die Ecken des durch $A\vec{x} \geq \vec{b}$ angegebenen Polyeders; nur dort können optimale Lösungen liegen. Dies führt allerdings dazu, dass es LP-Aufgaben gibt, bei denen das Simplexverfahren Exponentialzeit benötigt. Theoretisch (!) besser ist daher die von Khachian entwickelte Ellipsoidmethode, die stets in Polynomzeit arbeitet.
Wir werden beide Verfahren hier nicht weiter erläutern, sondern das Wissen um sie als Black Box im Folgenden beutzen. Es gibt auch spezielle Softwarepakete für LP, die man in den folgenden Programmen einsetzen könnte.

7.2 Äquivalente Formulierungen

Es gibt andere Formulierungen von LP-Aufgaben, die sich leicht in die Form $(*)$ bringen lassen.
Beispiele hierfür seien im Folgenden kurz erläutert:

$$\boxed{\begin{array}{l}\text{Ggb: } \vec{c} \in \mathbb{R}^n, \vec{b} \in \mathbb{R}^m, \vec{b'} \in \mathbb{R}^l, A \in \mathbb{R}^{m \times n}, A' \in \mathbb{R}^{l \times n}, r \leq n \\ \text{minimiere } \vec{c}^T \cdot \vec{x} \\ \text{unter den Bedingungen} \qquad A\vec{x} \geq \vec{b}, \\ \qquad\qquad\qquad\qquad\qquad\quad A'\vec{x} = \vec{b'} \\ \qquad\qquad\qquad\qquad\qquad x_1 \geq 0, \ldots, x_r \geq 0\end{array}}$$

Überführung in die Form $(*)$:

- Jede Gleichheitsbedingung $\vec{a}' \cdot \vec{x} = b'_i$ kann durch zwei lineare Ungleichungen ausgedrückt werden: $\vec{a}' \cdot \vec{x} \geq b'_i$ und $-\vec{a}' \cdot \vec{x} \geq -b'_i$.
- Jede Variable x_j ohne Vorzeichenbeschränkung wird durch zwei Variablen x_j^+, x_j^- ersetzt mit $x_j^+ \geq 0, x_j^- \geq 0$;
  jedes Vorkommen von x_j im ursprünglichen Linearen Programm wird durch $(x_j^+ - x_j^-)$ ersetzt.

Man kann die linearen Ungleichungsbedingungen in $(*)$ auch durch lineare Gleichungsbedingungen ersetzten. Dies führt auf das folgende Problem:

Ggb: $\vec{c} \in \mathbb{R}^n, \vec{b} \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$
minimiere $\vec{c}^T \cdot \vec{x}$
unter $A\vec{x} = \vec{b}$,
und $\vec{x} \geq \vec{0}$

Dazu ist jede Ungleichung $\vec{a} \cdot \vec{x} \geq b_i$ durch zwei Bedingungen $\vec{a} \cdot \vec{x} - s_i = b_i$ und $s_i \geq 0$ zu ersetzen; die neu eingeführte Variable s_i heißt auch **Schlupfvariable**. Durch Übergang von $\vec{c}$ auf $(-\vec{c})$, von $\vec{b}$ auf $(-\vec{b})$ und von A auf $(-A)$ lässt sich auch die folgende Maximierungsaufgabe leicht in die Form $(*)$ bringen:

Ggb: $\vec{c} \in \mathbb{R}^n, \vec{b} \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$	
maximiere $\vec{c}^T \cdot \vec{x}$	$\rightsquigarrow$ minimiere $(-\vec{c})^T \cdot \vec{x}$
unter $A \cdot \vec{x} \leq \vec{b}$,	unter $(-A) \cdot \vec{x} \geq -\vec{b}$,
$\vec{x} \geq \vec{0}$	$\vec{x} \geq \vec{0}$

7.3 Duale Programme

Zu jeder (zur Unterscheidung auch *primal* genannten) gegebenen LP-Minimierungsaufgabe lässt sich das so genannte duale Maximierungsprogramm wie folgt angeben:

Primal	Dual
Bedingung i: $\sum_{j=1}^n a_{ij} x_j \geq b_i$	Variable y_i: $y_i \geq 0$
Variable x_j: $x_j \geq 0$	Bedingung $\sum_{i=1}^m a_{ij} y_i \leq c_j$
minimiere $\vec{c}^T \vec{x}$	maximiere $\vec{b}^T \vec{y}$

Dabei gehen wir von einem primalen Programm der Form $(*)$ aus mit $A = (a_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n)$, $\vec{x} = (x_j \mid 1 \leq j \leq n)$, $\vec{b} = (b_j \mid 1 \leq j \leq m)$ und $\vec{c} = (c_i \mid 1 \leq i \leq n)$.

Der **Dualitätssatz** besagt, dass für (lösbare) Lineare Programme gilt:

$$\min \left\{ \vec{c}^T \cdot \vec{x} \mid \vec{x} \geq 0, A\vec{x} \geq \vec{b} \right\} = \max \left\{ \vec{b}^T \cdot \vec{y} \mid \vec{y} \geq 0, \vec{y}^T A \leq \vec{c} \right\}.$$

7.4 Ganzahliges Programmieren und Relaxation

Aus $(*)$ (oder einer der besprochenen Umformulierungen) entsteht ein ganzahlige lineares Programm, indem zusätzlich gefordert wird, dass zulässige Lösungen stets nur ganzzahlig sind oder gar nur bestimmte ganze Zahlen als Lösung akzeptiert werden. Während LP (wie gesagt) in Polynomzeit lösbar ist, ist ILP (integer LP) ein NP-hartes Problem. Es ist für uns jedoch von Interesse, da

- viele Optimierungsaufgaben sich als ILP formulieren lassen und
- die Vernachlässigung („Relaxation“) der Ganzzahligkeitsbedingungen zu einem LP führt, dessen Lösung eine Schranke für das ursprüngliche ILP bietet; die weiter gehende Hoffnung ist, dass man durch einfache Operationen wie Runden aus der LP-Lösung eine möglichst gute ILP-Lösung generieren kann.

Ein konkretes Beispiel: das **gewichtete Knotenüberdeckungsproblem** (vgl. Abschnitt 2.2) lässt sich wie folgt als ILP ausdrücken:

$$\begin{array}{ll} \text{minimiere} & \sum_{v_i \in V} c_i x_i \\ \text{unter} & x_i + x_j \geq 1 \text{ für } \{v_i, v_j\} \in E \\ & x_i \in \{0,1\} \text{ für } v_i \in V \end{array}$$

Ausgegangen wird dabei von einem Graphen $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$ mit Knotenkosten $c_i (c_i \geq 0)$ für v_i. Die Variable x_i „entspricht“ dem Knoten v_i und die Bedingung $x_i + x_j \geq 1$ „entspricht“ der Kante $\{v_i, v_j\}$.

Daher ist eine Knotenmenge $C \subseteq V$, gegeben durch $C = \{v_i \mid x_i = 1\}$, genau dann eine Knotenüberdeckung, wenn für alle Kanten $\{v_i, v_j\}$ die Ungleichung $x_i + x_j \geq 1$ gilt.
Eine mögliche Relaxation ist die folgende:

$$\begin{array}{ll} \text{minimiere} & \sum_{v_i \in V} c_i x_i \\ \text{unter} & x_i + x_j \geq 1 \text{ für } \{v_i, v_j\} \in E \\ \text{und} & x_i \geq 0,\ i = 1, \ldots, n \end{array}$$

Betrachte nun folgendes Programm:

RoundVC$(G = (V, E), \vec{c})$

1. Bestimme die ILP-Formulierung des VC-Problems.
2. Bestimme die zugehörige LP-Formulierung durch Relaxation.
3. Es sei $\vec{x}^*_G = (x^*_1, \ldots, x^*_n)$ eine optimale Lösung für die Relaxation.
4. Liefere $\{v_i \mid x^*_i \geq 0,5\}$ zurück.

Die Knotenüberdeckung wird also durch Runden der LP-Lösung gewonnen.

Satz 7.1 *RoundVC findet eine zulässige Lösung eines gewichteten Knotenüberdeckungsproblems vom Wert* $m_{LP}(G, \vec{c})$ *mit*

$$m_{LP}(G, \vec{c}) / m^*(G, \vec{c}) \leq 2.$$

Beweis: Die von RoundVC gelieferte Lösung $\{v_i \mid x_i^* \geq 0,5\}$ ist eine Knotenüberdeckung. Wäre nämlich $e = \{v_{i_1} v_{i_2}\}$ eine nicht abgedeckte Kante, so wäre $x_{i_1}^* + x_{i_2}^* < 1$, im Widerspruch zur Annahme, $\vec{x}_G^*$ sei Lösung der Relaxations-LP-Aufgabe.
Umgekehrt ist jede Lösung der ILP-Aufgabe auch eine Lösung der LP-Aufgabe, sodass insbesondere für die optimalen Lösungen gilt:

$$m^*(G, \vec{c}\,) \geq m_{LP}^*(G, \vec{c}\,) = \sum_{v_i \in V} c_i x_i^*.$$

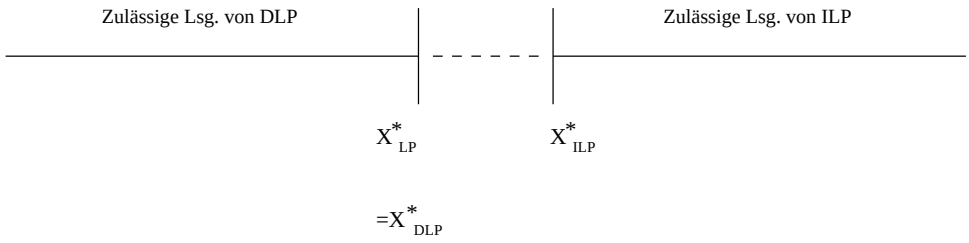
Weiter ist:

$$\begin{aligned} m_{LP}(G, \vec{c}\,) &= \sum_{x_i^* \geq 0,5 \; 1 \leq i \leq n} c_i \\ &\leq 2 \cdot \sum_{1 \leq i \leq n} c_i x_i^* \\ &= 2 \cdot \vec{c}^{\,T} \cdot \vec{x}_G^* \\ &= 2 \cdot m_{LP}^*(G, \vec{c}\,) \\ &\leq 2 \cdot m^*(G, \vec{c}\,) \qquad \square \end{aligned}$$

7.5 Primal-Duale Algorithmen für ILP

Nachteil von RoundVC ist, dass lineare Programme mit vielen Ungleichungsbedingungen zu lösen sind. Daher hat der entsprechende LP-Algorithmus eine (zu) große Laufzeit. Die Idee ist nun, (irgendwelche) zulässigen Lösungen für das zugehörige duale LP zu benutzen, um Schranken für das eigentlich zu lösende ILP zu gewinnen. Genauer sieht diese Überlegung wie folgt aus:

Betrachte ein (Minimierungs-)ILP, die zugehörige Relaxation LP sowie dessen (Maximierungs-)Dual-DLP mit optimalen Werten x_{ILP}^*, x_{LP}^* und x_{DLP}^*. Aus dem Dualitätssatz ergibt sich für eine Verteilung der Werte von Lösungen auf der Zahlengeraden:



Tatsächlich wird bei den Primal-Dual-Näherungsalgorithmen für ILPs vermieden, explizit die optimale Schranke $x_{LP}^* = x_{DLP}^*$ zu berechnen; es werden statt dessen sukzessive bessere zulässige Lösungen des DLPs berechnet.

Wir geben Konkret ein Beispiel:

PDVC $(G, \vec{c}\,)$

1. Bestimme die ILP-Formulierung des VC-Problems.
2. Bestimme die zugehörige LP-Formulierung durch Relaxation.
3. Bestimme die zugehörige DLP-Formulierung.

Bemerkung: DLP_{VC} lautet:

maximiere	$\sum_{\{v_i,v_j\}\in E} y_{\{i,j\}}$
unter	$\sum_{j:\{v_i,v_j\}\in E} y_{\{i,j\}} \leq c_i$ für jedes $v_i \in V$
	$y_{\{i,j\}} \geq 0$ für alle $\{v_i, v_j\} \in E$

4. Setze jede duale Variable $y_{\{i,j\}}$ auf Null.
   [Dies ist eine zulässige Lösung des DLP.]
5. $V' := \emptyset$; [V' enthält die Knoten der Überdeckungsapproximation.]
6. Solange V' keine Knotenüberdeckung ist, tue:

   6a. Wähle Kante $e = \{v_i, v_j\}$ mit $e \cap V' = \emptyset$.

   6b.
   $$\varepsilon := \min \left\{ c_i - \sum_{k:\{v_i,v_k\}\in E} y_{\{i,k\}}, c_j - \sum_{k:\{v_j,v_k\}\in E} y_{\{j,k\}} \right\}$$

   6c.
   $$y_{\{i,j\}} := y_{\{i,j\}} + \varepsilon$$

   6d. Wenn
   $$\sum_{k=\{v_i,v_k\}\in E} y_{i,k} = c_i$$

dann	$V' := V' \cup \{v_i\}$
sonst	$V' := V' \cup \{v_j\}$

7. Gib V' aus.

Satz 7.2 *PDVC findet eine zulässige Lösung des gewichteten Knotenüberdeckungsproblems vom Wert* $m_{PD}(G, \vec{c})$ *mit*

$$\frac{m_{PD}(G, \vec{c})}{m^*(G, \vec{c})} \leq 2.$$

Beweis: Die Schleife in Schritt 6 wird erst verlassen, wenn in V' eine zulässige Lösung „aufgesammelt“ wurde.
Schritt 6d gewährleistet für die von PDVC gelieferte Überdeckung V':

$$\forall v_i \in V' : \sum_{j:\{v_i,v_j\}\in E} y_{\{i,j\}} = c_i$$

Also ist:

$$\begin{aligned} m_{PD}(G,\vec{c}) &= \sum_{v_i \in V'} c_i \\ &= \sum_{v_i \in V'} \sum_{j:\{v_i,v_j\} \in E} y_{\{i,j\}} \\ &\leq \sum_{v_i \in V} \sum_{j:\{v_i,v_j\} \in E} y_{\{i,j\}} \\ &= 2 \cdot \sum_{\{v_i,v_j\} \in E} y_{\{i,j\}} \\ &\leq 2 \cdot m^*(G,\vec{c}), \end{aligned}$$

denn $\sum_{\{v_i,v_j\} \in E} y_{\{i,j\}}$ ist der Wert einer zulässigen Lösung des DLP. □

Bemerkung 7.3 *Der soeben vorgestellte Algorithmus PDVC ist tatsächlich der schon in Abschnitt 2.4 behandelte von Bar-Yehuda und Even. Um dies einzusehen, setze man anfangs in PDVC* $w(v_i) := c_i$ *als Gewichtsreduzierung und als Schritt „6e“ füge man* $w(v_i) := w(v_i) - \varepsilon$ *und* $w(v_j) := w(v_j) - \varepsilon$ *ein. Dann gilt offenbar stets:*

$$w(v_i) = c_i - \sum_{k:\{v_i,v_k\} \in E} y_{\{i,k\}}.$$

In termini des Kapitels 2 ist ε *daher der Nicht-Null-Wert der Gewichtsreduktionsfunktion* $\delta_{\{v_i,v_j\}}$.

Dieser Zusammenhang lässt sich noch allgemeiner darstellen, siehe auch:
R. Bar-Yehuda, D. Rawitz: On the equivalence between the primal-dual scheme and the local-ratio technique. Seiten 24–35 IN: M. Goemanns u.a.: Approximation, Randomization and Combinatorical Optimization, LNCS 2129, Springer, 2001.

8 Dynamisches Programmieren

Dynamisches Programmieren ist im Grunde genommen eine Technik, wie aus Optima, die für Teilprobleme des ursprünglichen ermittelt wurden, das Optimum für das Grundproblem berechnet werden kann. Dabei werden Mehrfachberechnungen durch Buchführung (in einer Tabelle) vermieden.[9] Die Tabelle wird dabei sukzessive von kleinen zu immer größeren Problemen hin aufgebaut. Wir werden in diesem Kapitel —allgemeiner gesprochen— sehen, wie man Lösungsstrategien, die stets das Optimum liefern, aber (vermutlich notgedrungen) eine superpolynomielle Laufzeit haben, dazu benutzen kann, Näherungsalgorithmen zu entwerfen.
Betrachten wir diese Vorgehensweise für das schon bekannte Rucksackproblem.

8.1 Ein exakter Algorithmus für das Rucksackproblem

Gegeben sind also: eine Rucksackkapazität $b \in \mathbb{N}$, Gegenstände $\{x_1, \ldots, x_n\}$ mit zugehöriger Größenangabe $a_i \in \mathbb{N}$ und Profitangabe $p_i \in \mathbb{N}$.

Wir wollen davon ausgehen,

- dass alle Gegenstände der Instanz einzeln in den leeren Rucksack passen; d.h. formal, dass $\forall i : a_i \leq b$, und
- dass *nicht* alle Gegenstände der Instanz zusammen in den leeren Rucksack passen; d.h. formal, dass $\sum_{i=1}^{n} a_i > b$; sonst wäre die optimale Lösung ja trivial zu bekommen.

Für das dynamische Programmieren betrachten wir für jedes $1 \leq k \leq n$ und jedes $0 \leq p \leq \sum_{i=1}^{n} p_i$ das Problem, eine Teilmenge von $\{x_1, \ldots, x_n\}$ zu finden mit Gesamtprofit p und Gesamtgröße höchstens b. Es bezeichnen $M^*(k,p) \subseteq \{x_1, \ldots, x_k\}$ eine optimale Lösung dieses Problems und $S^*(k,p)$ die entsprechende Gesamtgröße. Als Sonderfall setzen wir $S^*(k,p) := 1 + \sum_{i=1}^{n} a_i$, falls $M^*(k,p)$ undefiniert ist.

Damit gilt

$$M^*(1,p) = \begin{cases} \emptyset, & p = 0 \\ \{x_1\}, & p = p_1 \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

Für $k \geq 2$ gilt der folgende (rekursive) Zusammenhang:

$$M^*(k,p) = \begin{cases} M^*(k-1, p-p_k) \cup \{x_k\} & \text{falls gilt: (i) } p_k \leq p, \\ & \text{(ii) } M^*(k-1,p-p_k) \text{ ist definiert,} \\ & \text{(iii) } S^*(k-1,p) \\ & \qquad \geq S^*(k-1,p-p_k) + a_k \\ & \text{(iv) } S^*(k-1,p-p_k) + a_k \leq b \\ M^*(k-1,p) & \text{sonst} \end{cases}$$

In Worten heißt dies: Die beste Teilmenge von $\{x_1, \ldots x_k\}$ mit Profit p ist entweder die beste Teilmenge von $\{x_1, \ldots, x_{k-1}\}$ mit Profit p (d.h., x_k ist in jener Teilmenge

[9] bekanntes Beispiel aus Grundvorlesung: CYK-Algorithmus

nicht enthalten) oder es ist die beste Teilmenge von $\{x_1, \ldots, x_{k-1}\}$ mit Profit $p - p_k$ (plus Gegenstand x_k).

Dynamisches Programmieren bedeutet nun, die Werte von M^* und S^* für aufsteigende Argumente der Reihe nach zu berechnen.

Konkret sieht der Algorithmus wie folgt aus:

KnapsackDynProg $(X = \{x_1, \ldots, x_n\}, \{p_1, \ldots, p_n\}, \{a_1, \ldots, a_n\}, b)$

1. Für $p := 0$ bis $\sum_{i=1}^{n} p_i$ tue

   1a. Setze $M^*(1, p) :=$ undefiniert

   1b. Setze $S^*(1, p) := 1 + \sum_{i=1}^{n} a_i$

2a. $M^*(1, 0) := \emptyset; S^*(1, 0) := 0;$

2b. $M^*(1, p_1) := \{x_1\}; S^*(1, p_1) := a_1;$

3. Für $k := 2$ bis n tue:
   Für $p := 0$ bis $\sum_{i=1}^{n} p_i$ tue:

   3a. Falls $p_k \leq p$ und $M^*(k-1, p-p_k)$ ist definiert
   und $S^*(k-1, p-p_k) + a_k \leq \min\{b, S^*(k-1, p)\}$
   dann

   3a1. Setze $M^*(k, p) := M^*(k-1, p-p_k) \cup \{x_k\}$

   3a2. Setze $S^*(k, p) := S^*(k-1, p-p_k) + a_k$

   3b. sonst

   3b1. Setze $M^*(k, p) := M^*(k-1, p)$;

   3b2. Setze $S^*(k, p) := S^*(k-1, p)$

4. Sei p^* das maximale p, für das $M^*(n, p)$ definiert ist.
   (Beachte: $M^*(n, 0) = \emptyset$ ist stets definiert!)

5. Liefere $M^*(n, p^*)$.

Satz 8.1 *KnapsackDynProg findet stets eine optimale Lösung, und zwar in Zeit*

$$O(n \cdot \sum_{i=1}^{n} p_i).$$

Hinweis: Diese Laufzeit ist *nicht* polynomiell, da die p_i als binär codiert angenommen werden. Solche so genannten pseudopolynomialen Algorithmen untersuchen wir näher in Kap. 12.

8.2 Ein Näherungsverfahren für das Rucksackproblem

KnapsackFPTAS ($X = \{x_1, \ldots, x_n\}, \{p_1, \ldots, p_n\}, \{a_1, \ldots a_n\}, b, r$)
($r \in \mathbb{Q}, r > 1$ ist der angestrebte Approximations-Leistungsfaktor)

1. $p_{max} := \max\{p_i \mid 1 \leq i \leq n\}$;
2. $t := \left\lfloor \log_2(\frac{r-1}{r} \cdot \frac{p_{max}}{n}) \right\rfloor$;
3. $x' :=$ Instanz mit Profiten $p'_i := \lfloor p_i / 2^t \rfloor$;
4. Liefere KnapsackDynProg(x').

Satz 8.2 *Gegeben seien eine Instanz x von MaximumKnapsack (mit n Gegenständen) und $r > 1, r \in \mathbb{Q}$. KnapsackFPTAS(x) liefert eine Lösung in Zeit $O(r \cdot n^3/(r-1))$, deren Maß $m_{AS}(x,r)$ der Ungleichung $m^*(x)/m_{AS}(x,r) \leq r$ genügt.*

Beweis: Es bezeichne $Y(x,r)$ die von KnapsackFPTAS gelieferte Lösung und $Y^*(x)$ eine optimale Lösung von Instanz x (mit Maß $m^*(x)$).

Es gilt:

$$\begin{array}{lll}
m_{AS}(x,r) = \sum_{x_j \in Y(x,r)} p_j & \geq & \sum_{x_j \in Y(x,r)} 2^t \cdot \lfloor p_j/2^t \rfloor \\
\text{(nach Definition der } \lfloor\cdot\rfloor\text{-Funktion)} & & \\
& \geq & \sum_{x_j \in Y^*(x)} 2^t \cdot \lfloor p_j/2^t \rfloor \\
\text{(denn } \sum_{x_j \in Y(x,r)} \lfloor p_j/2^t \rfloor \text{ ist der} & & \\
\text{optimale Wert des „skalierten Problems“)} & & \\
& \geq & \sum_{x_j \in Y^*(x)} (p_j - 2^t) \\
\text{(nach Definition der } \lfloor\cdot\rfloor\text{-Funktion)} & & \\
& = & \left(\sum_{x_j \in Y^*(x)} p_j\right) - 2^t \cdot |Y^*(x)| \\
& = & m^*(x) - 2^t \cdot |Y^*(x)|
\end{array}$$

$$\left.\begin{array}{l} \rightsquigarrow m^*(x) - m_{AS}(x,r) \leq 2^t \cdot |Y^*(x)| \leq 2^t \cdot n \\ \text{Offenbar gilt ferner: } n \cdot p_{\max} \geq m^*(x) \geq p_{\max} \end{array}\right\} \rightsquigarrow$$

$$\rightsquigarrow \frac{m^*(x) - m_{AS}(x,r)}{m^*(x)} \leq \frac{n \cdot 2^t}{p_{\max}}$$

$$\begin{array}{rcl}
\Rightarrow m^*(x) & \leq & \frac{p_{\max}}{p_{\max} - n \cdot 2^t} m_{AS}(x,r) \\
& \leq & \frac{p_{\max}}{p_{\max} - n \cdot \frac{r-1}{r} \cdot \frac{p_{\max}}{n}} m_{AS}(x,r) \\
& = & r \cdot m_{AS}(x,r)
\end{array}$$

Laufzeitabschätzung:

$$\begin{array}{rcll}
O\left(n \cdot \sum_{i=1}^n p'_i\right) & \leq & O\left(n \cdot \sum_{i=1}^n \frac{p_i}{2^t}\right) & \\
& \leq & O\left(n^2 \frac{p_{\max}}{2^t}\right) & \\
& \leq & O\left(n^2 \frac{p_{\max}}{\frac{r-1}{r} \frac{p_{\max}}{n}}\right) & \\
& = & O\left(n^3 \cdot \frac{r}{r-1}\right) & \square
\end{array}$$

Bemerkung 8.3 *Näheres zu Approximationsschemata und vollen Approximationsschemata s. Kap. 12.*

9 Weitere Strategien - Was tun bei konkreten Problemen?

In den vorigen Kapiteln haben wir verschiedene Strategien kennen gelernt um Approximationsalgorithmen zu entwerfen. Leider ist diese Liste nicht vollständig, denn Algorithmenentwurf ist i.A. ein sehr kreativer Prozess. Dabei ist es manchmal hilfreich, sich an einen altbekannten Algrithmus zu erinnern.
Betrachten wir das folgende graphentheoretische Problem, welches wichtig ist um die Zuverlässigkeit von Netzmodellen zu erhöhen.

Billigste Erweiterung zu starkem Zusammenhang

I : $G = (V, A)$ gerichteter, vollständiger Graph mit Kantenkostenfunktion $c : A \to \mathbb{N}$, eine vorliegende Kantenmenge A_0
S : $\{A_1 \subseteq A \setminus A_0 \mid (V, A_0 \cup A_1)$ ist stark zusammenhängend$\}$
Ein gerichteter Graph heißt stark zusammenhängend, wenn es zwischen je zwei Knoten einen (gerichteten) Pfad gibt.
m : $c(A_1)$
opt : min.

Hinweis: Das Problem ist NP-vollständig.

Die Idee, die man hierfür haben könnte, ist folgende:

Bekanntermaßen lässt sich ein minimaler Spannbaum in Polynomzeit finden. Kann man nun erzwingen, dass (durch geeignete Modifikation der Kantengewichte) zwei Bäume T_1 und T_2 (aufgefasst als Kantenmenge) mit gleicher Wurzel r, aber unterschiedlicher Pfeilrichtung gefunden werden, so gibt es in dem Graphen, der nur aus den so gewichteten Baumkanten besteht, zwischen je zwei Knoten x und y jedenfalls einen Pfad von x nach y, der über r führt. Auf diese Weise ließe sich eine 2-Approximation gewinnen, denn eine optimale Erweiterung C^* müsste (in gewissem Sinne) auch die Bäume T_1 und T_2 enthalten.

Konkretisieren wir diese Idee:

STC$(G = (V, A), c, A_0)$

1. Definiere neue Abstandsfunktion d_1 wie folgt:

$$d_1(u, v) = \begin{cases} 0, & \text{falls } (v, u) \in A_0 \\ c((v, u)), & \text{falls } (v, u) \notin A_0 \end{cases}$$

2. Finde minimalen Spannbaum T_1 (aufgefasst als Kantenmenge).
Sei r die Wurzel von T_1.
Setze $A_1 = \{(u, v) \mid (v, u) \in T_1\}$.

3. Definiere Abstandsfunktion d_2 wie folgt:

$$d_2(u,v) = \begin{cases} 0, & (u,v) \in A_0 \cup A_1 \\ \infty, & v = r \\ c((u,v)), & \text{sonst} \end{cases}$$

4. Finde minimalen Spannbaum T_2 (mit Abstandsfunktion d_2).
5. Liefere $(A_1 \cup T_2) \setminus A_0$.

Satz 9.1 *STC ist eine 2-Approximation für das Problem, eine billigste Erweiterung zu starkem Zusammenhang zu finden.*

Beweis: Durch die Definition von d_2 wird erzwungen, dass r (ebenfalls) die Wurzel von T_2 ist, sodass tatsächlich eine Erweiterung geliefert wird, die starken Zusammenhang garantiert. In Schritt 2 wird eine Kantenmenge T_1 gefunden, sodass jeder Knoten mit der Wurzel r durch einen Pfad verbunden ist. In einer optimalen Erweiterung C^* gibt es natürlich auch einen solchen Pfad.
Da T_1 minimaler Spannbaum ist, gilt $c(T_1 \setminus A_0) = d_1(T_1) \leq d_1(C^*) = c(C^*)$. Ist nämlich T_1' ein minimaler Spannbaum in $G(A_0 \cup C^*)$ (bzgl. des Abstandmaßes d_1), so ist sicher $d_1(T_1) \leq d_1(T_1')$, da T_1 auch andere, evtl, billigere Kanten benutzen darf. Da T_1' eine Kantenteilmenge von C^* ist, gilt $d_1(T_1') \leq d_1(C^*)$.
Entsprechend sieht man: $c(T_2 \setminus (A_0 \cup T_1)) = d_2(T_2) \leq d_2(C^*) \leq c(C^*)$. □

Literatur:

- K. P. Eswaran and R. E. Tarjan: Augmentation problems. SIAM J. Computing 5(1976), 653–665.
- G. N. Frederickson and J. Jájá: Approximation algorithms for several graph augmentation problems. SIAM J. Computing 10(1981), 270–283.
- H. Nagamochi and T. Ibaraki: An approximation for finding the smallest 2-edge connected subgraph containing a specified spanning tree. In: T. Asano u.a. (Herausgeber), Proc. COCOON'99, LNCS Band 1627, pp. 221–230. Springer, 1999.

Hinweis: Erweiterungsprobleme sind wichtig, wenn es darum geht, (bestehende) (Rechner-)Netzwerke ausfallsicherer zu machen. Daher wird in den genannten Referenzen auch das Problem besprochen, einen gegebenenen zusammenhängenden ungerichteten Graphen k-fach mit möglichst geringem Aufwand (insbesondere zweifach) zusammenhängend zu machen, um mehrere unabhängige „Kommunikationswege“ zwischen „Rechenknoten“ zu gewährleisten.

Approximationsklassen

In diesem dritten Teil der Vorlesung wollen wir unser Augenmerk nicht mehr in erster Linie auf die unterschiedlichen algorithmischen Techniken zum Entwurf von Näherungsalgorithmen richten, sondern vielmehr Versuche unternehmen, Probleme bezüglich ihrer Approximierbarkeit zu klassifizieren.
Dabei werden wir insbesondere eine feinere Struktur innerhalb der Klasse NPO entdecken.

In gewisser Weise gehört dieser Vorlesungsteil also zum Gebiet der so genannten (strukturellen) Komplexitätstheorie. Daher wird in der Vorlesung mit dem Titel „Komplexitätstheorie“ auch auf diese Aspekte von Näherungsalgorithmen eingegangen. Wir werden aber auch in diesem Zusammenhang nicht nur die Phänomenologie der Komplexitätsklassen betrachten, sondern unser Augenmerk weiterhin auf das Finden von Näherungsalgorithmen richten.

10 Absolute Approximation

Wir hatten am Anfang bereits bemerkt, dass die Betrachtung der absoluten Näherungsgüte meist sinnlos ist. Wir werden dies in diesem kurzen Kapitel näher begründen.

Absoluter Fehler

Gegeben sei ein Optimierungsproblem $\mathcal{P}$. Dann ist für jede Instanz x von $\mathcal{P}$ und jede zulässige Lösung y von x der **absolute Fehler** von y bezüglich x definiert als

$$D(x, y) := |m^*(x) - m(x, y)|$$

Dabei ist wiederum $m^*(x)$ der Wert einer optimalen Lösung der Instanz x und $m(x, y)$ sei der Wert der Lösung y.
Ist $\mathcal{A}$ ein **Approximationsalgorithmus für** $\mathcal{P}$, d.h. gilt

$$\forall x \in I_{\mathcal{P}} : \mathcal{A}(x) \in S_{\mathcal{P}}(x),$$

so heißt $\mathcal{A}$ **absoluter Approximationsalgorithmus**, wenn es eine Konstante K gibt, so dass für alle $x \in I_{\mathcal{P}} : D(x, \mathcal{A}(x)) \leq K$ gilt.

Statt Approximationsalgorithmus sagen wir auch oft *Näherungsalgorithmus*.

Beispiele: Es gibt nur wenige Beispiele für NPO-Probleme mit absoluten Näherungsalgorithmen. Ein solches Beispiel liefert das Problem des Färbens von planaren Graphen.
Betrachten wir dazu folgendes Korollar aus dem Beweis von Satz 5.15:

Satz 10.1 *Betrachte die Knotenmenge des Eingabegraphen $G = (V, E)$ als geordnet: $(v_1, \ldots, v_n)$. (Dies ist auch die Reihenfolge, in der die Partitionsheuristik arbeitet.)*
Es sei $G_i = G(\{v_1, \ldots, v_i\})$; $d_i(v)$ bezeichne den Grad von v in G_i.
Die Partitionsheuristik benötigt dann höchstens

$$1 + \max_{1 \leq i \leq n} d_i(v_i)$$

viele Farben zum Färben von G_i. □

Wir wollen jetzt die Ordnung der Knoten genauer festlegen:
v_n sein ein Knoten minimalen Grades in $G = (V, E)$.
Induktiv gehen wir davon aus, die Reihenfolge $(v_{i+1}, \ldots, v_n)$ sei bereits festgelegt. Dann sei v_i ein Knoten minimalen Grades in $G(V \setminus \{v_{i+1}, \ldots, v_n\})$.

Satz 10.2 *Ist G ein planarer Graph, so benötigt die Partitionsheuristik mit der soeben definierten Knotenordnung höchstens 6 Farben zum Färben von G.*

Beweis: Wegen Satz 10.1 reicht es zu zeigen, dass

$$\max\{d_i(v_i) \mid 1 \leq i \leq n\} \leq 5.$$

Dazu ein kleiner **Exkurs**:

Wir erinnern wir an den
Eulerschen Polyedersatz: Für planare Graphen gilt:

$$f - m + n = 2.$$

Dabei ist:
$f = \#$ Flächen des (als eingebettet gedachten) Graphen,
$m = \#$ Kanten,
$n = \#$ Knoten.
Daraus folgt: $\boxed{m \leq 3n - 6}$.
Bezeichnet nämlich l_i die Anzahl der an Fläche F_i angrenzenden Kanten, so gilt, da $\forall i(l_i \geq 3)$:

$$2m = \sum_{i=1}^{f} l_i \geq 3f = 3m - 3n + 6.$$

Daher gilt:
$(*)$ Jeder planare Graph enthält wenigstens einen Knoten von Grad ≤ 5.
Sonst wäre $2m = \sum_{i=1}^{n} d(v_i) \geq 6n$, also $m \geq 3n$, im Widerspruch zum soeben Hergeleiteten.

Aus $(*)$ ergibt sich $\forall i : d_i(v_i) \leq 5$, da stets ein kleinstgradiger Knoten v_i in $G_i = G(\{v_1, \ldots, v_i\})$ ausgewählt wird und alle G_i planar sind, weil $G = G_n$ planar ist und die planaren Graphen unter der Operation „Fortlassen von Knoten und inzidenten Kanten“ abgeschlossen sind. □

Folgerung 10.3 *Das Satz 10.2 zu Grunde liegende Verfahren ist ein absoluter Approximationsalgorithmus zum Färben planarer Graphen. Die absolute Fehlerschranke von 5 lässt sich noch auf 3 reduzieren, da 2-färbbare Graphen mit einer Greedy-Strategie optimal gefärbt werden können.* □

Im Allgemeinen sind aber keine absoluten Näherungsgarantien zu erwarten. Schon das (approximationsalgorithmisch relativ einfache) Rucksackproblem gestattet (wohl höchstwahrscheinlich) kein polynomielles Näherungsverfahren mit konstantem absoluten Fehler. Genauer gilt:

Satz 10.4 *Unter der Annahme $P \neq NP$ gilt:*
Es gibt keinen polynomiellen absoluten Approximationsalgorithmus für Maximum-Knapsack.

Beweis: Es sei X eine Menge Gegenstände sowie zugehörigen Profiten p_i und Größen a_i und b die Rucksackkapazität. Gäbe es einen polynomiellen absoluten Approximationsalgorithmus für MaximumKnapsack mit Fehlerschranke K, so könnten wir das Rucksackproblem in Polynomzeit wie folgt lösen:

Wir generieren eine neue Instanz, indem wir alle Profite mit $(K+1)$ multiplizieren. Die Menge der zulässigen Lösungen wird dadurch nicht verändert. Der Wert jeder Lösung ist nun aber ein Vielfaches von $K+1$, was zur Folge hat, dass jede Lösung mit absolutem Fehler K bereits optimal ist. □

Ähnliche Argumente lasen sich für viele andere Probleme angeben.

11 Relative Approximation; die Klasse APX

11.1 Definitionen der Grundbegriffe

Interessanter Weise sind die in der Literatur (wenigstens) zwei verschiedene Maße gängig, um die relative Güte von Näherungsverfahren anzugeben.

Definition 11.1 *Ist $\mathcal{P}$ ein Optimierungsproblem, so ist —für jede Instanz x von $\mathcal{P}$ und für jede zulässige Lösung y von x— der **relative Fehler** von y bezüglich x definiert als*

$$E(x,y) := \frac{|m^*(x) - m(x,y)|}{\max\{m^*(x), m(x,y)\}} = \frac{D(x,y)}{\max\{m^*(x), m(x,y)\}}.$$

*Ist $\mathcal{A}$ ein Approximationsalgorithmus für $\mathcal{P}$, so heißt $\mathcal{A}$ ϵ**-Approximation**, wenn*

$$\forall x \in I_{\mathcal{P}} : E(x, \mathcal{A}(x)) \leq \epsilon.$$

Bemerkung 11.2 *Der relative Fehler ist gleich Null, wenn die Lösung optimal ist, und er liegt nahe bei Eins, wenn die Lösung schlecht ist.*

Definition 11.3 *Ist $\mathcal{P}$ ein Optimierungsproblem, so ist —für jede Instanz x von $\mathcal{P}$ und für jede zulässige Lösung y von x— die **Leistungsgüte** (engl: performance ratio) von y bezüglich x definiert als*

$$R(x,y) := \max\left\{\frac{m^*(x)}{m(x,y)}, \frac{m(x,y)}{m^*(x)}\right\}.$$

*Ist $\mathcal{A}$ ein Approximationsalgorithmus für $\mathcal{P}$, so heißt $\mathcal{A}$ r**-Approximation**, wenn*

$$\forall x \in I_{\mathcal{P}} : R(x, \mathcal{A}(x)) \leq r.$$

Bemerkung 11.4 *Die Leistungsgüte ist gleich Eins, wenn die Lösung optimal ist, und sie ist sehr groß, wenn die Lösung schlecht ist. Offenbar gilt:*

$$E(x,y) = 1 - \frac{1}{R(x,y)}.$$

Ist nämlich $\mathcal{P}$ ein Maximierungsproblem, so ist:

$$1 - \frac{1}{R(x,y)} = 1 - \frac{1}{\frac{m^*(x)}{m(x,y)}} = 1 - \frac{m(x,y)}{m^*(x)} = \frac{m^*(x) - m(x,y)}{m^*(x)}.$$

Ist $\mathcal{P}$ ein Minimierungsproblem, so gilt:

$$1 - \frac{1}{R(x,y)} = 1 - \frac{m^*(x)}{m(x,y)} = \frac{m(x,y) - m^*(x)}{m(x,y)}.$$

Hinweis: entspricht r-Approximation aus Kapitel 2 !

Beispiel 11.5 *Die (diversen) Näherungsalgorithmen für das Knotenüberdeckungsproblem sind sowohl $\frac{1}{2}$-Approximation (im Sinne des relativen Fehlers) als auch 2-Approximation (im Sinne der Leistungsgüte).*

Definition 11.6 *$\mathcal{P}$ heißt **ϵ-approximatierbar** (bzw. **r-approximierbar**), wenn es eine Polynomzeit-ϵ-Approximation (bzw. r-Approximation) für $\mathcal{P}$ gibt.*

Bemerkung 11.7 *Algorithmen $\mathcal{A}$, die (im Minimierungsfall)*

$$m_{\mathcal{A}}(x, y) \leq r \cdot m^*(x) + k$$

*erfüllen, sind „nur“ $(r + k)$-Approximationen (im Sinne von Definition 11.3), werden aber oft als r-Approximationen in der Literatur angesprochen. Wir werden solche Algorithmen **asymptotische r-Approximationen** nennen. Im Zusammenhang mit Approximationsschemata ist dieser Unterschied bedeutender, siehe Kapitel 12 und 14.*

Definition 11.8 *APX ist die Klasse aller NPO-Probleme, die r-approximierbar für ein $r \geq 1$ sind.*

In den vorigen Abschnitten haben wir bereits einige Probleme aus APX kennen gelernt. Wir wollen an dieser Stelle noch ein weiteres Problem erörtern.

11.2 MAXSAT

I : Menge C von Klauseln über einer Menge V von Variablen
(Eine Klausel ist eine Disjunktion von Literalen.
Ein Literal ist eine Variable oder ihre Negation.
Formal ist C als Menge von Mengen von Literalen aufzufassen.)
S : Menge der Wertzuweisungen $f : V \to \{true, false\}$
m : $|\{c \in C \mid f(c) = true\}|$
opt : max

Beispiel: Die KNF-Formel $(x \vee \bar{y}) \wedge (\bar{x} \vee y)$ würden wir als $C = \{\{x, \bar{y}\}, \{\bar{x}, y\}\}$ schreiben.

Wir analysieren den folgenden Greedy-Algorithmus:

GreedySAT(C, V)

1. Für alle $v \in V$ setze $f(v) := true$ (Anfangsbelegung);
2. Wiederhole, solange $C \neq \emptyset$:

2a. Es sei l ein Literal, das in einer maximalen Zahl von Klauseln vorkommt; es sei x die zugehörige Variable.

2b. Bestimme die Menge C_x von Klauseln, in denen x vorkommt.

2c. Bestimme die Menge $C_{\overline{x}}$ von Klauseln, in denen $\overline{x}$ vorkommt.

2d. Ist $l = x$, so

2d(i) $C := C \setminus C_x$;
2d(ii) Lösche $\overline{x}$ aus allen Klauseln in $C_{\overline{x}} \setminus C_x$
2d(iii) Lösche alle leeren Klauseln aus C.

2e. sonst:

2e(0) $f(x) := \mathit{false}$;
2e(i) $C := C \setminus C_{\overline{x}}$;
2e(ii) Lösche x aus allen Klauseln $C_x \setminus C_{\overline{x}}$;
2e(iii) Lösche alle leeren Klauseln aus C.

3. Liefere f zurück.

Satz 11.9 *GreedySAT ist eine Polynomzeit-2-Approximation für MAXSAT.*

Beweis: Per Induktion über die Anzahl der Variablen zeigen wir:
Ist eine Instanz mit c Klauseln gegeben, so liefert GreedySAT stets eine Lösung, die mindestens $c/2$ der Klauseln erfüllt. Da trivialerweise c eine obere Schranke für den Wert einer optimalen Lösung ist, folgt so die Behauptung.
Im Fall, dass es nur eine Variable gibt, folgt die Behauptung trivialerweise. Es werde nun angenommen, die Behauptung sei für $n-1$ Variablen ($n > 1$) bewiesen. Von GreedySAT werde als erstes die Variable x betrachtet. Es seien c_x bzw. $c_{\overline{x}}$ die Anzahl der Klauseln, in denen x bzw. $\overline{x}$ vorkommen. Ohne Einschränkung betrachten wir $c_x \geq c_{\overline{x}}$, d.h., GreedySAT wird x auf *true* setzen. Nach dieser Zuweisung müssen (nach Schritt 2d(i) bzw. 2d(iii)) noch $\tilde{c} \geq c - c_x - c_{\overline{x}}$ viele Klauseln (mit $n-1$ Variablen) betrachtet wurden. Nach Induktionsvoraussetzung werden daher von GreedySAT wenigstens $\tilde{c}/2 \geq (c - c_x - c_{\overline{x}})/2$ der $\tilde{c}$ vielen Klauseln erfüllt. Insgesamt werden daher wenigstens $c_x + (c - c_x - c_{\overline{x}})/2 = \frac{c+c_x-c_{\overline{x}}}{2} \geq \frac{c}{2}$ Klauseln der ursprünglichen Klauselmenge erfüllt. □

Andererseits gibt es auch Probleme, die höchstwahrscheinlich nicht zu APX gehören, wie der folgende Satz zeigt.

11.3 Das Handelsreisendenproblem MTS

Satz 11.10 *Wenn das allgemeine Handelsreisendenproblem MTS zu APX gehörte, dann wäre $P = NP$.*

Beweis: Wir beweisen das Resultat duch Reduktion vom Hamiltonschen Kreisproblem. Ein Hamiltonscher Kreis ist dabei eine Permutation $(v_1, \ldots v_n)$ der Knoten eines ungerichteten Graphen $G = (V, E), |V| = n$, sodass $\forall 1 \leq i \leq n \{v_i, v_{i+1}\} \in E$

(der Einfachheit halber sei $v_{n+1} = v_1$). Für jedes $r \geq 1$ konstruieren wir eine Instanz von MTS derart, dass aus der Existenz einer r-Approximation für MTS folgen würde, dass es einen Polynomzeitalgorithmus zum Hamiltonschen Kreisproblem gäbe. Da das Hamiltonsche Kreisproblem NP-vollständig ist, folgt die Behauptung. Geben wir uns also ein $r \geq 1$ vor. Ist $G = (V, E)$ eine Hamiltonsche Kreisproblem-instanz, so definieren wir (auf derselben Knotenmenge V) eine Abstandsfunktion d, die zusammen mit V das fragliche MTS-Problem angibt. Setze dazu:

$$d(u, v) := \begin{cases} 1, & \{u, v\} \in E \\ 1 + |V| \cdot r, & \{u, v\} \notin E \end{cases}$$

MTS hat eine (minimale) Lösung vom Wert $|V|$ gdw. G einen Hamiltonschen Kreis enthält.
Eine r-Approximation $\mathcal{A}$ für MTS könnte wie folgt zum Lösen des Hamiltonschen Kreisproblems verwendet werden:

- Liefert $\mathcal{A}$ einen Wert $\leq r \cdot |V|$ zurück, so hatte das ursprüngliche Kreisproblem eine Lösung.
- Liefert $\mathcal{A}$ einen Wert $> r \cdot |V|$ zurück, so war das ursprüngliche Kreisproblem unlösbar. □

Korollar 11.11 *Ist $P \neq NP$, so ist $APX \neq NPO$.* □

Interessanterweise verringert sich (approximationstheoretisch betrachtet) die Komplexität von MTS, wenn man sich auf metrische Instanzen einschränkt. Cristofides ersann einen Algorithmus für MMTS (metrisches MTS), der eine Leistungsgüte von 3/2 gewährleistet. Diesen wollen wir nun kennen lernen (und dabei den Greedy-Algorithmus aus Kapitel 4 essentiell verbessern).
Dazu benötigen wir ein paar weitere graphentheoretische Begriffe und graphenalgorithmische Ergebnisse.

Ein **Multigraph** ist ein Paar $M = (V, E)$, wobei V eine endliche Knotenmenge ist und F eine Multimenge von Kanten. Eine Multimenge von Kanten (über V) ist dabei (vereinfacht) eine Abbildung $F : \{\{u, v\} \mid u, v \in V\} \to \mathbb{N}_0$; sogenannte „Mehrfachkanten“ sind also zugelassen. Wir werden im Folgenden auch Multigraphen mit **Kantengewichten** betrachten; dabei wird jeder einzelnen Kante $e = (\{u, v\}, j)$, $j \leq F(\{u, v\})$ ein Gewicht $c(e)$ zugeordnet. Man sagt, ein Graph $G = (V, E)$ mit Kantengewichten erfülle die $\triangle$-*Ungleichung*, wenn $\forall u, x, v \in V : c(\{u, v\}) \leq c(\{u, x\}) + c(\{x, v\})$, falls $\{u, v\}, \{u, x\}, \{x, v\} \in E$. Eine Folge von Knoten $f \in V^*$ heißt **Eulersch**, wenn jedes $v \in V$ mindestens einmal in f vorkommt, der erste Knoten von f gleich dem letzten ist und für alle Zerlegungen $f = xuvy, u, v \in V$ gilt: $F(\{u, v\}) \geq 1$, sowie

$$\forall u, v \in V, u \neq v : F(\{u, v\}) = |\{x \in V^* \mid \exists y \in V^* : f = xuvy \vee f = xvuy\}|.$$

Mit anderen Worten: jede Kante des Multigraphen wird genau einmal durchlaufen. Ein Multigraph heißt Eulersch, wenn es eine Eulersche Knotenfolge gibt.

Mitteilung 11.12 *Ein Multigraph ist genau dann Eulersch, falls alle seine Knoten (zusammenhängen und) geraden Grad besitzen.*

Es gibt einen Polynomzeitalgorithmus, der entscheidet, ob ein gegebener Multigraph Eulersch ist und dabei auch eine zugehörige Knotenfolge konstruiert.

Lemma 11.13 *Ist $G = (V, E)$ ein vollständiger Graph mit Kantengewichten mit Gewichtsfunktion c, der die $\triangle$-Ungleichung erfüllt und ist $M = (V, F)$ irgendein Eulerscher Multigraph mit der selben Knotenmenge derart, dass alle Kanten zwischen zwei beliebigen Knoten $\{u, v\} \subseteq V$ das Gewicht $c(\{u, v\})$ haben, dann findet man in Polynomzeit eine Tour I in G mit einem Wert, der höchstens gleich der Summe der Gewichte der Kanten in M ist.*

Beweis: Es sei $V = \{v_1, \ldots, v_n\}$. Per Definition ist die Summe der Gewichte aller Kanten, die in einer Eulerschen Knotenfolge „vorkommen“, gleich der Summe der Gewichte der Kanten in M.
Wir bestimmen als eine Eulersche Knotenfolge $f = v_{i_1} \ldots v_{i_m}$ in M. Es sei $j(r)$ der erste Index in $1, \ldots, m$ mit $v_{i_{j(r)}} = v_r, 1 \leq r \leq n$. Wir gehen davon aus, dass die $v_1, \ldots, v_n$ so sortiert sind, dass

$$j(1) \leq j(2) \leq \cdots \leq j(n)$$

gilt. Wegen der $\triangle$-Ungleichung gilt:

$$\forall 1 \leq r \leq n : c(\{v_r, v_{(r+1 \bmod m)+1}\}) = c\left(\left\{v_{i_{j(r)}}, v_{i_{(j(r+1) \bmod m)+1}}\right\}\right)$$

$$\leq c\left(\left\{v_{i_{j(r)}}, v_{i_{j(r) \bmod m+1}}\right\}\right) + \left(\left\{v_{i_{j(r) \bmod m+1}}, v_{i_{j(r) \bmod m+2}}\right\}\right) + \ldots$$

$$\ldots + \left(\left\{v_{i_{j(r+1) \bmod m}}, v_{i_{j(r+1) \bmod m+1}}\right\}\right)$$

Für die Tour $v_{i_{j(1)}}, \ldots, v_{i_{j(n)}}$ gilt daher die Behauptung. □

Ein **Matching** in einem Graphen ist eine Kantenmenge, in der keine zwei Kanten einen gemeinsamen Endpunkt haben.
In der Standard-Algorithmenvorlesung wird ein Polynomzeitalgorithmus zum Auffinden größtmöglicher (Maximum) Matchings vorgestellt. Ein Matching heißt **perfekt**, wenn jeder Knoten des Graphen Endpunkt einer Kante aus dem Matching ist.
Es gibt einen Polynomzeitalgorithmus zum Auffinden minimaler perfekter Matchings, wobei „Minimalität“ im Sinne der jetzt angenommenen Kantengewichte des Graphen zu sehen ist.

Wir stellen jetzt den Algorithmus von Christofides vor.

MMTS-CHR($G = (V, E), c$) (G vollständiger Graph, c Kantengewichtsfunktion)

1. Finde einen (bzgl. c) minimalen Spannbaum $T = (V, E_T)$ von G.
2. Es sei $U \subseteq V$ die Menge der Knoten in T mit ungeradem Grad.
3. Finde ein (bzgl. c) minimales perfektes Matching H in $G[U]$.

4. Bilde den Multigraphen $M = (V, E_T + H)$.
   (In M gibt es nur Einfach- oder Doppelkanten.)
5. Finde eine Eulersche Knotenfolge f in M.
6. Bilde (wie in obigem Lemma) eine Tour I aus f.
7. Liefere I zurück.

Satz 11.14 *Ist $(G = (V, E), c)$ eine Instanz von MMTS, so liefert MMTS-CHR in Polynomzeit eine Lösung mit Leistungsgüte $\leq \frac{3}{2}$.*

Beweis: Mit obigen Vorbemerkungen und Grundkenntnissen aus einer Algorithmenvorlesung ist klar, dass MMTS-CHR in Polynomzeit arbeitet. Wir zeigen jetzt die Leistungsgüteschranke:
Dazu betrachten wir den Multigraph $M = (V, E_T + H)$ und eine Eulersche Knotenfolge f. (Beachte: M ist Eulersch wegen obiger Mitteilung!). Nach dem vorigen Lemma können wir eine Tour I finden, die

$$m(G, I) \leq c(E_T) + c(H)$$

genügt. Im Folgenden schätzen wir $c(E_T)$ und $c(H)$ nach oben mit Hilfe von $m^*(G)$ ab.

Behauptung 1:

$$c(H) \leq \frac{m^*(G)}{2}.$$

Beweis: Es sei $(v_{i_1}, \ldots, v_{i_{|U|}})$ die Folge der Knoten aus U in der Reihenfolge, in der sie in einer fixierten optimalen Tour I^* vorkommen.
Offensichtlich sind

$$H_1 = \left\{(v_{i_1}, v_{i_2}), \ldots, (v_{i_{|U|-1}}, v_{i_{|U|}})\right\} \text{ und}$$

$$H_1 = \left\{(v_{i_2}, v_{i_3}), \ldots, (v_{i_{|U|-2}}, v_{i_{|U|-1}}), (v_{i_{|U|}}, v_{i_1})\right\}$$

perfekte Matchings. Wegen der Gültigkeit der $\triangle$-Ungleichung gilt:

$$c(H_1) + c(H_2) \leq m^*(G) = c(I^*).$$

Da H ein perfektes Matching mit minimalem Gewicht ist, gilt $c(H) \leq c(H_1)$ und $c(H) \leq c(H_2)$, also gilt $2c(H) \leq m^*(G)$.

Behauptung 2:

$$c(E_T) \leq m^*(G).$$

Hierzu beobachte man: Eine (optimale Tour) ist ein Spannbaum (ein „Spannpfad“) plus einer zusätzlichen Kante. Da T minimaler Spannbaum ist, gilt die Behauptung.
Zusammengenommen haben wir gezeigt:

$$m(G, I) \leq m^*(G) + \frac{m^*(G)}{2} = \frac{3}{2} m^*(G).$$

□

Wir wollen den Algorithmus an einem **praktischen Beispiel** verdeutlichen:

Gegeben sei folgende Straßen-km-Entfernungstabelle zwischen den Städten Amsterdam, Berlin, Genf, Mailand, Prag, Rom, Warschau und Wien:

AMS	685	925	1180	960	1755	1235	1180
	BER	1160	1105	340	1530	585	630
		GEN	325	950	880	1575	1025
			MAI	870	575	1495	830
				PRA	1290	625	290
					ROM	1915	1130
						WAR	765
							WIE

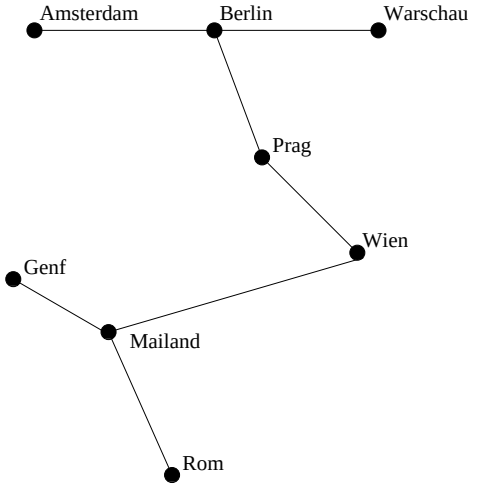


Abbildung 1: Ein minimaler Spannbaum

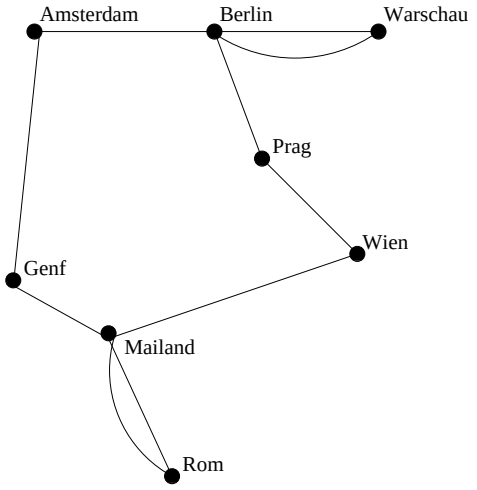


Abbildung 2: Der so erzeugte Multigraph

Die Bilder 1, 2 und 3 zeigen die drei Hauptphasen des Algorithmus: Schritt 1, 4 und 6. Als Eulersche Knotenfolge nehmen wir dabei

AMS-BER-WAR-BER-PRA-WIE-MAI-ROM-MAI-GEN-AMS

Eine optimale Tour finden Sie in Bild 4. Diese ist nicht durch bloßes zyklisches Verschieben der Eulerschen Knotenfolge zu erzielen, da die Kante Wien-Rom bereits im Spannbaum fehlt.

Schließlich wollen wir die **asymptotische Schärfe** der für den Algorithmus von Christofides gefundene Schranke nachweisen.

Dazu betrachte man die in Abbildung 5 angegebene Instanzenschar G_n von MMTS. Neben den angegebenen Bemeßungen beachte man, dass der Winkel zwischen $\overline{a_1 a_{n+1}}$ sowie $\overline{a_1 b_1}$ (und ebenso der zwischen $\overline{a_1 b_1}$ und $\overline{b_1 a_2}$, usw.) $60°$ beträgt; ansonsten verstehe man das Beispiel als ein in der Euklidischen Ebene befindliches. Ein möglicher minimaler Spannbaum besteht aus der in Abbildung 5 angegebenen Tour (ohne die Kante $\{a_1, a_{n+1}\}$).

Ausgehend von diesem Spannbaum bestimmt MMTS-CHR die in (3) angegebene Tour vom Wert $3n + 2\epsilon$.
Der Wert der in (4) angegebenen (optimalen) Tour beträgt aber lediglich $2n+1+4\epsilon$.

Bemerkung 11.15 *Bis heute wurde kein besserer Approximationsalgorithmus („besser“ nur im Sinne der Approximationsgüte) gefunden als MMTS-CHR für den metrischen Fall. Beschränkt man sich jedoch weiter auf den Euklidischen Fall, so gibt es sogar ein PTAS (→ nächstes Kapitel).*

11.4 Grenzen der Approximierbarkeit — Die Gap-Technik

Die in Satz 11.10 kennen gelernte Beweistechnik lässt sich wie folgt allgemeiner darstellen.

Satz 11.16 *Es seien $\mathcal{P}'$ ein NP-vollständiges Entscheidungsproblem und $\mathcal{P}$ ein Minimierungsproblem aus NPO. Angenommen, es gibt zwei in Polynomzeit berechenbare Funktionen*

$$f : I_{\mathcal{P}'} \to I_{\mathcal{P}} \text{ und}$$

$$c : I_{\mathcal{P}'} \to \mathbb{N} \text{ sowie}$$

eine Konstante $gap > 0$, sodass für jede Instanz x von $\mathcal{P}'$ gilt:

$$m^*(f(x)) = \begin{cases} c(x), & x \text{ ist positive Instanz von } \mathcal{P}' \\ c(x)(1+gap), & \text{sonst.} \end{cases}$$

Dann gibt es keine Polynomzeit-r-Approximation für $\mathcal{P}$ mit $r < 1 + gap$, falls $P \neq NP$ gilt.

Beweis: Nehmen wir an, es gäbe solch eine r-Approximation $\mathcal{A}, r < 1 + gap$, für $\mathcal{P}$. Wir können den Polynomzeitalgorithmus $\mathcal{A}$ wie folgt dazu benutzen, um $\mathcal{P}'$ in Polynomzeit zu lösen:

$\mathcal{A}'(x)$:[10]

1. Berechne $f(x)$.
2. Berechne $y := \mathcal{A}(f(x))$.
3. Berechne $c := c(x)$.
4. Gilt $m(f(x), y) < c \cdot (1 + gap)$, so antworte JA!
5. Gilt $m(f(x), y) \geq c \cdot (1 + gap)$, so antworte NEIN!

Warum ist $\mathcal{A}'$ korrekt?

Ist x eine positive Instanz von $\mathcal{P}'$, so gilt nach Voraussetzung

$$\frac{m(f(x), y)}{c} = \frac{m(f(x), y)}{m^*(f(x))} \leq r < 1 + gap$$

$\mathcal{A}'$ antwortet in diesem Fall also richtig (Schritt 4.).

Ist x eine negative Instanz von $\mathcal{P}'$, so gilt nach Voraussetzung

$$m(f(x), y) \geq m^*(f(x)) = c \cdot (1 + gap),$$

sodass $\mathcal{A}'$ auch in diesem Fall korrekt antwortet (Schritt 5.). □

11.5 Anwendung auf das Graphenfärbeproblem

Mitteilung 11.17 *Für **planare** Graphen (die nach dem Vierfarben-Satz ja 4-färbbar sind) ist die Frage nach der 3-Färbbarkeit NP-vollständig.*

Daher gilt: Das (allgemeine sowie das auf planare Graphen eingeschränkte) Graphenfärbeproblem lässt sich nicht besser als mit Faktor $r < 4/3$ approximieren. ($gap = 1/3$)

Man kann hier sogar noch mehr zeigen:

Mitteilung 11.18 *Gilt $P \neq NP$, so liegt MinimumGraphColoring in $NPO \setminus APX$.*

[10] x ist eine Instanz von $\mathcal{P}'$.

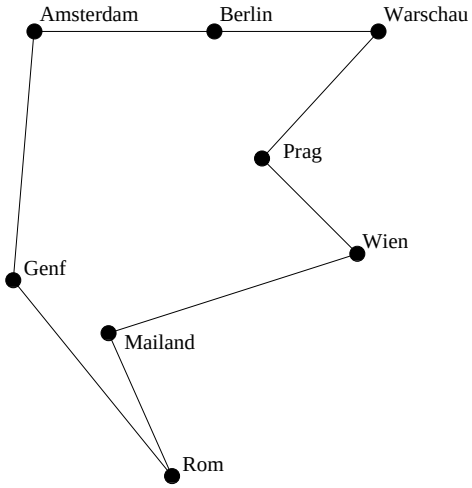


Abbildung 3: Die von Christofides erhaltene Tour I (vom Wert 5395)

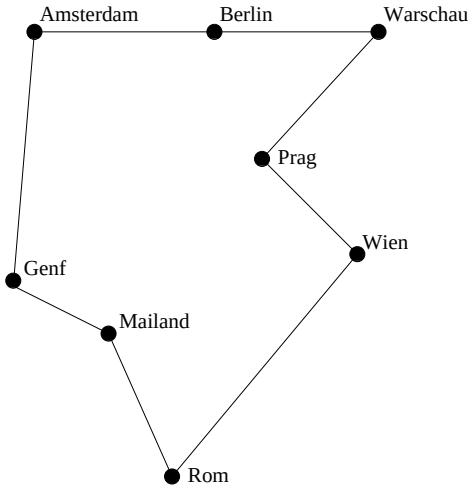


Abbildung 4: Ein optimale Tour I^* (vom Wert 5140)

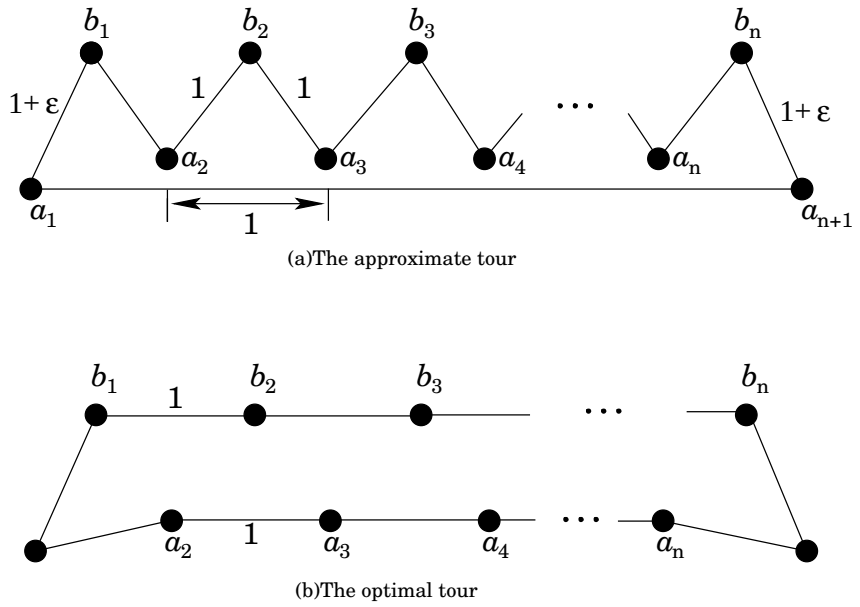


Abbildung 5: Zur Schärfe von Christofidis

12 Polynomzeit-Approximationsschemata

Wir hatten im vorigen Kapitel Beispiele für Probleme kennen gelernt, die sich nicht mit konstantem Faktor approximieren lassen (sofern nicht P = NP) und für Probleme, die sich *nicht* mit beliebigem Faktor approximieren lassen (wie Graphenfärben auf planaren Graphen). Jetzt wollen wir uns Probleme ansehen, die beliebig genaue Approximationsverfahren gestatten, wobei eine erhöhte Genauigkeit der Näherung jedoch durch eine erhöhte Laufzeit eingehandelt wird.

Definition 12.1 *Es sei $\mathcal{P} \in$ NPO. Ein Algorithmus $\mathcal{A}$ heißt* ***Polynomzeit-Approximationsschema*** *(engl: polynomial-time approximation scheme, kurz PTAS), falls $\mathcal{A}$ für jedes Paar von Eingaben (x, r), x Instanz von $\mathcal{P}$ und $r > 1$, eine r-approximative Lösung zurück liefert in einer Zeit, die polynomiell in $|x|$ ist (aber möglicherweise nicht-polynomiell in $\frac{1}{r-1}$).*
Die zugehörige Problemklasse nennen wir PTAS.

12.1 Kleinstmögliche Zweiteilung (Minimum Partition, MP)

Problembeschreibung:

I : Endliche Menge X von Gegenständen, zu jedem $x_i \in X$ ein Gewicht $a_i \in \mathbb{N}$.
S : $\{Y_1, Y_2 \subseteq X \mid Y_1 \cap Y_2 = \emptyset, Y_1 \cup Y_2 = X\}$
m : $\max\left\{\sum_{x_i \in Y_1} a_i, \sum_{x_i \in Y_2} a_i\right\}$
opt : min.

Es ist klar, dass MP in APX liegt, denn die Zweiteilung $(X, \emptyset)$ ist trivialerweise eine 2-Approximation.

Wir zeigen jetzt:

Satz 12.2 *MP $\in$ PTAS.*

Beweis: Betrachte dazu das folgende Programm:

MP-PTAS(X, a, r)

1. Ist $r \geq 2$, so liefere $(X, \emptyset)$.
2. Sortiere X absteigend bzgl. ihrer Gewichte.
   Es sei $(x_1, \ldots, x_n)$ die so erhaltene Folge.
3. $k(r) := \left\lceil \frac{2-r}{r-1} \right\rceil$;
4. Finde eine optimale Zweiteilung (Y_1, Y_2) für $(x_1, \ldots, x_{k(r)})$.
5. Für $j := k(r) + 1$ bis n tue

5a. Falls

$$\sum_{x_i \in Y_1} a_i \leq \sum_{x_i \in Y_2} a_i,$$

5b. so

$$Y_1 := Y_1 \cup \{x_j\},$$

5c. sonst

$$Y_2 := Y_2 \cup \{x_j\}.$$

6. Liefere (Y_1, Y_2).

Klar ist: Die Laufzeit von MP-PTAS ist für festes r polynomiell in n; allerdings führt Schritt 4 zu exponentieller Laufzeit in $k(r)$ mit $k(r) \in O\left(\frac{1}{r-1}\right)$.

Wir zeigen nun, dass die Leistungsgüte, gegeben $x = (X, a)$ und $r > 1$, durch r beschränkt ist. Klar ist dies, falls $r \geq 2$. Sei nun also $r \in (1, 2)$. Setze $a(Y) := \sum_{x_j \in Y} a_j, Y \subseteq X$ und $L := \frac{a(X)}{2}$. O. E. sei $a(Y_1) \geq a(Y_2)$, und x_h das letzte Element, das zu Y_1 hinzugefügt wurde durch MP-PTAS.

Also ist $a(Y_1) - a_h \leq a(Y_2)$

$$\rightsquigarrow 2 \cdot a(Y_1) - a_h \leq a(Y_1) + a(Y_2) = 2L$$

$$\rightsquigarrow \quad a(Y_1) - L \leq \frac{a_h}{2} \qquad (*)$$

Wurde x_h bereits in Schritt 4 eingefügt, so liefert MP-PTAS sogar eine optimale Lösung.
Wurde x_h in Schritt 5 eingefügt, so haben wir (natürlich) $a_h \leq a_j$ für $1 \leq j \leq k(r)$ (absteigende Sortierung!) und

$$2L \geq \sum_{1 \leq j \leq h} a_j \geq h \cdot a_h \geq (k(r) + 1) \cdot a_h. \qquad [+]$$

Wegen $a(Y_1) \geq L \geq a(Y_2)$ und $m^*(x) \geq L$ gilt ferner:

$$\frac{a(Y_1)}{m^*(x)} \leq \frac{a(Y_1)}{L} \overset{(*)}{\leq} 1 + \frac{a_h}{2L} \overset{[+]}{\leq} 1 + \frac{1}{k(r)+1} \leq 1 + \frac{1}{\frac{2-r}{r-1} + 1} = r.$$

□

Im Abschnitt über Dynamisches Programmieren hatten wir ein PTAS für MaximumKnapsack kennen gelernt, das ebenfalls darauf beruht, für eine (geeignet verstandene) „Subinstanz“ eine optimale Lösung zu bestimmen und diese dann (geeignet modifiziert) als Lösung des ursprünglichen Problems zu betrachten.
Diese Vorgehensweise ist typisch für die Entwicklung von PTAS. Wir wollen hierzu noch ein Beispiel aus nderen Bereichen studieren, zunächst das Problem, größtmögliche unabhängige Mengen in planaren Graphen zu finden.

Dazu benötigen wir noch einige Begriffe:

Es sei eine planare Einbettung (also eine „Zeichnung“ in der Ebene) eines planaren Graphen G gegeben. Die **Schicht** eines Knotens von G ist induktiv wie folgt bestimmt:

1. Alle Knoten, die an die äußere Fläche grenzen, gehören zur Schicht 1.
2. Für jedes $i > 1$ seien die Knoten der Schicht $1, \ldots, i-1$ bekannt; wir entfernen diese (in Gedanken) aus dem eingebetteten Graphen. Alle Knoten, die nun an die äußere Fläche grenzen, gehören zur Schicht i.

Die Anzahl der Schichten eines eingebetteten Graphen heißt auch seine *Außerplanarität*.

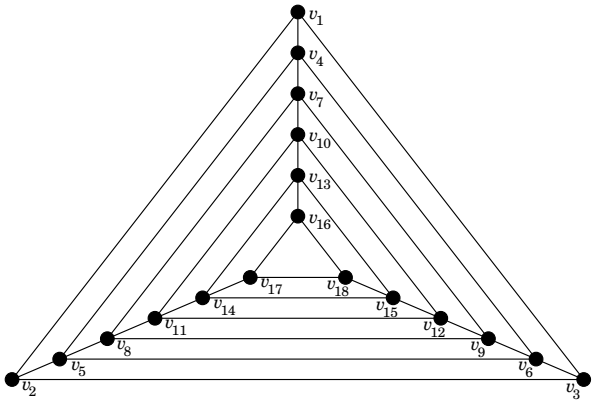


Abbildung 6:

Abbildung 6 zeigt die planare Einbettung eines Graphen mit sechs Schichten. Solche Einbettungen können aber sehr viel komplizierter werden, mit Antennen, meheren „Zentren“ etc.

B. S. Baker konnte zeigen:

Mitteilung 12.3 *Ist K die Außerplanarität eines eingebetteten Graphen G mit n Knoten, so kann eine größtmögliche unabhängige Menge für G in Zeit $O(8^K \cdot n)$ bestimmt werden.*

Dieses Resultat verwendete sie, um ein PTAS für MIS (auf planaren Graphen) anzugeben, welches wir jetzt vorstellen.

Satz 12.4 *MIS (auf planaren Graphen) $\in$ PTAS.*

Beweis: Betrachte dazu das folgende Programm:

MIS-PTAS $(G = (V, E), r)$

1. Setze $K := \lceil 1/(r-1) \rceil$.
2. Bestimme eine planare Einbettung von G.

3. Berechne die Knotenschichten.
   (Sei V_i die Menge der Knoten auf Schicht i.)
4. Für $i = 0$ bis K tue

   4a. Sei $\overline{V_i}$ die Vereinigung aller V_j mit $j \equiv i \mod K+1$

   4b. $G_i := G(V \setminus \overline{V_i})$
   (G_i hat Außerplanarität K)

   4c. Berechne größtmögliche unabhängige Mengen I_i von G_i.
5. Sei $I_m (m = 0, \ldots, K)$ derart, dass $|I_m| = \max\{|I_i| \mid 0 \leq i \leq K\}$.
6. Liefere I_m zurück.

Es sei $r > 1$ die angestrebte Approximationsgüte und setze $K := \left\lceil \frac{1}{r-1} \right\rceil$. Sei ein eingebetteter Graph G gegeben. Wähle $V_i, \overline{V_i}, G_i$ wie im Programm. Nach Konstruktion (jede $K+1$ Schicht ist entfernt worden!) ist jedes G_i K-außerplanar. Eine größtmögliche unabhängige Menge I_i von G_i kann in Zeit $O(8^K \cdot n)$ (gemäß obiger Mitteilung) gefunden werden. Es sei I_m eine Menge größtmöglicher Mächtigkeit unter den $I_0, \ldots, I_K$.
Bezeichne nun I^* eine größtmögliche unabhängige Menge von G. Es gibt dann ein r, $0 \leq r \leq K$, mit $\left|\overline{V_r} \cap I^*\right| \leq |I^*|/(K+1)$. Da I_r eine größtmögliche unabhängige Menge auf G_r ist, gilt:

$$\begin{aligned} |I_r| &\geq |V \setminus \overline{V_r} \cap I^*| \\ &= |I^* \setminus (\overline{V_r} \cap I^*)| \\ &\geq |I^*| - \frac{|I^*|}{K+1} \\ &= \frac{K}{K+1}|I^*| \end{aligned}$$

Also ist

$$|I_m| \geq |I_r| \geq \frac{K}{K+1}|I^*|$$

Daraus folgt: $R(G, I_m) = \frac{I^*}{I_m} \leq \frac{K+1}{K} \leq r$. □

12.2 Das Kreisscheibenüberdeckungsproblem (Disc Cover)

Eine ähnliche Strategie wie Baker ersannen (in etwa gleichzeitig) D. S. Hochbaum und W. Maas für Überdeckungs- und Packungsprobleme in so genannten *geometrischen Graphen*.
Wir werden hier konkret das Überdeckungsproblem einer gegebenen n-Punktmenge der Euklidischen Ebene mit Hilfe von Kreisscheiben vom (einheitlichen) Durchmesser D betrachten.

$I =$ $\{(x_i, y_i) \mid 1 \leq i \leq n\} \subseteq \mathbb{R}^2$ $(, D)$
$S:$ $\{\mathcal{D} = \{D_1, \ldots, D_K\} \mid \bigcup_{i=1}^{K} D_i \supseteq I\}$
D_i Kreisscheibe vom Durchmesser D.
$m:$ $K = |\mathcal{D}|$
opt : min.

Um dieses Problem anzugehen, treffen wir zunächst einige Vereinbarungen:

Wir gehen davon aus, die Punkte aus I liegen alle in einem Rechteck $R \subseteq \mathbb{R}^2$. Des weiteren sei l der so genannte **Verschiebeparameter** (dessen Bedeutung weiter unten klar werden wird).
In einer ersten Phase unterteilen wir R in vertikale Streifen der Breite D; jeder Streifen sei links offen und rechts abgeschlossen. Wir betrachten stets Gruppen von l aufeinander folgenden Streifen, also *Makrostreifen* der Breite $l \cdot D$, abgesehen von evtl. schmaleren „Randstreifen“. Offenbar lässt sich R in l verschiedene Weisen in Makrostreifen unterteilen; $M_1, \ldots, M_l$ seien die entsprechenden Makrostreifenmengen.
Es sei $\mathcal{A}$ ein Algorithmus, der eine zulässige Lösung für Streifen der Maximalbreite $l \cdot D$ berechnet. Für eine Makrostreifenmenge M_i sei $\mathcal{A}(M_i)$ der Algorithmus, der $\mathcal{A}$ auf jeden Makrostreifen aus M_i anwendet und die Vereinigung der so erhaltenen Kreisscheibenmengen als Lösung liefert; diese ist sicherlich zulässig.
Der Verschiebealgorithmus $S_{\mathcal{A}}$ liefert nun eine solche Lösung, die minimal unter den Lösungen von $\mathcal{A}(M_1), \ldots, \mathcal{A}(M_l)$ ist.
Es seien ferner $r_{\mathcal{A}}$ und $r_{S_{\mathcal{A}}}$ die Leistungsgüten (performance ratios) von $\mathcal{A}$ und $S_{\mathcal{A}}$. Dann gilt:

Lemma 12.5 ***Das Verschiebelemma:***

$$r_{S_{\mathcal{A}}} \leq r_{\mathcal{A}} \left(1 + \frac{1}{l}\right).$$

Beweis: Per Definition ist

$$m_{\mathcal{A}(M_i)} = \sum_{J \in M_i} m_{\mathcal{A}}(J) \leq r_{\mathcal{A}} \cdot \sum_{J \in M_i} m^*(J).$$

Dabei seien $m_{\mathcal{A}}(J)$ bzw. $m^*(J)$ der von Algorithmus $\mathcal{A}$ bzw. einem optimalen Algorithmus gelieferte Wert, angewendet auf die durch den Makrostreifen J gegebene Subinstanz.
Es sei S^* eine optimale Lösung des Ausgangsproblems vom Wert m^*. Es sei $S^*[i] \subseteq S^*$ diejenige Menge von Kreisscheiben, die Punkte von I aus zwei angrenzenden Makrostreifen von M_i abdeckt.
Offenbar gilt:

$$\forall i, j : S^*[i] \cap S^*[j] \neq \emptyset \rightarrow i = j \qquad [+]$$

Da $m^*(J)$ der optimale Wert für Streifen J ist, gilt:

$$m^*(J) \leq |\{D \in S^* \mid D \cap J \neq \emptyset\}|$$

$$\begin{aligned} \rightsquigarrow \sum_{J \in M_i} m^*(J) &\leq \sum_{J \in M_i} |\{D \in S^* \mid D \cap J \neq \emptyset\}| \\ &= |S^*| + |S^*[i]|, \end{aligned}$$

denn (nur) die Scheiben aus $S^*[i]$ sind doppelt zu zählen. Da das Minimum durch das arithmetische Mittel majorisiert wird, gilt schließlich:

$$\begin{aligned} m_{S_{\mathcal{A}}} &= \min_{i=1,\ldots,l} m_{\mathcal{A}(M_i)} \leq \frac{1}{l}\sum_{i=1}^{l} m_{\mathcal{A}(M_i)} \\ &\leq \frac{1}{l}\sum_{i=1}^{l} r_{\mathcal{A}} \cdot \sum_{J\in M_i} m^*(J) \\ &\leq \frac{r_{\mathcal{A}}}{l}\sum_{i=1}^{l} (|S^*| + |S^*[i]|) \\ &= \frac{r_{\mathcal{A}}}{l}\left(l\cdot m^* + \sum_{i=1}^{l} |S^*[i]|\right) \\ &\leq \frac{r_{\mathcal{A}}}{l}(l+1)m^* \end{aligned}$$

□

Satz 12.6 *Das Kreisscheibenüberdeckungsproblem liegt in PTAS.*

Beweis: Durch zweimaliges (geschachteltes) Anwenden des Verschiebelemmas, wobei einmal das Ausgangsrechteck in vertikale und das andere Mal in horizontale Streifen unterteilt wird, erhalten wir einen Algorithmus $\mathcal{B}$ mit

$$r_{\mathcal{B}} \leq r_{\square}\left(1+\frac{1}{l}\right)^2.$$

$r_{\square}$ sei dabei die Leistungsgüte eines Algorithmus für eine quadratische Fläche der größtmöglichen Seitenlänge $l \cdot D$. Wir zeigen nun: $r_{\square} = 1$ für einen Algorithmus, der (lediglich) exponentiell in l ist.

Damit ist die Behauptung gezeigt, denn für vorliegende Leistungsgüte r braucht nur ein l mit $\left(1+\frac{1}{l}\right)^2 \leq r$ bestimmt zu werden.

Betrachte nun ein Quadrat der größtmöglichen Seitenlänge $l \cdot D$, das mit Kreisscheiben vom Durchmesser D abzudecken ist. Klar ist, dass man höchstens $O(l^2)$ viele Scheiben benötigt, um das vorliegende Quadrat lückenlos abzudecken. [+]
Es sei im Weiteren $\tilde{n}$ die Anzahl der (ursprünglich) vorgegebenen n Punkte, die im betrachteten Quadrat liegen. Es sei x einer dieser Punkte. Wird x (in einer optimalen Überdeckung) von einer Scheibe abgedeckt, die keinen anderen Punkt abdeckt, so ist die exakte Position dieser Scheibe im Grunde gleichgültig.
Andernfalls gibt es noch andere Punkte, die von der selben Kreisscheibe abgedeckt werden. O.E. können wir annehmen, zwei dieser Punkte liegen auf dem Rand der Scheibe. Da durch jene zwei Punkte nur zwei verschiedene Kreise gezogen werden können, die diese Punkte auf dem Rand liegen haben, gibt es insgesamt höchstens $2 \cdot \begin{pmatrix} \tilde{n} \\ 2 \end{pmatrix} = O(\tilde{n}^2)$ viele Kreisscheibenpositionen zu berücksichtigen.
Wegen [+] sind an diesen Positionen (höchstens) $O(l^2)$ viele auszuwählen, so dass insgesamt maximal $O(\tilde{n}^{p(l)})$ viele Kreisscheibenanordnungen durchzutesten sind für ein Polynom p. □

12.3 Zum Zusammenhang mit der Konstruktion von Baker

Ist $\mathcal{D}$ eine Menge von Kreisscheiben, so ist $\mathcal{D}$ in natürlicher Weise der folgende Graph zugeordnet. $G_{\mathcal{D}} = (\mathcal{D}, E_{\mathcal{D}})$ mit $\{D_1, D_2\} \in E$ gdw. $D_1 \cap D_2 \neq \emptyset$.
Diese geometrischen Graphen sind ein naheliegendes Modell für viele reale Probleme.

Mitteilung 12.7 *(Satz von Koebe):*
Ein geometrischer Graph $G_{\mathcal{D}} = (\mathcal{D}, E_{\mathcal{D}})$ ist planar, wenn $\forall \{D_1, D_2\} \in E_{\mathcal{D}} : |D_1 \cap D_2| = 1$. Jeder planare Graph lässt sich umgekehrt als geometrischer Graph darstellen, dessen Kreisscheiben sich paarweise in einem Punkt treffen (also sich nur berühren).

Speziell kümmern wir uns nun um **Einheitskreisgraphen**, das sind geometrische Graphen mit Kreisscheiben vom Radius 1.

Satz 12.8 *MIS —eingeschränkt auf Einheitskreisgraphen— liegt in PTAS.*

Beweis: (Skizze) Wie in vorigem Satz unterteilt man ein die tangierten Punkte der Ebene einschließendes Rechteck nacheinander in vertikale und horizontale Streifen der Breite k. Es ist leicht einzusehen, dass eine unabhängige Menge von Einheitskreisscheiben in einem Quadrat der Seitenlänge k höchstens $O(k^2)$ viele Elemente hat. Dies ermöglicht wiederum, MIS für Quadrate der Seitenlänge k optimal in einer Zeit $O(n^{\mathrm{pol}(k)})$ zu lösen, wobei n die Zahl der gegebenen Kreisscheiben und pol ein Polynom ist.
Indem wir jetzt (für jeden vertikalen Streifen) alle Graphen G_i betrachten, deren Streifen-Zentren in einem der Quadrate $Q_1, \ldots, Q_q$ (der Seitenlänge jeweils $\leq k$) liegen für ein Q_j mit $j \neq i \mod (n+1)$, so können wir eine $\frac{k}{k+1}$-Näherung an ein MIS für solch einen Streifen erhalten. □

Literatur:

- H. B. Hunt, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns: A unified approach to approximation schemes for NP- and PSPACE-hard problems for geometric graphs. In: Algorithms ESA'94, LNCS 855 (1994), 424-435.
- Hochbaum/Maass: Approximation schemes for covering and packing problems in image processing and VLSI. In: J. ACM 32 (1985), 130-136.

12.4 APX versus PTAS

Es stellt sich natürlich die Frage, ob die triviale Inklusion PTAS $\subseteq$ APX echt ist oder nicht. Der folgende Satz beantwortet diese Frage mit einer in der Komplexitätstheorie üblichen Relativierung.

Satz 12.9 *Gilt $P \neq NP$, so $PTAS \subsetneq APX$.*

Beweis: Wir zeigen, dass —voraussgesetzt P $\neq$ NP— kein r-approximativer für das in Kapitel 5 als zu APX gehörig gezeigte Bin-Packing-Problem existiert für $r \leq 3/2 - \epsilon$ für jedes $\epsilon > 0$.
Betrachte dazu folgende NP-vollständige Entscheidungsproblemvariante für das soeben behandelte Partitionsproblem PARTITION.

Ggb: Gegenstände $X = \{x_1, \ldots, x_n\}$ mit Gewichten $a_i \in \mathbb{N}$
Gefragt: Gibt es eine Bipartition (Y_1, Y_2) von X mit

$$\sum_{x_j \in Y_1} a_j = \sum_{x_j \in Y_2} a_j \qquad ?$$

Wir zeigen, wie man PARTITION in Polynomzeit lösen könnte, wenn es für Bin-Packing eine r-Approximation mit $r \leq 3/2 - \epsilon$ gäbe. Sei also (X, a) eine Instanz von Partition $B := \sum_{x \in X} a(x)$. Wir definieren nun die „zugehörige“ Instanz von Bin-Packing:
Jedem Gegenstand $x_i \in X$ mit Gewicht a_i ordnen wir einen Gegenstand x_i' der Größe $a_i' = \frac{2a_i}{B}$ zu. Dabei können wir $a_i \leq B/2$ voraussetzen, da wir sonst trivialerweise eine unlösbare Instanz von PARTITION vorzuliegen hätten.
$\rightsquigarrow$ Instanz (X, a') von Bin-Packing. Ist nun (X, a) eine positive Instanz von PARTITION, so ist $m^*(X, a') = 2$ für das zugehörige Bin-Packing-Problem. Ist aber (X, a) eine negative Instanz von PARTITION, so wird in jedem Fall ein dritter Behälter für (X, a') benötigt. Daraus ergibt sich die Behauptung, denn die angegebene Reduktion ist Polynomzeit berechenbar. □

Störend an den bisher in diesem Kapitel vorgestellten Approximationsschemata mag sein, dass die Laufzeit der Algorithmen exponentiell von der angestrebten Güte der Approximation abhängt —man möge dies überprüfen! Wir stellen jetzt einen „schöneren“ PTAS-Begriff vor.

Definition 12.10 *Es sei $\mathcal{P} \in NPO$. Ein Algorithmus $\mathcal{A}$ heißt **volles Polynomzeit-Approximations-Schema** (engl. fully PTAS, kurz FPTAS), falls $\mathcal{A}$ für jedes Paar von Eingaben (x, r), x eine Instanz von $\mathcal{P}$ und $r > 1$, eine r-approximative Lösung zurück liefert in einer Zeit, die polynomiell in $|x|$ und in $\frac{1}{r-1}$ ist.*
Die zugehörige Problemklasse nennen wir FPTAS.

In Abschnitt 4.1 haben wir bereits ein (einfaches) Beispiel für ein Problem aus FPTAS kennen gelernt, nämlich das Rucksackproblem. Tatsächlich gibt es nur verhältnismäßig wenige (natürliche) Probleme in FPTAS; die meisten von ihnen sind Verwandte des Rucksackproblems.
Der Grund scheint darin zu liegen, dass man von einer ganzen Reihe von NPO-Problemen leicht nachweisen kann, dass sie nicht zu FPTAS gehören.

Definition 12.11 *Ein Optimierungsproblem heißt **polynomiell beschränkt**, wenn es ein Polynom p gibt derart, dass —für jede Instanz $x \in I_{\mathcal{P}}$ und jede zulässige Lösung $y \in S_{\mathcal{P}}(x)$— $m(x, y) \leq p(|x|)$ gilt.*

Satz 12.12 *Gilt $P \neq NP$, so gehört kein NP-hartes polynomiell beschränktes Optimierungsproblem zu FPTAS.*

Beweis: Angenommen, es gibt ein FPTAS $\mathcal{A}$ für das Maximierungsproblem $\mathcal{P}$, dessen Laufzeit durch $q(|x|, \frac{1}{r-1})$ für ein Polynom q beschränkt ist (für jede Instanz x und jedes $r > 1$). Da $\mathcal{P}$ polynomiell beschränkt ist, gibt es ein Polynom p mit $m^*(x) \leq p(|x|)[*]$ für jede Instanz x. Wir zeigen: für $r = 1 + \frac{1}{p(|x|)}$ liefert $\mathcal{A}(x, r)$ eine optimale Lösung für x (in Polynomzeit $q(|x|, p(|x|))$:

Da $m(x, \mathcal{A}(x, r))$ r-Approximation ist, gilt:

$$m(x, \mathcal{A}(x, r)) \geq m^*(x)\frac{p(|x|)}{p(|x|)+1} = m^*(x) - \frac{m^*(x)}{p(|x|)+1} \overset{[*]}{>} m^*(x) - 1$$

Da $m(x, \mathcal{A}(x, r)), m^*(x) \in \mathbb{N}$ und $m^*(x)$ größtmöglich, folgt

$$m^*(x) = m(x, \mathcal{A}(x, r)).$$

Aus der NP-Härte von $\mathcal{P}$ folgt nun die Behauptung. □

12.5 NP-Härte und Pseudo-Polynomialität

Eine gewisse Sonderrolle spielen Optimierungsprobleme, deren Schwierigkeit von der gewählten Codierung der (ganzen) Zahlen (unär oder binär) abhängt.
Ist x eine Instanz eines NPO-Problems, so bezeichne $\max(x)$ den Betrag der betragsmäßig größten vorkommenden Zahl.

Definition 12.13 *Ein NPO-Problem $\mathcal{P}$ heißt **pseudo-polynomiell**, wenn es eine Algorithmus $\mathcal{A}$ zu Lösung von $\mathcal{P}$ gibt, der, für jede Instanz x, mit einer Laufzeit arbeitet, die durch ein Polynom in $|x|$ und $\max(x)$ beschränkt ist.*

Beispiel 12.14 *Das Rucksackproblem ist pseudo-polynomiell, siehe Satz 8.1. Für eine Instanz $x = (X = \{x_1, \ldots, x_n\}, \{p_1, \ldots, p_n\}, \{a_1, \ldots, a_n\}, b)$ würde $\max(x) = \max(p_1, \ldots, p_n, a_1, \ldots, a_n, b)$ sein.*
Für die Laufzeit des Algorithmus von Kapitel 8 ist zu beachten:

$$O\left(n\sum_{i=1}^{n} p_i\right) \subseteq O\left(n^2 p_{\max}\right) \subseteq O\left(|x|^2 \max(x)\right).$$

Satz 12.15 *Es sei $\mathcal{P} \in FPTAS$ für $\mathcal{P}$. Gibt es ein Polynom p, sodass für alle Instanzen $x \in I_{\mathcal{P}}$ gilt:*

$$m^*(x) \leq p(|x|, \max(x)),$$

so ist $\mathcal{P}$ pseudo-polynomiell.

Beweis: Es sei $\mathcal{A}$ ein FPTAS für $\mathcal{P}$. Betrachte

$$\mathcal{A}'(x) = \mathcal{A}\left(x, 1 + \frac{1}{p(|x|, \max(x)) + 1}\right).$$

Aufgrund der Ganzzahligkeit der Lösungen liefert $\mathcal{A}'(x)$ ein Optimum. Ist die Laufzeit von $\mathcal{A}(x, r)$ durch $q\left(|x|, \frac{1}{r-1}\right)$ beschränkt, so ist die Laufzeit von $\mathcal{A}(x)$ durch $q(|x|, p(|x|, \max(x)) + 1)$ beschränkt, d.h. $\mathcal{P}$ ist pseudo-polynomiell. □

Definition 12.16 *Es sei $\mathcal{P}$ ein NPO-Problem und p ein Polynom. $\mathcal{P}^{\max,p}$ bezeichne die Einschränkung von $\mathcal{P}$ auf Instanzen x mit* $\max(x) \leq p(|x|)$. *$\mathcal{P}$ heißt stark NP-hart, wenn es ein Polynom p gibt, sodass $\mathcal{P}^{\max,p}$ NP-hart ist.*

Satz 12.17 *Ist $P \neq NP$, so ist kein stark NP-hartes Problem pseudo-polynomiell.*

Beweis: Angenommen, $\mathcal{P}$ ist ein stark NP-hartes Problem, das auch pseudo-polynomiell ist. Dann gibt es einen Algorithmus $\mathcal{A}$, der $\mathcal{P}$ in der Zeit $q(|x|, \max(x))$ löst für ein geeignetes Polynom q und jede Instanz $x \in I_{\mathcal{P}}$. Für jedes Polynom p kann $\mathcal{P}^{\max,p}$ daher in Zeit $q(|x|, p(|x|))$ gelöst werden. Da $\mathcal{P}$ stark NP-hart ist, müsste für ein p jedoch $\mathcal{P}^{\max,p}$ NP-hart sein, woraus P = NP folgt. □.

Folgerung 12.18 *Ist $\mathcal{P}$ ein stark NP-hartes NPO-Problem und gibt es ein Polynom p, sodass $m^*(x) \leq p(|x|, \max(x))$ für jede Instanz $x \in I_{\mathcal{P}}$, so gilt $\mathcal{P} \notin FPTAS$, wenn nicht $P = NP$.* □

Die Sätze aus diesem Abschnitt gestatten es, für ein Problem zu schließen, dass es (unter gewissen komplexitätstheoretischen Annahmen) kein FPTAS besitzt. Dazu dient auch der folgende Zusammenhang, für dessen genaueres Verständnis wir auf die Vorlesung über parametrisierte Algorithmen verweisen.

Mitteilung 12.19 *Besitzt das einem Optimierungsproblem $\mathcal{P}$ entsprechende parametrisierte Entscheidungsproblem* keinen *Festparameteralgorithmus, so liegt $\mathcal{P}$ nicht in FPTAS.*

13 Zwischen APX und NPO

Wie wir in Kapitel 11 gesehen hatten, gibt es Probleme (wie das Handelsreisendenproblem), die sich (vorbehaltlich P $\neq$ NP) nicht bis auf einen konstanten Faktor mit einem Polynomzeitalgorithmus angenähert lösen lassen.
In solchen Fällen stellt sich die Frage, ob es denn wenigstens möglich ist, Leistungsgarantien für Approximationsalgorithmen zu finden, die von der Länge der Instanz abhängen.
In diesem Abschnitt lernen wir Beispiele für solche Näherungsalgorithmen kennen.
Betrachten wir zunächst eine weitere Verallgemeinerung des uns nun schon wohlbekannten Knotenüberdeckungsproblems:

13.1 Das Mengenüberdeckungsproblem (Set Cover, SC)

I : Eine Kollektion C von Teilmengen einer endlichen Grundmenge S.
S : Eine Mengenüberdeckung $C' \subseteq C$ für S, d.h. eine Teilkollektion, sodass jedes Element der Grundmenge S zu wenigstens einem Element der Teilkollektion gehört.
m : $|C'|$
opt : min

Zusammenhang mit Knotenüberdeckung Das Knotenüberdeckungsproblem ist insofern ein Mengenüberdeckungsproblem, als dass dort die Grundmengenelemente „Kanten“ heißen und ein Element der Kollektion einem „Knoten“ entspricht, genauer der Menge von Kanten, die mit nämlichen Knoten inzidieren.
Insofern ist der folgende Greedy-Algorithmus für das Mengenüberdeckungsproblem eine unmittelbare Verallgemeinerung eines Greedy-Algorithmus des Knotenüberdeckungsproblems, denn die Auswahl eines Elementes der Kollektion mit größter Mächtigkeit entspricht gerade der Wahl eines Knotens von größtem Grad.
In ähnlicher Weise lässt sich auch das Hitting-Set-Problem als Spezialfall des Mengenüberdeckungsproblems auffassen (und umgekehrt).

GreedySC (S, C)

0. Teste, ob $\bigcup_{c \in C} c = S$ gilt.
1. $U := S$
2. Für jede Menge $c_i \in C$ setze $c'_i := c_i$;
3. $C' := \emptyset$;
4. Solange $U \neq \emptyset$ tue:
   - 4a. Wähle unter den c'_i eine Menge c'_j größter Mächtigkeit
   - 4b. $C' := C' \cup \{c^*_j\}$;
   - 4c. $U := U \setminus c'_j$;
   - 4d. Für jede Menge c'_i setze $c'_i := c'_i - c'_j$.

5. Liefere C' zurück.

Satz 13.1 *Ist $n = |S|$, so ist GreedySC ein $(\lfloor \ln n \rfloor + 1)$-approximativer Polynomzeitalgorithmus für das Mengenüberdeckungsproblem.*

Beweis: Starten wir zunächst mit einem kleinen mathematischen

---

Exkurs

$H(r) := \sum_{i=1}^{r} \frac{1}{i}$ bezeichnet die **r-te harmonische Zahl**. Die Folge $(H(r))$ konvergiert nicht, wächst aber recht langsam, genauer gesagt wie der natürliche Logarithmus. Noch genauer gilt (siehe Graham/Knuth/Patashnik: Concrete Mathematics, Addison-Wesley, 1989):

$$\lfloor \ln r \rfloor \leq H(r) \leq \lfloor \ln r \rfloor + 1$$

Um die Aussage des Satzes zu beweisen, werden wir zeigen:

$$\sum_{c_i \in C^*} H\left(|c_i|\right) \geq |C'| \qquad (*)$$

Dabei ist C^* irgendeine optimale Überdeckung und C' eine GreedyVC gelieferte Lösung. Wegen $|c_i| \leq n$ folgt aus $(*)$:

$$|C'| \leq \sum_{c_i \in C^*} H\left(|c_i|\right) \leq \sum_{c_i \in C^*} H(n) \leq |C^*|\, H(n) \leq |C^*| \left(\lfloor \ln n \rfloor + 1\right),$$

was die Satzbehauptung liefert. (Die Polynomzeitbehauptung ist trivial.)

Um $(*)$ nachweisen zu können, führen wir noch einige Bezeichnungen ein:
Ist $x = \left(S, \overbrace{\{c_1, \ldots, c_m\}}^{=C}\right)$ eine SC-Instanz, so sei $a_1, \ldots, a_{|C'|}$ die Folge der Indizes derjenigen Teilmengen aus der Kollektion, die zu $C' \subseteq C$ gehören, d.h.

$$C' = \left\{c_{a_1}, \ldots c_{a_{|C'|}}\right\}.$$

Für jedes $j \in \{1, \ldots, |C'|\}, i \in \{1, \ldots, m\}$ sei c_i^j der „überlebende“ Teil von c_i, **bevor** die Teilmenge c_{a_j} gewählt worden ist durch GreedySC.
Damit ist $c_i^1 = c_i$ (für alle $i \in \{1, \ldots, m\}$).
Außerdem vereinbaren wir: $c_i^{|C'|+1} = \emptyset$ für alle $i \in \{1, \ldots, m\}$. Die Menge der Elemente von c_i, die das erste Mal durch c_{a_j} überdeckt werden, ist

$$c_i \cap c_{a_j}^j = c_i^j \cap c_{a_j}^j = c_i^j \setminus c_i^{j+1}. \qquad [+]$$

l_i sei der größte Index, für den $c_i^{l_i} \neq \emptyset$ gilt, d.h. $c_i^{l_i+1} = \emptyset$, m.a.W., nachdem $c_{a_{l_i}}$ zur Überdeckungs-Kollektion hinzugefügt wurde, sind alle Elemente von c_i abgedeckt.

Mit diesen Bezeichnugsweisen zeigen wir zwei Behauptungen:

Behauptung 1:

$$\forall i \in \{1, \ldots, m\} : H(|c_i|) \geq \sum_{j=1}^{|C'|} \frac{\left|c_i \cap c_{a_j}^j\right|}{\left|c_{a_j}^j\right|}$$

Behauptung 2: $\forall C'' \subseteq C, C''$ ist Mengenüberdeckung

$$\sum_{c_i \in C''} \sum_{j=1}^{|C'|} \frac{\left|c_i \cap c_{a_j}^j\right|}{\left|c_{a_j}^j\right|} \geq |C'|$$

Aus beiden Behauptungen folgt sofort $(*)$, denn Behauptung 2 gilt insbesondere für $C'' = C^*$, also ist

$$|C'| \leq \sum_{c_i \in C^*} \sum_{j=1}^{|C'|} \frac{\left|c_i \cap c_{a_j}^j\right|}{\left|c_{a_j}^j\right|} \leq \sum_{c_i \in C^*} H(|c_i|).$$

Beide Behauptungen ergeben sich nun nach einigen leichten algebraischen Manipulationen.
Zu Behauptung 1: Es sei $i \in \{1, \ldots, m\}$. Da GreedySC für jedes $1 \leq j \leq |C'|$ stets eine größte überlebende Menge wählt, gilt

$$[\$] \qquad \left|c_i^j\right| \leq \left|c_{a_j}^j\right|$$

in unserer Notation.

Also ist:

$$\begin{aligned}
\sum_{j=1}^{|C'|} \frac{\left|c_i \cap c_{a_j}^j\right|}{\left|c_{a_j}^j\right|} &\overset{[+]}{=} \sum_{j=1}^{C'} \frac{\left|c_i^j\right| - \left|c_i^{j+1}\right|}{\left|c_{a_j}^j\right|} \\
&\overset{[\$]}{\leq} \sum_{j=1}^{|C'|} \frac{\left|c_i^j\right| - \left|c_i^{j+1}\right|}{\left|c_i^j\right|} \\
&\overset{\text{Def. } l_i}{\leq} \sum_{j=1}^{l_i} \sum_{K=|c_i^{j+1}|+1}^{|c_i^j|} \frac{1}{|c_i^j|} \\
&\overset{[\sharp]}{\leq} \sum_{j=1}^{l_i} \sum_{k=1}^{|c_i^j|-|c_i^{j+1}|} \frac{1}{k+|c_i^{j+1}|} \\
&\overset{|c_i^{j+1}|\geq 0}{\leq} \sum_{j=1}^{l_i} \sum_{k=1}^{|c_i^j|-|c_i^{j+1}|} \frac{1}{k} \\
&= \sum_{j=1}^{l_i} \left(H\left(\left|c_i^j\right|\right) - H\left(\left|c_i^{j+1}\right|\right)\right) \\
&\overset{\text{Teleskop}}{=} H\left(\left|c_i^1\right|\right) - H\left(\left|c_i^{l_i+1}\right|\right) \\
&\overset{\text{s. o.}}{=} H\left(|c_i|\right)
\end{aligned}$$

Die Ungleichung [♯] ist trivial, denn sie bedeutet:

$$\forall k = 1, \ldots, |c_i^j| - |c_i^{j+1}| : |c_i^j| \geq k + |c_i^{j+1}|.$$

Zu Behauptung 2:

$$\begin{aligned}
\sum_{c_i \in C''} \sum_{j=1}^{|C'|} \frac{\left|c_i \cap c_{a_j}^i\right|}{\left|c_{a_j}^j\right|} &= \sum_{j=1}^{|C'|} \frac{1}{\left|c_{a_j}^j\right|} \sum_{c_i \in C''} \left|c_i \cap c_{a_j}^j\right| \\
&\overset{C'' \text{ Überdeckung!}}{\geq} \sum_{j=1}^{|C'|} \frac{1}{\left|c_{a_j}^j\right|} \cdot \left|c_{a_j}^j\right| = |C'|
\end{aligned}$$

Beim letzten $\geq$ ist $\neq$ möglich, falls gewisse Elemente mehrfach abgedeckt werden. □

Bemerkung 13.2 *Die in Satz 13.1 angegebene Abschätzung der Leistungsgüte von GreedySC ist nicht verbesserbar. Es gibt eine Folge von Instanzen, für die die Schranke auch erreicht wird.*

Bemerkung 13.3 *GreedySC ist optimal in dem Sinne, dass es (unter einigen „wahrscheinlich“ komplexitätstheoretischen Annahmen) für kein $\epsilon > 0$ einen „$(\ln - \epsilon)$-Approximationsalgorithmus“ für SC gibt.*

Betrachten wir jetzt noch ein weiteres Problem „jenseits von APX“:

13.2 Graphenfärben

Schon weiter oben haben wir diskutiert, wie man Algorithmen zum Auffinden von unabhängigen Mengen in Graphen „gebrauchen“ kann, um Algorithmen zum Graphenfärben zu erhalten. Auf dieser Idee fußt auch folgendes Verfahren.

GreddyGC (G=(V,E))

1. $i := 0; U := V$
2. Solange $U \neq \emptyset$ tue:

   2a. $i := i + 1$;
   2b. $I :=$ GreedyIndependentSet$(G(U))$;
   2c. Färbe Knoten aus I mit „Farbe“ i;
   2d. $U := U \setminus I$
3. Liefere so erhaltene Färbung $f : V \to \mathbb{N}$ zurück.

Lemma 13.4 *Ist $G = (V, E)$ k-färbbar, so benötigt GreedyGC höchstens $\frac{3|V|}{\log_k(|V|)}$ viele Farben zum Färben von G.*

Beweis: Es sei $H = G(U)$ irgendein im Schritt 2b. verarbeiteter Teilgraph von G. Da G k-färbbar ist, ist auch H k-färbbar. Zum Fortgang des Beweises brauchen wir folgenden

Hilfsatz 13.5 *Wird ein Graph H GreedyIndependentSet als Eingabe gegeben, so wird eine unabhängige Menge $I \subseteq U$ geliefert, die wenigstens $\lceil \log_k(|U|) \rceil$ viele Knoten enthält.*

Beweis des Hilfsatzes: Da H k-färbbar, enthält H eine unabhängige Menge $\overline{I}$ mit wenigstens $|U|/k$ Knoten. Jeder Knoten dieser Menge hat einen Maximalgrad $(|U| - |U|/k)$ (denn sonst gäbe es Verbindungen zwischen Knoten $\overline{I}$). Damit ist trivialerweise der Minimalgrad in H maximal $(|U| - |U|/k)$. Da als erster Knoten v_1 (für I) von GreedyIndependentSet einer minimalen Grades gewählt wird und er samt seinen höchstens $(|U| - |U|/k)$ vielen Nachbarn entfernt wird, wird der nächste Knoten v_2 in $H(U \setminus N[v_1])$ gesucht mit

$$|U \setminus N[v_1]| > |U| - (|U|/k + 1) = |U|/k - 1.$$

Da $H(U \setminus N[v_1])$ k-färbbar, lässt sich das Argument wiederholen.
Abgebrochen wird diese Suche, wenn U erschöpft ist.
Dazu sind mindestens $\lceil \log_k(|U|) \rceil$ viele Schritte nötig, und soviele Knoten werden auch wenigstens zu I hinzugenommen. □

Wegen des Hilfsatzes werden wenigstens $\lceil \log_k(|U|) \rceil$ viele Knoten mit Farbe i gefärbt. Wie groß ist U vor dem Durchlaufen der Schritte 2a.-2d.?

Fall 1: Solange $|U| \geq |V|/\log_k(|V|)$, gilt wegen $|V|/\log_k(|V|) > \sqrt{|V|}$:

$$\lceil \log_k |U| \rceil \geq \log_k |U| \geq \log_k \left(|V| / \log_k (|V|) \right) > \log_k \sqrt{|V|} = \frac{1}{2} \log_k (|V|)$$

Da U bei jedem Durchlauf der Schleife „2“ um wenigstens $\frac{1}{2} \log_k (|V|)$ abnimmt, kommt der Fall 1 höchstens $2\,|V| / \log_k (|V|)$ oft zur Anwendung, und dabei werden höchstens $2\,|V| / \log_k (|V|)$ viele Farben benutzt.

Fall 2: Sobald $|U| < |V|/\log_k(|V|)$, reichen trivialerweise $|V|/\log_k(|V|)$ viele Farben aus.

Fall 1 und Fall 2 zusammen liefern die Behauptung des Lemmas. □

Satz 13.6 *Ist $n = |V|$, so ist GreedyGC ein $O(n/\log(n))$-approximativer Algorithmus fürs Graphenfärben.*

Beweis: Nach dem Lemma benötiget GreedyGC höchstens $3n/\log_k(n)$ viele Farben mit $k = m^*(G)$. Also ist

$$\frac{m(G, \text{GreedyGC}(G))}{m^*(G)} \leq \frac{3n \log(m^*(G)/\log(n)}{m^*(G)} \leq \frac{3n}{\log(n)}.$$

□

Bemerkung 13.7 *Der soeben dargestellte $O(n/\log n)$-approximative Algorithmus ist nicht bestmöglich. So wurde ein $O\left(n \frac{(\log\log n)^2}{(\log n)^3}\right)$-approximativer Algorithmus gefunden. Auf der anderen Seite ist bekannt, dass es —sofern nicht $P = NP$— keinen $n^{1/7-\epsilon}$-approximativen Algorithmus geben kann für jedes $\epsilon > 0$. Insbesondere dürfte das Graphenfärbeproblem nicht in APX liegen.*

14 Zwischen PTAS und APX

Wir stellen hier eine Verallgemeinerung der Klasse PTAS vor, bei der die Näherungsgüte mit wachsendem Wert einer optimalen Lösung sich verbessern kann, die vorgegebene Schranke r also nur einen asymptotischen Sinn erfüllt.

Definition 14.1 *Ein NPO-Problem $\mathcal{P}$ liegt in PTAS$^\infty$, in Worten: $\mathcal{P}$ hat ein asymptotisches Approximationsschema, wenn es einen Algorithmus $\mathcal{A}$ gibt und eine Konstante k, sodass —für jede Instanz x von $\mathcal{P}$ und für jede rationale Zahl $r \geq 1$— $\mathcal{A}(x, r)$ in Polynomzeit eine Lösung liefert, deren Leistungsfaktor höchstens $r + \frac{k}{m^*(x)}$ beträgt.*

Natürlich gilt: PTAS $\subseteq$ PTAS$^\infty$ $\subseteq$ APX. $\quad (*)$

Mitteilung 14.2 *Unter der Annahme $P \neq NP$ sind beide Inklusionen in $(*)$ echt.*

Wir werden hier zwei Probleme aus PTAS$^\infty$ betrachten: das Kantenfärben in einem Graphen sowie Bin-Packing. Wir haben in Kapitel 12 gesehen, dass Bin-Packing nicht in PTAS liegt, sofern P $\neq$ NP. Damit hätten wir den ersten Teil der Mitteilung 14.2 am Ende dieses Kapitels sogar bewiesen.

14.1 Kantenfärben

I : Graph $G = (V, E)$
S : Eine Färbung der Kanten, d.h. eine Partition von E in $E_1, \ldots, E_K$, sodass für jedes $1 \leq i \leq K$ gilt: Keine zwei Kanten aus E_i haben einen gemeinsamen Endpunkt
m : Anzahl der Farben, d. h. also K.
opt : min.

Satz 14.3 *(Vizing) Es gibt einen Polynomzeitalgorithmus, welcher bei Eingabe eines Graphen G mit Maximalgrad Δ eine Kantenfärbung mit höchstens $\Delta + 1$ vielen Farben liefert.*

Beweis: Ist $G = (V, E)$ der Eingabegraph vom Maximalgrad Δ, so färbt $\mathcal{A}$ G, indem $\mathcal{A}$ nacheinander die Kanten von E färbt, wobei evtl. „frühere“ Kantenfärbungen später revidiert werden. $\mathcal{A}$ arbeitet im Grunde wie folgt:

1. $E' := E; \overline{E} := \emptyset$;
2. Solange $E' \neq \emptyset$, tue:
   2a) Wähle Kante $\{u, v\} \in E'$.
   2b) Erweitere die Kantenfärbung von $\overline{E}$ auf $\overline{E} \cup \{\{u, v\}\}$, sodass $\big(V, \overline{E} \cup \{\{u, v\}\}\big)$ mit höchstens $\Delta + 1$ vielen Farben gefärbt ist.

2c) $E' := E' \setminus \{\{u,v\}\}; \overline{E} := \overline{E} \cup \{\{u,v\}\}$

Natürlich müssen wir 2b) jetzt noch präzisieren. Als Farbmenge nehmen wir $F = \{1,\ldots,\Delta+1\}$. Eine Färbung ist eine Abbildung $f : E \to F$. Wir gehen davon aus, dass für $(V;\overline{E})$ eine Kantenfärbung mit höchstens $\Delta+1$ vielen Farben konstruiert wurde. (Formal führen wir also einen Induktionsbeweis über $\left|\overline{E}\right|$ mit trivialen Induktionsanfang für $\overline{E} = \emptyset$.) f ist also partiell und vorläufig auf $\overline{E}$ festgelegt.

Wir führen folgende Notation ein:

- Ist $v \in V$ beliebig, so bezeichne

$$C(v) = \left\{\tilde{f} \in F \mid \neg\exists u \in V : \{u,v\} \in E \wedge f(\{u,v\}) = \tilde{f}\right\}$$

  die Farbmenge, die für „weitere Kanten“, die mit v inzidieren, noch „frei“ ist.

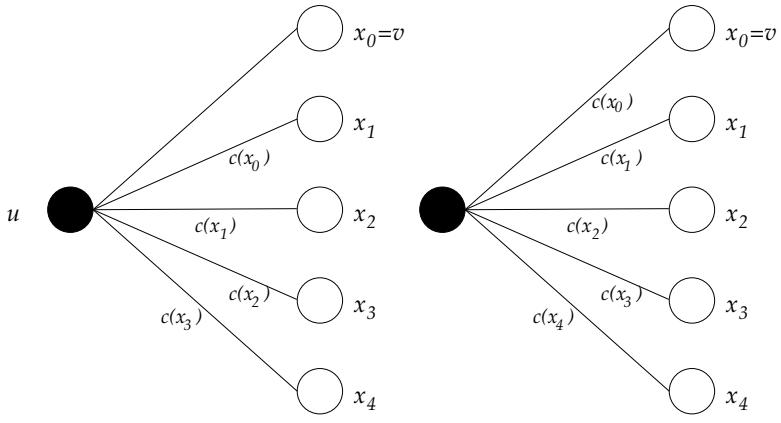
- Da $\delta(v) \leq \Delta$, ist für alle $v \in V$ die Menge $C(v) \neq \emptyset$. Daher gibt es eine Repräsentantenfunktion $c : V \to F$ mit $\forall v \in V : c(v) \in C(v)$.

Es sei jetzt (und im Folgenden) $\{u,v\}$ die in Schritt 2a) gewählte Kante. Wir betrachten jetzt sogenannte Kantenfarbfolgen, das sind mit u inzidierende [11] Knotenfolgen $x_0,\ldots,x_s$ mit $x_0 = v$ und $f(\{u,x_i\}) = c(x_{i-1})$. Eine Kantenfarbfolge heißt maximal, wenn es keine weitere Kanten $\{u,x\}$ in $\overline{E}$ gibt (mit $x \notin \{x_0,\ldots,x_s\}$) mit $f(\{u,x\}) = c(x_s)$.

Hilfsatz 14.4 *Ist $x_0,\ldots,x_s$ eine Kantenfarbfolge mit $c(x_s) \in C(u)$, so ist f zu einer Färbung auf $\overline{E} \cup \{\{u,v\}\}$ erweiterbar.*

Beweis: Setze einfach $f(\{u,x_i\}) := x(x_i)$ für $0 \leq i \leq s$ und lasse sonst die alte Färbung bestehen. □

Die Situation vor (links) und nach (rechts) dem Umfärben mit dem Hilfssatz, $s = 4$:



[11] Inzidenz bzgl. $\overline{E} \cup \{\{u,v\}\}$

In Polynomzeit kann nun eine maximale Kantenfarbfolge (an u) gefunden werden. Wegen dem Hilfsatz können wir $c(x_s) \notin C(u)$ annehmen.
Wegen der Maximalität der Folge gibt es ein $0 \leq i \leq s$ mit $f(\{u, x_i\}) = c(x_s)$, denn $c(x_s)$ ist ja nicht mehr „frei“ zum Färben von u-inzidenten Kanten.

Damit gilt:

$$c(x_{i-1}) = c(x_s).$$

Betrachte nun einen längsten Pfad p_{i-1} in $(V, \overline{E})$, der bei x_{i-1} startet und dessen Kanten abwechselnd mit $c(u)$ und $c(x_s)$ gefärbt sind. Es bezeichne w den letzten Knoten dieser Folge. Wir unterscheiden zwei Fälle:

(a) $w \neq u$. Dann können wir alle ursprünglich mit $c(u)$ gefärbten Kanten mit $c(x_s)$ färben und umgekehrt. Weiter färben wir nun Kante $\{u, x_{i-1}\}$ mit $c(u)$. Für diese erhaltene Kantenfarbfolge $x_0, \ldots, x_{i-1}$ ist der Hilfsatz anwendbar.

(b) $w = u$. Jetzt suchen wir in Linearzeit einem weiteren längsten Pfad p_s, der bei x_s startet und dessen Kanten abwechselnd mit $c(u)$ und $c(x_s)$ gefärbt sind. Angenommen p_s würde p_{i-1} irgendwo „schneiden“, so kann dies nicht „mittendrin geschehen“, da so an einem Knoten zwei verschiedene mit $c(u)$ oder $c(x_s)$ gefärbte Kanten anstoßen. Wäre aber der letzte Knoten von p_s gleich u, so stoßen an u zwei Kanten mit der selben Farbe $c(x_s)$.
Also können wir p_s analog zu (a) umfärben und dann den Hilfsatz anwenden. □

Folgerung 14.5 *Minimales Kantenfärben gehört zu PTAS$^\infty$.* □

Beweis: Da $\Delta(G) \leq m^*(G)$, gilt für die Lsung $m(G)$ unseres Algorithmus $m(G) \leq \Delta(G) + 1 \leq m^*(G) + 1$. Daher ist:

$$\frac{m(G)}{m^*(G)} \leq \frac{m^*(G) + 1}{m^*(G)} = 1 + \frac{1}{m^*(G)}.$$

□

Mitteilung 14.6 *Mit Hilfe der Gap-Technik lässt sich zeigen, dass das Kantenfärbeproblem kein PTAS besitzt, sofern nicht $P = NP$.*

14.2 Bin Packing (BP)

Der Schlüssel für die Entwicklung eines PTAS$^\infty$ für BinPacking liegt in der polynomiellen Lösbarkeit der folgenden eingeschränkten Variante: **Minimum** (c, δ) **eingeschränktes BP** (für $c \in \mathbb{N}, c > 0$ und $\delta \in \mathbb{Q}$, $\delta \leq 1$) ist gegeben, falls es höchstens c unterschiedliche Größen der Gegenstände gibt und jeder Gegenstand wenigstens δ groß ist (also einen Bruchteil der mit 1 angenommenen Kapazität jedes Behälters).
Um Bruchzahlen zu vermeiden bei Angabe der Gegenstandsgrößen s_i, gestatten wir noch die Angabe der Behältnisgröße B, weichen also formal von unserer Ursprungsdefinition von BinPacking leicht ab. In diesem Sinne wäre $(I = \{3 : 4, 5 : 2, 7 : 1\}, B = 8)$ eine Instanz von Minimum $(3, 3/8)$-eingeschränktem BP.

Lemma 14.7 *Minimum (c, δ)-eingeschränktem BP kann in der Zeit $O(n^q)$ gelöst werden, wobei n die Zahl der Gegenstände in der Eingabeinstanz ist und q nur von c und δ, nicht aber von n abhängt.*

Beweis: Es sei (I, B) eine Instanz von Min.(c, δ)-eingeschränktem BP. Der **Typ** eines Behälters ist ein c-dimensionaler Vektor $\vec{b} = (t_1, \ldots, t_c)$ von natürlichen Zahlen mit $0 \leq t_i \leq n_i$, sodass $\sum_{i=1}^{c} t_i s_i \geq B$.
Für jeden Typ gilt wegen $\delta B \leq s_i$:

$$\sum_{i=1}^{c} t_i \quad \leq \quad \frac{1}{\delta B} \sum t_i s_i \leq \frac{1}{\delta}$$

Erinnerung

$$\left| \left\{ (x_1, \ldots, x_s) \mid x_i \in \mathbb{N}, x_i \geq 0, \sum x_i \leq m \right\} \right| = \left(\begin{array}{c} m+s \\ m \end{array} \right) = \left(\begin{array}{c} m+s \\ s \end{array} \right)$$

Anwendung: Es gibt höchstens $q = \left(\begin{array}{c} c + \lfloor \frac{1}{\delta} \rfloor \\ c \end{array} \right)$ viele Typen von Behältern. Eine zulässige Lösung von Minimum (c, δ)-eingeschränktem BP kann daher durch einen q-dimensionalen Vektor $\vec{y} = (y_1, \ldots, y_q)$ beschrieben werden, wobei $y_i \geq 0$ angibt, wie viele Behälter von Typ i in der Lösung vorkommen. Da trivialer Weise höchstens n Behälter verwendet werden müssen, gibt es daher höchstens n^q verschiedene zulässige Lösungen. In diesem Raum kann in der Zeit $O(n^q)$ nach einer optimalen Lösung gesucht werden.

□

Das PTAS$^\infty$ für Bin-Packing lässt sich wie folgt skizzieren:

1. Entferne „kleine Gegenstände“.
2. Gruppiere die verbleibenden Gegenstände in eine konstante Zahl von Größenklassen.
3. Finde optimale Lösung für verbleibende Instanz (wie eben).
4. Mache die Gruppierung aus Schritt 2 rückgängig.
5. Füge die „kleinen Gegenstände“ wieder ein.

Ein Verfahren für den dritten Schritt haben wir soeben kennen gelernt. Schritt 1 dient im wesentlichen dazu, die erforderliche δ-Schranke zu bekommen, und Schritt 2 dient dazu, die Zahl der verschiedenen vorkommenden Größen zu beschränken.

Zum Gruppieren der Gegenstände

Es sei x eine Bin-Packing-Instanz. Die Gegenstände $x_1, \ldots, x_n$ seien der Größe nach absteigend sortiert. Für jede natürliche Zahl $k \leq n$ sei $m := \lfloor n/k \rfloor$. Teile die n

Gegenstände auf $m(+1)$ Gruppen G_i auf mit $G_i = \{x_{(i-1)k+1}, \ldots, x_{ik}\}$, $1 \leq i \leq m$ und $G_{m+1} = \{x_{mk+1}, \ldots, x_n\}$.
Wir definieren nun eine neue Instanz x_g^k von BP (mit der selben Behältergröße), sodass x_g^k für jeden Gegenstand x_j von der ursprünglichen Instanz, der aus G_i war, einen Gegenstand der Größe $s(x_{(i-1)k+1})$ enthält für $i = 2, \ldots, m+1$. Es gibt also höchstens mk Gegenstände in x_g^k, und diese Gegenstände haben höchstens m verschiedene Größen. Wir ignorieren die k größten Gegenstände auf diese Weise.

Lemma 14.8 $m^*(x_g^k) \leq m^*(x) \leq m^*(x_g^k) + k$.

Beweis: Jede Lösung für x_g^k ist trivialerweise eine für x, wenn wir weiter k Behälter für die ersten (größten) Gegenstände von x bereithalten.
Eine Lösung von x wiederum kann wie folgt in eine für x_g^k umgebildet werden: (a) Entferne alle Gegenstände aus der letzten Gruppe, (b) ersetze jeden Gegenstand aus G_i durch einen Gegenstand, der so groß ist wie x_{ik+1} (wenn nötig; G_{m+1} hat evtl. weniger Elemente als G_m). □

Zur Behandlung „kleiner Gegenstände“

Es sei x eine BP-Instanz. Für jedes $\delta \in \mathbb{Q}, \delta \in \left(0, \frac{1}{2}\right]$, sei x_δ die Instanz, die aus x durch Fortlassen aller Gegenstände, die kleiner als δB sind, entsteht. Haben wir nun eine Lösung für x_δ, die M Behälter benötigt, so benutzen wir FirstFit, um die kleinen Gegenstände wieder einzufügen.

Lemma 14.9 *Auf diese Weise lässt sich in Polynomzeit eine Lösung für x finden, die höchstens* $\max(M, (1+2\delta)m^*(x)+1)$ *viele Behälter benutzt.*

Beweis: Wir unterscheiden zwei Fälle:

(a) Es wurden keine neuen Behälter durch FirstFit geöffnet $\rightarrow M$ Behälter reichen.

(b) Es wurden $M' \geq 1$ neue Behälter durch FirstFit geöffnet. Wie schon in Kapitel 5 überlegt, enthalten alle Behälter (mit möglicher Ausnahme des zuletzt geöffneten) höchstens δB vielen freien Platz. Daher ist

$$(1-\delta) \underbrace{(M + M' - 1)}_{\#\text{ Behälter bis auf den letzten}} \leq \frac{\sum_{i=1}^{n} s(x_i)}{B} \leq m^*(x)$$

$$\rightsquigarrow M + M' \leq \frac{1}{1-\delta} m^*(x) + 1 \quad \leq \quad (1+2\delta)m^*(x) + 1$$

denn wegen $\delta \in \left(0, \frac{1}{2}\right]$ gilt $2\delta^2 \leq \delta$, also $1 \leq 1 - 2\delta^2 + \delta$. □

14.3 EIN PTAS$^\infty$ für BP

AsymptBP(x, B, r)

1. Falls $r \geq 2$ nimm NextFit (oder FirstFit) (s. Kapitel 5).
2. $\delta := (r-1)/2$;
3. Sei x_δ die aus x durch Fortlassen von Gegenständen der Größe $< \delta B$ entstehende Instanz; n' sei die Zahl der Gegenstände in x_δ.
4. $k := \lceil (r-1)^2 n'/2 \rceil$;
5. Sei $x^k_{\delta,g}$ die aus x_δ durch Gruppierung gebildete Instanz.
6. Finde optimale Lösung $x^k_{\delta,g}$ vom Wert $m^*(x^k_{\delta,g})$.
7. Füge die ersten (größten) k Gegenstände von x_δ in k „neue“ Behälter.
8. Wende FirstFit an, um die „kleineren Gegenstände“ wieder einzufügen.
9. Liefere die so erhaltene Packungsvorschrift zurück.

Satz 14.10 *AsymptBP ist ein PTAS$^\infty$ für BP.*

Beweis: AsymptBP läuft in Polynomzeit, da eine optimale Lösung für $x^k_{\delta,g}$ in Zeit n^q gefunden werden kann mit $q = f(\lceil n'/k \rceil, \delta)$, siehe Lemma 14.7. Beachte: $\lceil n'/k \rceil$ hängt nicht von n ab, nur von r (bzw. δ).
Wegen Lemma 14.8 ist der Wert einer Lösung von AsymptBP beschränkt durch $m^*(x^k_{\delta,g}) + k$. Alle Gegenstände in x_δ haben eine Größe von wenigstens δB, sodass $\delta n' \leq m^*(x_\delta)$ folgt, und damit

$$k \leq \frac{(r-1)^2}{2} n' + 1 = (r-1)\delta n' + 1 \leq (r-1) m^*(x_\delta) + 1.$$

Mit Lemma 14.8 folgt $m^*(x^k_{\delta,g}) + k \leq m^*(x_\delta) + (r-1)m^*(x_\delta) + 1 = r m^*(x_\delta) + 1$. Benutzen wir $r = (1 + 2\delta)$ in Lemma 14.9, so erhalten wir, dass maximal

$$\max(r m^*(x_\delta) + 1, r m^*(x) + 1) \leq r m^*(x) + 1$$

viele Behälter von AsymptBP benutzt werden. □

Man kann sogar zeigen:

Mitteilung 14.11 *BinPacking liegt in FPTAS$^\infty$.*

15 Approximationsklassen und Reduktionen

15.1 Erinnerung: Reduktion für P versus NP

In Kapitel 3 hatten wir den allgemeinen Begriff einer Turing-Reduktion (mit Polynomzeitbeschränkung) kennen gelernt. Aus der Grundvorlesung in Theoretischer Informatik dürfte ein abgeschwächter Begriff noch eher bekannt sein:

Definition 15.1 *Ein Entscheidungsproblem $\mathcal{P}_1$ heißt* Karp-reduzierbar *(oder many-one-reduzierbar) auf ein Entscheidungsproblem $\mathcal{P}_2$, wenn es einem (Polynomzeit-)Algorithmus R gibt, der eine Instanz x von $\mathcal{P}_1$ in eine Instanz y von $\mathcal{P}_2$ überführt in einer Weise, dass x eine Ja-Instanz von $\mathcal{P}_1$ ist und y eine Ja-Instanz von $\mathcal{P}_2$ ist.*

Zentral für die Entwicklung von P versus NP-Theorie ist es, Probleme zu kennen, die hart für NP sind in dem Sinne, dass ein deterministischer Polynomzeitalgorithmus für *ein* solches Problem die Existenz deterministischer Polynomzeitalgorithmen für *alle* NP-vollständigen Probleme nach sich ziehen würde. Wir wollen etwas Entsprechendes auch im Falle der Optimierungsprobleme entwickeln, müssen uns aber erst einmal an die wichtigsten Dinge aus der NP-Vollständigkeitstheorie erinnern, da die Verhältnisse dort einfacher sind.

Das „generische“ NP-vollständige Problem ist das Folgende:

Ggb: nichtdeterministische Turing-Maschine, Eingabe x von TM, Polynom p

Frage: Akzeptiert TM das Wort x in höchstens $p(|x|)$ Schritten?

Dieses Problem liegt in NP, und würde es in P liegen, so wäre P=NP.

Das vielleicht wichtigste NP-vollständige Problem ist das **Erfüllbarkeitsproblem (SAT)**:

Ggb: KNF Formel $\mathcal{F}$ auf einer Menge V von Booleschen Variablen.

Frage: Ist $\mathcal{F}$ erfüllbar? D.h., gibt es eine Variablenbelegung $f : V \to \{true, false\}$, die $\mathcal{F}$ „wahr macht“?

Satz 15.2 *Satz von Cook(-Levin) Das Erfüllbarkeitsproblem ist NP-vollständig.*

Zum Beweis verweisen wir auf andere Vorlesungen.
Die Beweisidee besteht in der formelmäßigen Darstellung des „Rechenteppichs“ einer Turing-Maschine.

Wir wollen den Karpschen Reduktionsbegriff an zwei Beispielen üben.

$\{0,1\}$-Lineares Programmieren

Ggb: Menge von Variablen $Z = \{z_1, \ldots, z_n\}$, die Werte aus $\{0,1\}$ annehmen können; Menge I von linearen Ungleichungen (mit Variablen aus Z und ganzzahligen Koeffizienten).

Frage: Hat I eine Lösung, d.h. irgendeine Variablenbelegung, die alle Ungleichungen erfüllt?

Lemma 15.3 *$\{0,1\}$-lineares Programmieren ist NP-hart.*

Beweis: Betrachte eine Instanz $x = (V, \mathcal{F})$ von SAT mit $V = \{x_1, \ldots, x_n\}$. Es sei $l_{j_1} \vee \ldots \vee l_{j_{n_j}}$ die j-te Klausel in $\mathcal{F}$. Als entsprechende Ungleichung sehen wir $\varrho_{j_1} + \ldots + \varrho_{j_{n_j}} \geq 1$ an mit $\varrho_{j_x} = z_i$, falls $l_{j_k} = x_i$, und $\varrho_{j_k} = (1 - z_i)$, falls $l_{j_k} = \overline{x_i}$. Dadurch ergibt sich eine $\{0,1\}$-LP-Instanz $y = (Z, I)$. Ist $f : V \to \{true, false\}$ eine Wahrheitsbelegung, so ist $f(\mathcal{F}) = true$ gdw. f' erfüllt alle Ungleichungen in I, wobei $f'(z_i) = 1$ gdw. $f(x_i) = true$. □

3SAT

Wie SAT, nur dass jede Klausel (höchstens) drei Literale enthält.

Lemma 15.4 *3SAT ist NP-hart.*

Beweis: Wir zeigen, wie allgemeine SAT-Formeln in (hinsichtlich der Erfüllbarkeit) äquivalente 3SAT-Formeln überführt werden können. Ist $l_{j_1} \vee \ldots \vee l_{j_{n_j}}$ eine Klausel mit $n_j > 3$, so kann durch Einführen von $n_j - 3$ Variablen $y_{j,1}, \ldots y_{j,n-3}$ und insgesamt $n_j - 2$ Klauseln die 3SAT-Restriktion erfüllt werden. Die Klauseln sehen dafür wie folgt aus:

$$(l_{j_1} \vee l_{j_2} \vee y_{j_1}), (\overline{y_{j_1}} \vee l_{j_3} \vee y_{j_2}), \ldots, (\overline{y_{j,n_j-4}} \vee l_{j_{n_j-2}} \vee y_{j,n_j-3}), (\overline{y_{j,n_j-3}} \vee l_{j_{n_j-1}} \vee l_{j_{n_j}})$$

□

15.2 Die Welt von NPO-Problemen

Betrachten wir zunächst die folgende, den Begriff eines r-approximativen Algorithmus nur verallgemeinernde Definition:

Definition 15.5 *Ist $\mathcal{P}$ ein NPO-Problem, $\mathcal{A}$ ein Approximationsalgorithmus für $\mathcal{P}$ und $r : \mathbb{N} \to (1, \infty)$ eine Abbildung, so heißt $\mathcal{A}$ $r(n)$-**Approximation**, falls für jede Instanz $x \in I_{\mathcal{P}}$ mit $S_{\mathcal{P}}(x) \neq \emptyset$ die Leistungsgüte der zulässigen Lösung $\mathcal{A}(x)$ der Ungleichung $R(x, \mathcal{A}) \leq (|x|)$ genügt.*

Das Verhalten des Algorithmus $\mathcal{A}$ ist bei Eingaben, die keine zulässige Lösungen haben, unbestimmt. Natürlich wird keine Lösung zurück geliefert.

Definition 15.6 *Ist $\mathcal{F}$ eine Klasse von Funktionen $f : \mathbb{N} \to (0, \infty)$ so bezeichnet $\mathcal{F}$-APX die Klasse der Probleme, für die ein $r(n)$-approximativer Polynomzeitalgorithmus (für ein $r \in \mathcal{F}$) existiert.*

Spezielle Funktionsklassen sind:

- LOG:= $O(\log(n))$
- POLY:= $\bigcup_{k>0} O(n^k)$
- EXP:= $\bigcup_{k>0} O(2^{n^k})$

Satz 15.7

$$PTAS \subseteq APX \subseteq LOG - APX \subseteq POLY - APX \subseteq EXP - APX \subseteq NPO.$$

Gilt vielleicht EXP − APX = NPO ?
Ein verführendes Argument ist das folgende:

Wegen der polynomiellen Schranke auf der Rechenzeit für die Maßfunktion $m_{\mathcal{P}}$ ist doch jedes NPO-Problem $\mathcal{P}$ $h \cdot 2^{n^k}$-approximierbar für geeignete h und k.
Es gibt eben Probleme, für die bereits die Frage, ob eine zulässige Lösung existiert, NP-hart ist. Dazu gibt es im Folgenden Beispiele.

Satz 15.8 *Wenn $P \neq NP$, so $EXP - APX \neq NPO$.*

Beweis: Betrachten wir das folgende NPO-Problem:
Minimum $\{0,1\}$-LP

$I =$ $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m, w \in \mathbb{N}^n$
$S:$ $x \in \{0,1\}^n$ mit $Ax \geq b$
$m:$ $\sum w_i x_i$ (Skalarprodukt von w und x)
opt : min.

Wäre Minimum $\{0,1\}$ − LP $\in$ EXP − APX, so könnten wir an dem Verhalten einer Polynomzeit-Approximation für eine Instanz x ablesen, ob die selbe Instanz x, aufgefasst als $\{0,1\}$-LP-Instanz, eine JA-Instanz ist oder nicht. Die Behauptung folgt mit Lemma 15.3. □

15.3 AP-Reduzierbarkeit

Bei Entscheidungsproblemen genügte es, einen Reduktionsbegriff von $\mathcal{P}_1$ auf $\mathcal{P}_2$ so zu definieren, dass man $\mathcal{P}_1$ „mit Hilfe von“ $\mathcal{P}_2$ lösen kann, was beim Karpschen Reduktionsbegriff bedeutet, dass Instanzen von $\mathcal{P}_1$ in Instanzen von $\mathcal{P}_2$ (in Polynomzeit) umgerechnet werden können. Dies genügt für einen Approximationsreduktionsbegriff nicht; vielmehr benötigen wir einen weiteren Algorithmus, der Lösungen von $\mathcal{P}_2$ in solche für $\mathcal{P}_1$ zurückrechnet, und letztere Rechnung sollte natürlich (in einem noch zu detaillierenden Sinne) die Näherungsgüte bewahren.

Schematisch können wir uns eine solche Approximationsreduktion wie in Abb. 7 vorstellen.

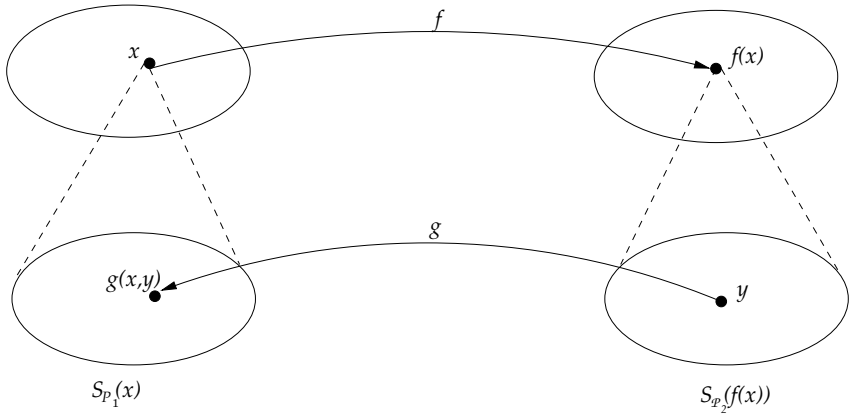


Abbildung 7: Schematische Approximationsreduktion

Approximationsgüteerhaltung am Beispiel:
Knotenüberdeckung $\rightsquigarrow$ MaxClique:

Ist $G = (V, E)$ ein Graph, so ist der **Komplementgraph** $G^c = (V, E^c)$ definiert durch $E^c = \{\{v_1, v_2\} \subseteq V \mid v_1 \neq v_2, \{v_1, v_2\} \notin E\}$.

Lemma 15.9 *$V' \subseteq V$ ist Knotenüberdeckung in G gdw, $V \setminus V'$ ist Clique in G^c.*

Beweis: Angenommen $V' \subseteq V$ ist Knotenüberdeckung. Gäbe es eine „Kante“ $\{u, v\} \notin E^c, u, v \in V \setminus V'$, so wäre $\{u, v\} \in E$ und $\{u, v\} \cap V = \emptyset$, also V' keine Überdeckung. Ist $V \setminus V'$ Clique, so betrachte eine Kante $\{u, v\}$ mit $\{u, v\} \cap V' = \emptyset$. Also ist $\{u, v\} \in V \setminus V'$, d. h. $\{u, v\} \in V \setminus V'$, d.h. $\{u, v\} \in E^c$. Kanten aus E sind also durch V' abgedeckt. □

Das Lemma 15.9 zeigt, dass das Knotenüberdeckungsproblem (Frage nach der Existenz einer Knotenüberdeckung mit höchstens k Knoten) auf das Cliquenproblem (Frage nach der Existenz einer Clique der Größe mindestens $|V| - k$) reduzieren lässt und umgekehrt (im Karpschen Sinne).
In obiger Notation haben wir (für beide Reduktionsrichtungen!):

$f(G) = G^c$ und, für $V' \subseteq V, g(G, V') = V \setminus V'$

Diese Approximationsreduktion erhält aber nicht die Approximationsgüte:
Betrachte die Graphenschar $(G_n)_{n\geq 1}$, wobei G_n aus zwei Cliquen mit jeweils n Knoten besteht, wobei der i-te Knoten der ersten Clique mit allen Knoten der zweiten Clique —mit Ausnahme des i-ten Knoten der zweiten Clique— verbunden ist. Jede maximale Clique von G_n enthält n Knoten.

Der Komplementgraph G_n^c besteht aus n disjunkten Paaren miteinander verbundener Knoten. Daher hat die triviale Lösung des MVC-Problems (man nehme alle Knoten als Knotenüberdeckung) eine Leistungsgüte von 2. Geht man zurück zum Ursprungsproblem, dem Cliquenproblem, so wäre die der MVC-Leistung „entsprechende“ Cliquenlösung die leere Menge.
Damit ist klar, dass die Näherungsgüte nicht erhalten bleibt bei dieser Reduktion.

Definition 15.10 *Betrachte $\mathcal{P}_1, \mathcal{P}_2 \in NPO$, $\mathcal{P}_1$ heiß **näherungserhaltend** auf $\mathcal{P}_2$ **reduzierbar**, kurz $\mathcal{P}_1$ ist **AP-reduzierbar** (AP bedeutet ausgeschrieben „approximation preserving“) auf $\mathcal{P}_2$, in Zeichen $\mathcal{P}_1 \leq_{AP} \mathcal{P}_2$, wenn es zwei Abbildungen f, g gibt und eine Konstante $\alpha \geq 1$ derart, dass folgende Bedingungen erfüllt sind:*

1. $\forall x \in I_{\mathcal{P}_1} \forall r \in \mathbb{Q} \cap (1, \infty) : f(x, r) \in I_{\mathcal{P}_2}$.
2. $\forall x \in I_{\mathcal{P}_1} \forall r \in \mathbb{Q} \cap (1, \infty) : S_{\mathcal{P}_1}(x) \neq \emptyset \rightarrow S_{\mathcal{P}_2}(x)(f(x, r)) \neq \emptyset$.
3. $\forall x \in I_{\mathcal{P}_1} \forall r \in \mathbb{Q} \cap (1, \infty) \forall y \in S_{\mathcal{P}_2}(f(x, r)) : g(x, y, r) \in S_{\mathcal{P}_1}(x)$.
4. *f, g sind durch Algorithmen $\mathcal{A}_f$, $\mathcal{A}_g$ berechenbar, deren Laufzeit polynomiell ist für jedes feste $r \in \mathbb{Q} \cap (1, \infty)$.*
5. $\forall x \in I_{\mathcal{P}_1} \forall r \in \mathbb{Q} \cap (1, \infty) \forall y \in S_{\mathcal{P}_2}(f(x, r)) :$

$$R_{\mathcal{P}_2}(f(x, r), y) \leq r \rightarrow R_{\mathcal{P}_1}(x, g(x, y, r)) \leq 1 + \alpha(r - 1)$$

Ein einfaches **Beispiel** für eine AP-Reduktion liefern MAXCLIQUE und MAX-IS durch Übergang auf den Komplementgraphen; die Clique wird so zur unabhängigen Menge.

Satz 15.11 *Betrachte $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \in NPO$.*

1. *Gilt $\mathcal{P}_1 \leq_{AP} \mathcal{P}_2$ und $\mathcal{P}_2 \leq_{AP} \mathcal{P}_3$, so auch $\mathcal{P}_1 \leq_{AP} \mathcal{P}_3$ (Transitivität)*
2. *Gilt $\mathcal{P}_1 \leq_{AP} \mathcal{P}_2$ und $\mathcal{P}_2 \in APX$, so folgt $\mathcal{P}_1 \in APX$.*
3. *Gilt $\mathcal{P}_1 \leq_{AP} \mathcal{P}_2$ und $\mathcal{P}_2 \in PTAS$, so folgt $\mathcal{P}_1 \in PTAS$.*

Beweis:

1. Ist intuitiv klar, wenn auch formal mühsam hinzuschreiben.

2. Sei (f, g, α) eine AP-Reduktion von $\mathcal{P}_1$ auf $\mathcal{P}_2$. Liegt $\mathcal{P}_2$ in APX und ist $\mathcal{A}_{\mathcal{P}_2}$ ein Algorithmus für $\mathcal{P}_2$ mit Leistungsgüte höchstens r, so ist

$$\mathcal{A}_{\mathcal{P}_1}(x) := g(x, \mathcal{A}_{\mathcal{P}_2}(f, (x, r)), r)$$

ein Polynomzeitalgorithmus der Leistungsgüte höchstens $1 + \alpha(r - 1)$.

3. Entsprechend überlegt man für Approximationsschemata, dass

$$\mathcal{A}_{\mathcal{P}_1}(x, r) = g(x, \mathcal{A}_{\mathcal{P}_2}(f(x, r'), r'), r')$$

mit $r' = 1 + (r-1)/\alpha$ ein Approximationsschema für $\mathcal{P}_1$ ist, sobald $\mathcal{A}_{\mathcal{P}_2}$ eines für $\mathcal{P}_2$ ist. □

Wegen Satz 15.11.1 ist die folgende Definition sinnvoll:

Definition 15.12 *Es sei $C \subseteq NPO$.*
Ein Problem $\mathcal{P}[\in NPO]$ heißt C-hart, wenn für jedes $\mathcal{P}' \in C$ gilt:

$$\mathcal{P}' \leq_{AP} \mathcal{P}.$$

*Ein C-hartes Problem heißt C-**vollständig**, wenn es in C liegt.*

Bemerkung 15.13 *In der Literatur werden verschiedene Reduktionsbegriffe für Approximationsprobleme betrachtet. Entsprechend gibt es auch verschiedene Härte- und Vollständigkeitsbegriffe. Näheres dazu im Buch von Ausiello et al., Kapitel 8!*

Im Folgenden werden wir noch einige konkrete AP-Vollständigkeitsbegriffe diskutieren. Dadurch wird auch der Umgang mit AP-Reduktionen geübt.

15.4 NPO-Vollständigkeit

Als (nahezu generische) NPO-vollständige Probleme betrachten wir:

a) MAXWSAT für Maximierungsprobleme aus NPO,

b) MINWSAT für Minimierungsprobleme aus NPO.

Konkreter: MAXWSAT (Maximum Weighted Satisfiability)

I : Boolesche Formeln φ mit Variablen $x_1, \ldots, x_n$ und nichtnegativen Gewichten $w_1, \ldots, w_n$
S : Belegung I der Variablen, sodass φ erfüllt wird.
m : $\max\{1, \sum_{i=1}^{n} w_i \tau(x_i)\}$; hierbei werden durch τ die Booleschen Werte *true* und *false* mit 1 und 0 identifiziert.
opt : max

MINWSAT ist das entsprechende Minimierungsproblem (opt = min).

Mitteilung 15.14 *a) MAXWSAT ist volländig für die Klasse der Maximierungsprobleme in NPO.*

b) MINWSAT ist vollständig für die Klasse der Minimierungsprobleme in NPO.

Der Beweis der Mitteilung ist analog zum Beweis des Satzes von Cook-Levin: Der Rechenteppich einer geeigneten Turingmaschine wird „logisch ausgedrückt".

Aus der Mitteilung alleine folgt *nicht*, dass MAXWSAT oder MINWSAT NPO-vollständig sind. Dies ergibt sich aber unmittelbar aus dem folgenden Satz.

Satz 15.15 *MAXWSAT und MINWSAT sind aufeinander AP-reduzierbar.*

Beweis: (Skizze) Wir beschreiben genauer eine Reduktion von MAXWSAT auf MINWSAT, die hinsichtlich Bedingung 5 *keine* AP-Reduktion ist, da das sich ergebende „α" von r abhängt, also nicht konstant ist. Danach deuten wir an, wie sich die Konstruktion als Spezialfall einer Schar von Reduktionen deuten lässt; mindestens eine Reduktion aus dieser Schar ist auch eine AP-Reduktion. In ähnlicher Weise kann man eine AP-Reduktion von MINWSAT auf MAXWSAT angeben.

Konstruktion einer „falschen" AP-Reduktion von MAXWSAT auf MINWSAT: Aus dem (nur angedeuteten) Beweis der vorigen Mitteilung ergibt sich, dass wir o.E. nur MAXWSAT-Instanzen mit Boolescher Formel betrachten müssen, die das Folgende erfüllen:

1. φ ist definiert über Variablen $v_1, \ldots, v_s$ mit Gewichten $w(v_i) = 2^{s-i}, i = 1, \ldots, s$ sowie über einigen anderen Variablen vom Gewicht Null.
2. Jede Belegung, die φ erfüllt, weist wenigstens einer der v_i den Wert *true* zu.

Es sei x eine solchermaßen eingeschränkte Instanz von MAXWSAT mit Boolescher Formel φ. Definiere:

$$f(x) := \varphi \wedge \alpha_1 \wedge \ldots \wedge \alpha_s \text{ mit } \alpha_i := (z_i \equiv (\overline{v_1} \wedge \ldots \wedge \overline{v_{i-1}}) \vee v_i));$$

z_i sind dabei neue Variablen mit $w(z_i) = 2^i, 1 \leq i \leq s$. Alle anderen Variablen haben Gewicht Null in der $f(x)$-Instanz.
Ist y eine erfüllende Belegung für $f(x)$, so sei $g(x, y)$ die Einschränkung von y auf die in φ vorkommenden Variablen.

Beachte: Genau eine der z_i-Variablen ist *true* in jeder erfüllenden Belegung von $f(x)$. Wäre keine der z_i-Variablen *true*, dann wären auch alle v_i-Variablen falsch, was 2. widerspricht. Nach Konstruktion der α_i sind aber keine zwei z_i-Variablen

wahr.
Also gilt für jede zulässige Lösung y von $f(x)$, dass $m(f(x),y) = 2^i$ für ein $1 \le i \le s$.

$$\begin{aligned} m(f(x),y) = 2^i \quad &\Leftrightarrow \quad z_i = 1 \Leftrightarrow v_1 = v_2 = \ldots v_{i-1} = 0 \wedge v_i = 1 \\ &\Leftrightarrow \quad 2^{s-i} \le m(x,g(x,y)) < 2 \cdot 2^{s-i} \\ &\rightsquigarrow \quad \frac{2^s}{m(f(x),y)} \le m(x,g(x,y)) < 2 \cdot \frac{2^s}{m(f(x),y)} \end{aligned}$$

für jede zulässige Lösung y von $f(x)$. $[*]$

Dies gilt natürlich auch für eine optimale Lösung y_f^* von $f(x)$.
Ist $\tilde{y}$ eine zulässige Lösung für x, also eine erfüllende Belegung von φ, so gibt es wegen 2) ein kleinstes i, für das v_i *true* ist. Durch $z_i = true$ und $z_j = false$ für $j \neq i$ lässt sich diese Belegung zu einer erfüllenden Belegung $\overline{\tilde{y}}$ von $f(x)$ erweitern. Einer optimalen Lösung $\tilde{y}^*$ von x entspricht so eine zulässige Lösung $\overline{\tilde{y}}^*$ von $f(x)$ mit der Eigenschaft $g(x,\overline{\tilde{y}}^*) = \tilde{y}^*$.

Für die Leistungsgüte von $g(x,y)$ ergibt sich:

$$\begin{aligned} R(x,(x,y)) \quad &= \quad \frac{m^*(x)}{m(x,g(x,y))} = \frac{m(x,\tilde{y}^*)}{m(x,g(x,y))} \overset{[*]}{<} \frac{2 \cdot \frac{2^s}{m(f(x),\overline{\tilde{y}}^*)}}{\frac{2^s}{m(f(x),y)}} \\ &\le \quad \frac{2 \cdot m(f(x),y)}{m^*(f(x))} = 2 \cdot R(f(x),y). \end{aligned}$$

Setzen wir diese Abschätzung in der letzten Bedingung der AP-Reduktions-Definition ein, so sehen wir, dass $\alpha = (2r-1)/(r-1)$ keine Konstante ist. Betrachte nun folgende Schar von Reduktionen:

$$f_k(x) := \varphi \wedge \bigwedge_{\substack{i=1,\ldots,s \\ b_1=0,1,\ldots,b_k=0,1}} \alpha_{i,b_1,\ldots,b_k}$$

mit

$$\alpha_{i,b_1,\ldots,b_k} = (z_{i,b_1,\ldots,b_k} \equiv (\overline{v_1} \wedge \ldots \wedge \overline{v_{i-1}} \wedge v_i \wedge (v_{i+1} \equiv b_1) \wedge \ldots \wedge (v_{i+k} \equiv b_k)))$$

(Falls $i + j > s$, entfallen die entsprechenden Bedingungen $v_{i+j} \equiv b_j$.)
Dafür sind $z_{i,b_1,\ldots,b_k}$ $2^k \cdot s$ viele neue Variablen.
Wie oben sind nur die z-Variablen solche mit nicht-verschwindendem Gewicht. Wir setzen hierbei

$$w(z_{i,b_1,\ldots,b_k}) = \left\lceil \frac{c \cdot 2^s}{w(v_i) + \sum_{j=1}^{k} b_j w(v_{i+j})} \right\rceil$$

für eine genügend große Konstante c.

Nach einiger (hier fortgelassener) Rechnung findet man

$$\frac{c \cdot 2^s}{m(f_k(x),y)} \le m(x,g(x,y)) < \frac{c \cdot 2^s}{m(f_k(x),y)} \cdot (1 + 2^{-k})$$

Dabei ist $g(x,y)$ wieder durch „Vergessen“ der z-Belegung definiert.
Wie zuvor erhält man somit

$$R(x,g(x,y)) < (1 + 2^{-k}) R(f_k(x),y).$$

Unsere zuvor durchgeführte Rechnung entspricht dem Spezialfall $k = 0$. Ist nun $r > 1$ vorgegeben, so wählen wir k so, dass $2^{-k} \leq (r-1)/r$.
Dann folgt aus $R(f_k(x), y) \leq r$ nämlich

$$R(f_k(x), y) < (1 + 2^{-k})R(f_k(x), y) \leq r + r2^{-k} \leq r + r - 1 = 1 + 2(r-1).$$

Mit $f(x, r) := f_{k_r}(x)$ ist $(f, g, 2)$ eine AP-Reduktion von MAXWSAT auf MINW-SAT. □

Folgerung 15.16 *Maximum Weighted 3-SAT ist NPO-vollständig.*

Beweis: Die Überführung in KNF ist in Polynomzeit möglich, ansonsten betrachte den Beweis von Lemma 15.4. □

Analog sieht man

Folgerung 15.17 *Minimum Weighted 3-SAT ist NPO-vollständig.* □

Folgerung 15.18 *Minimum* $\{0, 1\}$*-LP ist NPO-vollständig.*

Beweis: Kombiniere Folgerung 15.17 und (den Beweis von) Lemma 15.3. □

15.5 EXP-APX-Vollständigkeit

Der Unterschied zwischen EXP-APX und NPO besteht darin, dass für NPO-Probleme $\mathcal{P}$ die Menge $S_{\mathcal{P}}(x)$ für jede Instanz x NP-hart sein darf (und in den vorigen Beispielen für NPO-vollständige Probleme auch war). Dadurch, dass man „künstlich“ die Menge der zulässigen Lösungen trivialisiert, ohne die optimalen Lösugen zu verändern, kann man sich Probleme definieren, für die man EXP-APX-Vollständigkeit beweisen kann.

Bemerkung 15.19 *Der Nachteil der hier betrachteten NPO- bzw. der daraus abgeleiteten EXP-APX-vollständigen Probleme ist ihre „Künstlichkeit“.*
Wir werden abschließend APX-vollständige „natürliche“ Probleme kennen lernen.

15.6 APX-Vollständigkeit

Mit Hilfe der uns hier nicht zur Verfügung stehenden PCP-Kennzeichnung von NP lässt sich zeigen (s. das Buch von Ausiello et al.).

Mitteilung 15.20 *MAX 3-SAT ist vollständig für die Klasse der Maximierungsprobleme in APX.*

Daraus folgt zusammen mit dem folgenden Satz, dass MAX 3-SAT tatsächlich APX-vollständig ist.

Satz 15.21 *Zu jedem Minimierungsproblem $\mathcal{P} \in APX$ gibt es ein Maximierungsproblem $\mathcal{P}' \in APX$ mit $\mathcal{P} \leq_{AP} \mathcal{P}'$.*

Beweis: (Skizze) Es sei $\mathcal{A}$ eine r-Approximation von $\mathcal{P}$. Definiere $\mathcal{P}'$ wie $\mathcal{P}$ bis auf die Maßfunktion, die wie folgt definiert ist:

$$m_{\mathcal{P}'}(x,y) = \begin{cases} (\lceil r \rceil m_{\mathcal{P}}(x,\mathcal{A}(x)) - (\lceil r \rceil \cdot m_{\mathcal{P}}(x,y)), & m_{\mathcal{P}}(x,y) \leq m_{\mathcal{P}}(x,\mathcal{A}(x)) \\ m_{\mathcal{P}}(x,\mathcal{A}(x)), & \text{sonst} \end{cases}$$

$$\rightsquigarrow m_{\mathcal{P}}(x,\mathcal{A}(x)) \leq m^*_{\mathcal{P}'}(x) \leq (\lceil r \rceil m_{\mathcal{P}}(x,\mathcal{A}(x))$$

$\rightarrow \mathcal{A}$ ist $(\lceil r \rceil + 1)$-Approximation für das Maximierungsproblem $\mathcal{P}'$.

Die AP-Reduktion ist wie folgt definiert:

1. Für jede Instanz x setze $f(x) = x$.
2. Für jede Instanz x und für jede Lösung y von $f(x)$:

$$g(x,y) = \begin{cases} y, & m_{\mathcal{P}}(x,y) \leq m_{\mathcal{P}}(x,\mathcal{A}(x)) \\ \mathcal{A}(x), & \text{sonst} \end{cases}$$

3. $\alpha = \lceil r \rceil + 1$.

Um nachzuweisen, dass diese Reduktion wirklich eine AP-Reduktion ist, betrachte eine Lösung y mit $R'_{\mathcal{P}}, (x,y) = m^*_{\mathcal{P}'}(x)/m_{\mathcal{P}}(x,y) \leq r'$.
Zu zeigen ist:

$$R_{\mathcal{P}}(x, g(x,y)) \leq 1 + \alpha(r' - 1)$$

Dies kann durch Unterscheidung der Fälle $m_{\mathcal{P}}(x,y) \leq m_{\mathcal{P}}(x,\mathcal{A}(x))$ bzw. $m_{\mathcal{P}}(x,y) > m_{\mathcal{P}}(x,\mathcal{A}(x))$ durch einige Abschätzungen hergeleitet werden. □

L-Reduktionen—Motivation Wie kann man nun APX-Vollständigkeit für „natürliche“ Probleme beweisen? Hierbei ist es manchmal hilfreich, sich „alte“ bekannte NP-Vollständigkeitsbeweise für die betreffende Entscheidungsprobleme anzusehen. So wird für das Kantenschnittproblem üblicherweise (siehe das Buch von Ausiello et al.) der Weg von 3-SAT über MAX2SAT_D, NOT-ALL-EQUAL-3SAT zu MAXCUT_D gegangen.
Hierbei ist es häufig hilfreich, eine andere als die bislang diskutierte AP-Reduktion zu betrachten, da sich dabei NP-Reduktionen leichter übertragen:

Definition 15.22 *Es seien $\mathcal{P}_1$ und $\mathcal{P}_2$ zwei NPO-Probleme. $\mathcal{P}_1$ heißt **L-reduzierbar** auf $\mathcal{P}_2$, i. Z. $\mathcal{P}_1 \leq_L \mathcal{P}_2$, wenn zwei Funktionen f, g und zwei positive Konstanten β, γ existieren, sodass:*

1. *$\forall x \in I_{\mathcal{P}_1} : f(x) \in I_{\mathcal{P}_2}$ ist in Polynomzeit berechenbar.*

2. $\forall x \in I_{\mathcal{P}_1} : S_{\mathcal{P}_1}(x) \neq \emptyset \rightarrow S_{\mathcal{P}_2}(f(x)) \neq \emptyset$.
3. $\forall x \in I_{\mathcal{P}_1} \forall y \in S_{\mathcal{P}_1}(f(x)) : g(x,y) \in S_{\mathcal{P}_1}(x)$ *ist in Polynomzeit berechenbar.*
4. $\forall x \in I_{\mathcal{P}_1} : m^*_{\mathcal{P}_2}(f(x)) \leq \beta m^*_{\mathcal{P}_1}(x)$.
5. $\forall x \in I_{\mathcal{P}_1} \forall y \in s_{\mathcal{P}_2}(f(x)) :$

$$\left|m^*_{\mathcal{P}_1}(x) - m_{\mathcal{P}_1}(x, g(x,y))\right| \leq \gamma \cdot \left|m^*_{\mathcal{P}_2}(x) - m^*_{\mathcal{P}_2}(f(x), y)\right|$$

(f, g, β, γ) *heißt auch* ***L-Reduktion*** *von* $\mathcal{P}_1$ *auf* $\mathcal{P}_2$.

Mitteilung 15.23 *Es seien* $\mathcal{P}_1, \mathcal{P}_2 \in NPO$ *mit* $\mathcal{P}_1 \leq_L \mathcal{P}_2$. $\mathcal{P}_1 \in APX \rightarrow \mathcal{P}_1 \leq_{AP} \mathcal{P}_2$.

Als eine Anwendung und erstes Glied obiger Reduktionskette zeigen wir:

Satz 15.24 *MAX3SAT* $\leq_L$ *MAX2SAT.*

Beweis: Wir geben einer L-Reduktion $(f, g, 13, 1)$ an. Es sei φ eine Instanz von MAX3SAT mit m Klauseln $a_i \vee b_i \vee c_i, 1 \leq i \leq m$, wobei a_i, b_i und c_i Literale sind. Jeder dieser Klauseln ordnen wir die folgenden zehn neuen Klauseln zu (jede hat höchstens zwei Literale!):

$(*)$	a_i,	b_i,	c_i,	d_i,	$\overline{a_i} \vee \overline{b_i}$,	$\overline{a_i} \vee \overline{c_i}$,	$\overline{b_i} \vee \overline{c_i}$,	$a_i \vee \overline{d_i}$,	$b_i \vee \overline{d_i}$,	$c_i \vee \overline{d_i}$
	1	2	3	4	5	6	7	8	9	10

Dabei ist d_i eine neue Variable. $f(\varphi) = \varphi'$ sei die sich so ergebende Instanz von MAX2SAT.
Die folgende Tabelle zeigt, dass jede Belegung höchstens 7 der 10 Klauseln einer Gruppe der Form $(*)$ erfüllt (symmetrische Fälle sind nicht aufgeführt)

a_i	b_i	c_i	d_i	1	2	3	4	5	6	7	8	9	10	$\sum$
0	0	0	0					1	1	1	1	1	1	6
0	0	0	1				1	1	1	1				4
0	0	1	0			1		1	1	1	1	1	1	7←
0	0	1	1			1	1	1	1	1			1	6
0	1	1	0		1	1		1	1		1	1	1	7←
0	1	1	1		1	1	1	1	1			1	1	7←
1	1	1	0	1	1	1					1	1	1	6
1	1	1	1	1	1	1	1				1	1	1	7←

Für jede erfüllende Belegung der ursprünglichen 3-SAT-Klausel $a_i \vee b_i \vee c_i$ gibt es eine Belegung von $(*)$, die sieben Klauseln erfüllt; diese sind durch Pfeile in der Tabelle markiert. Wegen der ersten Zeile der Tabelle gibt es im Falle einer $a_i \vee b_i \vee c_i$ nicht-erfüllenden Belegung eine Belegung von d_i, die 6 Klauseln von $(*)$ erfüllt. Somit können wir schließen:

$$m^*(\varphi') = 6m + m^*(\varphi) \leq 12m^*(\varphi) + m^*(\varphi) = 13m^*(\varphi).$$

Die Ungleichung gilt, da zu einer KNF-Formel stets eine Belegung existiert, die wenigstens die Hälfte der Klausel erfüllt (siehe Analyse der Greedy-Heuristik für MAXSAT in Satz 11.9).

Ist τ' eine Belegung der Variablen von φ', so bezeichne $\tau = g(\varphi, \tau')$ die Restriktion von τ' auf die in φ vorkommenden Variablen. Durch Betrachten obiger Tabelle ergibt sich

$$m(\varphi, \tau) \geq m(\varphi', \tau') - 6m.$$

Damit haben wir:

$$\begin{aligned} |m^*(\varphi) - m(\varphi, g(\varphi, \tau'))| &= m^*(\varphi) - m(\varphi, \tau) \\ &= m^*(\varphi') - 6m - m(\varphi, \tau) \\ &\leq m^*(\varphi') - 6m - m(\varphi', \tau') + 6m \\ &= |m^*(\varphi') - m(\varphi', \tau')| \,. \square \end{aligned}$$

15.7 Facetten der Härte für Approximationen

Diese Darstellung folgt:
D.S. Johnson: The NP-Completeness Column: The Many Limits on Approximation. ACM Trans. Alg., Band 2, S. 473–489, 2006.

Der "Goldstandard" für Härteresultate:
Ist betrachtetes Optimierungsproblem "leicht" (das hängt von der konkreten Klasse ab, die man betrachtet), so folgt P = NP.

Beispiel 15.25 *Dinur und Safra konnten 2002 (ECCC, dann SODA 2005) zeigen: Falls $P \neq NP$, so lässt sich das Knotenüberdeckungsproblem nicht besser als* 1.36067 *approximieren.*

Diese Aussage lässt noch eine beträchtliche Lücke, denn das beste bekannte positive Approximationsergebnis stammt von Karakostas (ICALP 2005):
Das Knotenüberdeckungsproblem für n-Knotengraphen lässt sich in der Leistungsgüte $2 - \Theta(\frac{1}{\sqrt{\log(n)}})$ *approximieren.*
Unter schwächeren Annahmen lassen sich manchmal stärkere Aussagen gewinnen.

Intuitive Ähnlichkeit von Problemen

Bekannt: Das Cliquenproblem.
Feige et al. konnte 2006 zeigen (J.ACM Band 43, Seiten 268–292):

Mitteilung 15.26 *MAXCLIQUE lässt sich nicht mit einem Faktor* $2^{(\log(n))^{1-\epsilon}}$ *approximieren für irgendein $\epsilon > 0$, falls nicht $NP \subseteq DTIME(n^{polylog(n)})$.*

Unter stärkeren Voraussetzungen (Goldstandard) kann man eine schwächere Aussage zeigen:

Mitteilung 15.27 *Gilt $P \neq NP$, so lässt sich MAXCLIQUE für kein $\epsilon > 0$ auf einen Faktor von $n^{1-\epsilon}$ approximieren.*

Gelten ähnliche Schranken für ähnliche Probleme ?
Leider nicht immer...

Sehr ähnlich zu dem Problem des Findens großer Cliquen scheint doch das Problem zu sein, möglichst große ausgeglichene Bicliquen zu finden. Dies bedeutet genauer:

Ein vollständiger bipartiter Graph $K_{k,\ell}$ heißt auch *Biclique*. Eine Biclique heißt *ausgeglichen*, falls auf beiden Seiten der Bipartition gleichviele Knoten liegen.

Das Problem, möglichst große ausgeglichene Bicliquen in einem (nicht notwendig bipartiten) Graphen zu finden, ist bekanntermaßen NP-hart (auch bekannt als BCBS: balanced complete bipartite subgraph).

BCBS-Approximation

Oft begegnen wir Approximationsschranken, die randomisierte Komplexitätsklassen betreffen.
BP steht für "bounded error probabilistic".
Details zu dieser und vielen anderen Komplexitätsklassen finden Sie unter Komplexitäts-Wiki am Caltech.

Mitteilung 15.28 *BCBS lässt sich nicht mit einem Faktor n^δ approximieren für ein bestimmtes $\delta > 0$, falls nicht $NP \subseteq \bigcap_{\epsilon>0} BPTIME(2^{n^\epsilon})$.*

Eine ähnliche Approximationsschranke konnte Feige 2002 unter einer anderen Annahme zeigen, nämlich unter der sogenannten *Durchschnittsfallhärte* von 3SAT. Auch hier kommt aber wieder eine Randomisierung ins Spiel.

Label Cover: ein hübsches Hilfsproblem

Eine Instanz wird beschrieben durch:
einen bipartiten Graphen $G = (V_1, V_2; E)$, $E \subseteq V_1 \times V_2$,
natürliche Zahlen N, M, B,
Abbildungen $\pi_{u,v} : \{1, \ldots, M\} \to \{1, \ldots, N\}$ für jede Kante $(u, v) \in E$.
Frage: Gibt es Knotenmarkierungen $L_1 : V_1 \to \{1, \ldots, M\}$ und $L_2 : V_2 \to \{1, \ldots, N\}$, sodass mindestens B Kanten *abgedeckt* sind ?
Hierbei ist eine Kante (u, v) abgedeckt durch die Markierung, falls $\pi_{u,v}(L_1(u)) = L_2(v)$.

Arora und Lund konnten 1997 zeigen, dass die Maximierungsvariante von Label Cover hart zu approximieren ist für jeden konstanten Faktor, wenn nicht P = NP.
Arora und Lund konnten genauer folgendes zeigen:
Zu jedem $0,5 > \epsilon > 0$ gibt es eine Konstante $k(\epsilon)$, sodass für alle Instanzen von Label Cover mit $M, N \leq k(\epsilon)$ die nachstehend beschriebenen zwei Fälle NP-hart zu unterscheiden sind:
(1) Es gibt eine Markierung, die alle Kanten abdeckt.
(2) Es gibt keine Markierung, die mehr als $\epsilon \cdot |E|$ viele Kanten abdeckt.

Die so entstandene Lücke wird im Beweis der in der vorigen Folie gezeigten Behauptung verwendet.endslide

Unique Game Conjecture (UGC) ist ein seltsamer Name für eine Einschränkung von Label Cover; tatsächlich wurden Label Cover und die Unique Game Conjecture zunächt in Termini von Spielen definiert.

Vermutung (UGC): Für alle $\epsilon, \delta \in (0; 1/2)$ gibt es eine Konstante $k(\epsilon, \delta)$, sodass für Label Cover Instanzen mit $M = N = k(\epsilon, \delta)$, in denen darüber hinaus die Abbildungen $\pi_{u,v}$ sämtlich Permutationen sind, es keinen Polynomzeit-Algorithmus gibt, der unterscheiden kann zwischen
(1) Es gibt eine Markierung, die wenigstens $(1 - \delta)|E|$ viele Kanten abdeckt und
(2) Es gibt keine Markierung, die mehr als $\epsilon \cdot |E|$ viele Kanten abdeckt.

Viele Forscher bezweifeln (noch) die Gültigkeit der UGC.

Konsequenzen der UGC: Khot und Regev konnten 2003 zeigen:

Mitteilung 15.29 *Gilt die UGC, so gibt es keinen Algorithmus, der das Knotenüberdeckungsproblem auf einen konstanten Faktor besser als* 2 *approximiert.*

Noch überraschender ist das Ergebnis für MaxCut:
Hierbei wird für einen gegebenen Graphen $G = (V, E)$ nach einer Knotenmenge S gesucht, die die Zahl der Kanten mit genau einem Endpunkt in S maximiert.
Goemans und Williamson haben 1995 einen (komplizierten) α-Faktor-Approximationsalgorithmus gefunden mit $\alpha = \min_{0<\theta\leq\pi}(\pi(1 - \cos(\theta))/(2\theta))$.
Unter der Annahme der UGC ist dieser Faktor bestmöglich!

Die Approximationstheorie lässt noch viele weiter hübsche Ergebnisse und auch manche Überraschungen in den nächsten Jahren erwarten. Bleiben Sie am Ball !