Schließlich: $l_{CH} \leq l(T) + l(M) \leq l_{OPT} + \frac{l_{OPT}}{2} = \frac{3}{2} l_{OPT}$

$\square$

Laufzeit: $O(n^3)$ (Die Berechnung eines perfekten Matchings mit minimalem Gewicht braucht $\Theta(|U|^3) = O(n^3)$ Zeit und dies dominiert die Laufzeit; die anderen Schritte sind linear.) $\to O(n)$

[Bemerkung: es wurde gezeigt (2011) dass mit einer geschickten Wahl des Spannbaums ein kleinerer Approximationsfaktor erreichbar ist, wenn die Metrik ~~durch~~ die Distanzen in einem Graphen definiert wird.]

Die folgenden beiden (greedy) Heuristiken haben schlechteren Approximationsfaktor im Worst-Case als Christofides.

iii.) Nearest-Insertion Heuristik 2-approximativ

- sei $V = \{1, 2, \dots, n\}$
- sei $\{i, j\}$ ein Paar von Orten mit ~~minimaler~~ maximaler Distanz $d(i, j)$
- setze $V := V \setminus \{i, j\}$
- sei die (partielle) Rundreise $R = (i, j, i)$
- REPEAT
  - nimm einen ~~Knoten~~ Ort $k \in V$ mit ~~kürzestem~~ weitestem Abstand zu einem Ort in der aktuellen partiellen Rundreise R
    sei $V := V \setminus \{k\}$
  - füge k zwischen zwei benachbarten Orten in der partiellen Rundreise ein, so dass dies den kleinsten Anstieg der Länge der Rundreise resultiert

  UNTIL $V = \emptyset$

iv.) Farthest-Insertion Heuristik $6.5 \leq$ Approx-Faktor $\leq \log n + 1$

G 28.

In der Euklidschen Ebene $\mathbb{R}^2$ mit den gewöhnlichen (Euklidschen) Distanzen, fängt Nearest-Insertion und Farthest-Insertion Heuristik mit einer konvexen Hülle aller Orten als partieller Rundreise R an.

Durchschnittlicher Approx-Faktor im Experiment mit je 100000 zufälligen Punkten $\mathbb{R}^2$

Christofides :	$\alpha \sim 1.1$
Farthest-Insertion:	1.1
Nearest-Insertion :	$\alpha \sim 1.25$
Spannbaum :	$\alpha \sim 1.4$

3. Das Euklidsche TSP

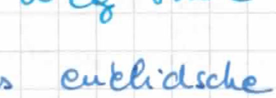
Eingabe: n Punkte $V \subseteq \mathbb{R}^d$ mit den euklidschen Distanzen

$$d(a,b) = \sqrt{(a_1-b_1)^2 + (a_2-b_2)^2 + \dots + (a_d-b_d)^2}$$

für jede $a = (a_1\ a_2 \dots a_d) \in V$
$b = (b_1\ b_2 \dots b_d)$

Ausgabe: eine kürzeste Rundreise die alle n Punkte genau einmal besucht.

(Bemerkung:



(a, b, c) und zurück zum a zählt auch als Rundreise, b wird nicht nochmal "besucht" wenn er auf dem kürzesten Weg von c nach a liegt)

Theorem: Für das euklidsche TSP gibt es ein PTAS (Arora's PTAS), also dieser Spezialfall des TSP Problems ist beliebig nah approximierbar. (siehe später)

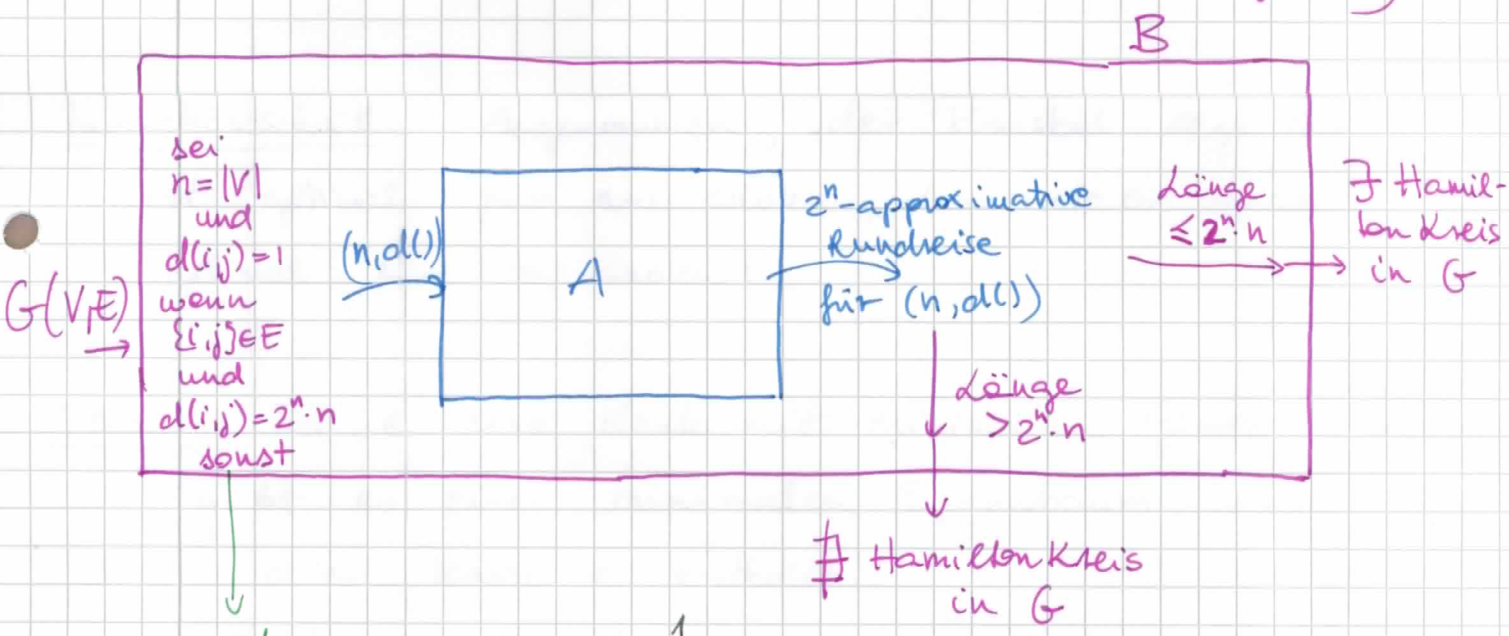
Travelling - Saleman - Problem

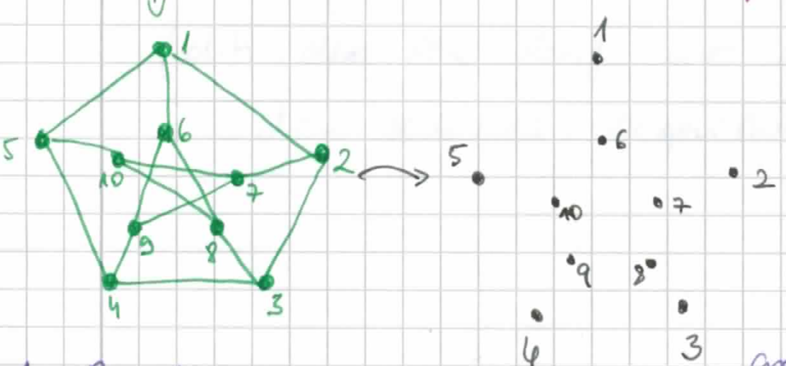
Thm: Das allgemeine TSP ist polynomiell nicht approximierbar um irgendeinen vernünftigen Faktor, zB. um Faktor $\alpha(n) = O(2^n)$.

Wir zeigen den Beweis für $\alpha(n) = 2^n$

→ Nehmen wir das Gegenteil an, dass es einen polynomiellen Algorithmus A gibt, der für jede Instanz $I = (n, d())$ von TSP, eine $\leq 2^n \cdot OPT(I)$ lange Rundreise ausgibt.

→ Wir zeigen, dass in diesem Fall ein Algorithmus B das HAMILTONSCHER KREIS Problem in polynomieller Laufzeit entscheiden kann (Angenommen $P \neq NP$, ist das ein Widerspruch, weil HAMILTONSCHER KREIS NP-vollständig ist)





$d(1,2) = 1$

$d(1,3) = 10 \cdot 2^{10}$

usw.

Jede Rundreise ist n-lang oder ~~mindestens~~ größer als $2^n \cdot n$ lang!

f 18.

es gibt Hamilton Kreis in $G(V,E)$ $\Leftrightarrow$ $\exists$ Rundreise mit Länge n für $(n, d())$ $\Leftrightarrow$ A muss eine RR. mit Länge $\leq 2^n \cdot n$ ausgeben (eine n-lange RR. eigentlich)

es gibt <u>keinen</u> Hamilton Kreis in $G(V,E)$ $\Leftrightarrow$ jede Rundreise für $(n, d())$ hat Länge $> 2^n \cdot n$ $\Leftrightarrow$ A gibt eine solche aus

Wir haben HAMILTONSCHER-KREIS auf die 2^n-Approximierung von TSP reduziert.

Zusammenfassung:

das allgemeine TSP — nicht mal $O(2^n)$-approximierbar

Spezialfall:
das metrische TSP — $\frac{3}{2}$-approximierbar; nicht $\frac{220}{219}$-approximierbar

noch spezieller Fall:
das euklidische TSP — $(1+\varepsilon)$-approximierbar; Arora's PTAS

d.) Das BINPACKING Problem

Eingabe: n Objekte mit Gewichten $g_1 g_2 \dots g_n$
$(0 \le g_i \le 1)$

Ausgabe: Verteile die Objekte in eine minimale Anzahl von Behälter (Bins) mit Kapazität 1

ähnlich zum SCHEDULING, nur hier ist die Anzahl der Behälter ("Maschinen") zu optimieren, und wir können die Kapazität = 1 der Behälter nicht erweitern.

G30.

Greedy Algorithmen für BIN PACKING

1. Next-Fit

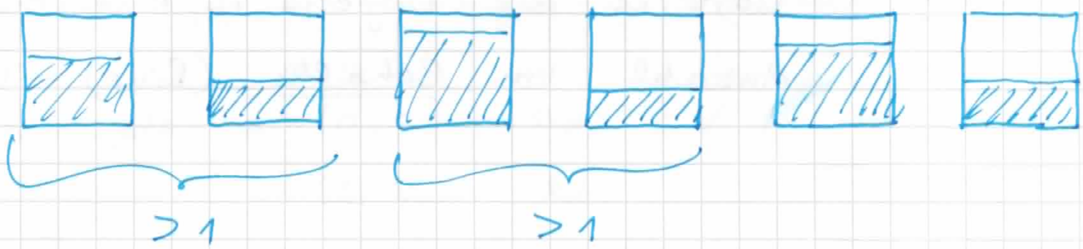
- setze $B := 1$ (Anzahl der geöffneten Behälter)
- FOR $i = 1$ TO n DO

  falls möglich, füge g_i in Behälter B ein;

  sonst $B := B+1$ und füge g_i in den neu geöffneten Behälter ein;

Behauptung: Next Fit benutzt höchstens $2\,OPT$ Behälter.
(sogar $\leq 2\,OPT - 1$)

Warum?



Je zwei aufeinanderfolgende Behälter enthalten Gesamtgewicht echt größer als 1. Deshalb:

Für B gerade und $G = \sum_{i=1}^{n} g_i$ gilt

Bin Kapazität, opt. Anzahl Bins

$$1 \cdot OPT \geq G > \frac{B \cdot 1}{2} \implies OPT \geq \frac{B}{2} + 1$$

Für B ungerade gilt

$$OPT \geq G > \frac{B-1}{2} \implies OPT \geq \frac{B-1}{2} + 1$$

Somit: $B \leq 2 \cdot OPT - 1$ in beiden Fällen

(da die optimale Anzahl der Behälter OPT ganzzahlig ist). □

2. First Fit

$\lfloor 1.7\, \mathrm{OPT} \rfloor$ (approx)

- setze $B = 1$
- FOR $i = 1$ TO n DO

  füge g_i in den ersten Behälter ein wo es reinpasst, wenn es in keinen geöffneten Behälter passt, dann öffne einen neuen Behälter $B := B+1$ und füge g_i ein.

Die ersten beiden waren online Algorithmen:
wir fügen Objekt i in die Behälter, ohne Objekte $i+1, i+2, \dots, n$ zu kennen.
Erreichen wir bessere Approximation mit offline Strategien — wenn wir die ganze Instanz am Anfang schon kennen, und z.B. sortieren dürfen?

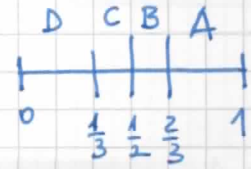
3. First Fit Decreasing (FFD)

- sortiere die Objekte nach absteigendem Gewicht
- wende First Fit für die sortierte Eingabe an

$$g_1 \geq g_2 \geq g_3 \geq \dots \geq g_n$$

Theorem: FFD benutzt maximal $\frac{3}{2}\mathrm{OPT} + 1$ Behälter.

Beweis:



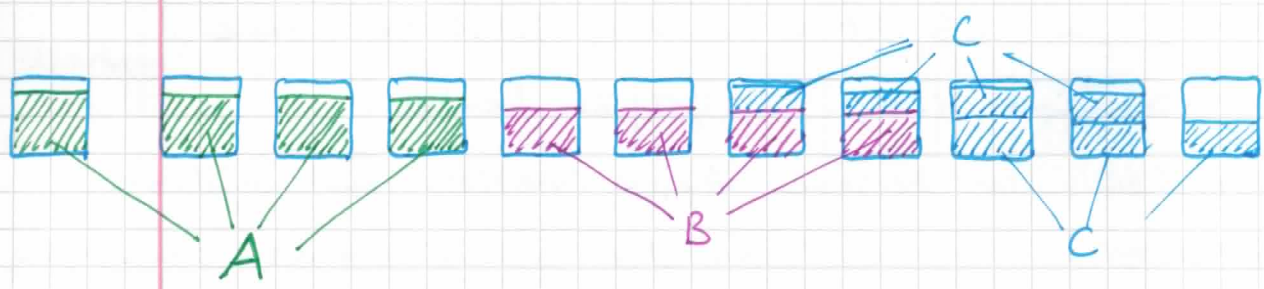
$A = \{ i \mid \frac{2}{3} < g_i \}$ (die Objekte mit Gewicht $> \frac{2}{3}$)

$B = \{ i \mid \frac{1}{2} < g_i \leq \frac{2}{3} \}$

$C = \{ i \mid \frac{1}{3} < g_i \leq \frac{1}{2} \}$

$D = \{ i \mid g_i \leq \frac{1}{3} \}$

Wie geht FFD mit den Objektmengen $A \cup B \cup C$ um?



FFD ist optimal auf den Objekten (nur) in $A \cup B \cup C$

(FFD ist äquivalent mit BFD über $A \cup B \cup C$ $\longrightarrow$)

Warum?

- Objekte in A und B brauchen in jedem Fall neue Behälter
- manche C-Objekte passen über ein B-Objekt
- C-Objekte die über kein noch allein stehendes B-Objekt passen, passen genau zu zweit in einen Behälter Also:

wir wollen so viele C-Objekte über B-Objekte packen wie nur möglich:

FFD ist ~~die bestmögliche~~ eine optimale Methode C-Objekte mit B-Objekte zu matchen: es kann (durch ein Vertausch-Argument) angenommen werden, dass in OPT das größte, zu B-passende C-Objekt in der kleinstmöglichen Lücke ist, und man kann iterativ (oder mit Induktion) weiter argumentieren.

□

– D-Objekte haben Größe $< \frac{1}{3}$

Fall 1: Wenn für D-Objekte keine weitere Behälter geöffnet werden, dann ist die Verteilung der Objekte weiterhin optimal $FFD(I) \leq OPT(I)$

Fall 2: wenn weitere Behälter für D-Objekte geöffnet wurden, dann sind alle Behälter bis auf den letzten bis über $\frac{2}{3}$ voll. Deshalb

?

$$OPT(I) \geq \sum_{i=1}^{n} g_i > (FFD(I) - 1)\frac{2}{3}$$

$$\frac{3}{2} OPT(I) + 1 > FFD(I)$$

□

Theorem: FFD benutzt sogar höchstens $\frac{11}{9} OPT + 2$ Behälter.

(ohne Beweis)

4. Best-Fit Decreasing (mindestens so gut wie FFD)

- sortiere die Objekte nach absteigendem Gewicht

$$g_1 \geq g_2 \geq g_3 \geq \ldots \geq g_n$$

- FOR $i = 1$ TO n DO

  füge Objekt i in den Behälter ein wo es noch möglich aber am knappsten ist (dessen verbleibende Restkapazität nach dem Einfügen minimal ist)

G34.

Nichtapproximierbarkeit des BIN-PACKING Problems

Theorem: BIN-PACKING besitzt keinen (effizienten) c-approximativen Algorithmus für $c < \frac{3}{2}$ (angenommen $P \neq NP$)

(Überlegungen:

Im BIN-PACKING die Anzahl der benutzten Behälter ist zu minimieren.

- Ist es für viele, oder für wenige Behälter (in OPT) schwieriger, eine $< \frac{3}{2}$ - Approximation zu erreichen?

- Was bedeutet für BIN-PACKING eine besser als $\frac{3}{2}$ Approximation?

  - wenn 200 Behälter ausreichen
    $\Rightarrow$ der Alg findet < 300
  - wenn 6 —"— reichen
    $\Rightarrow$ er findet < 9 ≤ 8
  - wenn 4 —"— reichen
    $\Rightarrow$ er findet < 6 ≤ 5
  - wenn 2 reichen
    $\Rightarrow$ er findet $< 3 = 2$ Behälter

Das letzte Problem ist NP-schwer: es ist nämlich "äquivalent" mit dem PARTITION Problem (fast))

PARTITION: für gegebene Zahlen $x_1 x_2 \dots x_n$ entscheide ob sie in zwei Mengen partitioniert werden können, s.d. beide Mengen die Summe

$$\frac{\sum_{i=1}^{n} x_i}{2}$$ haben.

→ wir konstruieren eine Instanz für BIN-PACKING: wir normieren die Zahlen (teilen durch $\frac{\sum_{i=1}^{n} x_i}{2}$) so dass die neuen Zahlen die Summe $=2$ haben: sei $g_i = \frac{2 \cdot x_i}{\sum_{i=1}^{n} x_i}$ $(i = 1, 2, \dots, n)$

→ Falls $x_1 \dots x_n$ eine JA-Instanz für PARTITION ist, dann passen die Gewichte in 2 Behälter, und ein $\left(\frac{3}{2} - \varepsilon\right)$-approximativer Algorithmus für BIN-PACKING sollte sie in weniger als 3, also in 2 Behälter verteilen können.

→ Falls $x_1 \dots x_n$ eine NEIN-Instanz für PARTITION ist, dann gibt der $\left(\frac{3}{2} - \varepsilon\right)$-approximativer Algorithmus auch mindestens 3 Behälter als Lösung für $g_1 g_2 \dots g_n$ aus (weil $OPT \geq 3$).

→ Somit könnte ein effizienter $\left(\frac{3}{2} - \varepsilon\right)$-approximativer Algorithmus für BIN-PACKING beliebige Instanz des PARTITION Problems effizient entscheiden. → geht nicht falls $P \neq NP$

$\square$

Insbesondere gibt es auch kein PTAS für BIN-PACKING

G.36. Ist das alles was wir sagen können?

Beachte:

- Dieses Problem verhält sich anders als z.B. skalierbare Probleme wie SCHEDULING oder min-SPANNBAUM:

jeder Algorithmus der nicht-optimal ist braucht mindestens OPT+1 Behälter für irgendeine Instanz (und sogar schon mit OPT=2 gilt dies)

- die untere Schranke $\frac{3}{2}$ sagt deshalb sehr wenig vom Problem aus (vielmehr nur von Instanzen mit OPT=2)

- Gibt es noch Hoffnung auf einen fast-optimalen Algorithmus? Er kann nicht fast-optimal werden was den Approximationsfaktor – also die multiplikative Güte betrifft. Wie könnte er in einem anderen Sinne fast optimal sein?

$\rightarrow$ es könnte noch sein, dass ein effizienter Algorithmus immer eine Lösung mit OPT+1 Bins ausgibt...

Ob ein effizienter Algorithmus mit additiver Güte 1 (ALG $\leq$ OPT+1) existiert, ist nicht bekannt. (aber auch nicht ausgeschlossen). Es gibt aber für jedes $\varepsilon > 0$ einen Algorithmus in Zeit poly(n) der $(1+\varepsilon)\cdot$OPT $+1$ Behälter braucht. Eine solche Familie von Algorithmen $(A_\varepsilon)_{\varepsilon>0}$ (d.h. mit irgendeiner additiven Konstante) nennt man <u>APTAS</u> (<u>Asymptotisches PTAS</u>) für ein Minimierungsproblem weil der Approximationsfaktor gegen $(1+\varepsilon)$ geht als OPT $\rightarrow \infty$

Vorbereitung

Behauptung: BIN-PACKING ist 'effizient' lösbar, falls die Anzahl der möglichen Gewichte eine Konstante G ist, und alle Gewichte größer als eine Konstante $\delta > 0$ sind.

G ist also die Anzahl der verschiedenen Gewichts-Typen

Beispiel:

Wir wissen, dass es nur 3 verschiedene Gewichte gibt, und diese Gewichte alle größer als $\frac{1}{100}$ sind.

Wir brauchen: einen Algorithmus, der für alle solche Eingaben Laufzeit $\leq q(n)$ hat für irgendein Polynom $q()$. (3 und $\frac{1}{100}$ sind hier Konstanten und dürfen im Exponenten in $q()$ erscheinen)

Idee: wir können alle möglichen Bepackungen ausprobieren, und die mit den wenigsten Behältern ausgeben; die Anzahl der möglichen Bepackungen ist polynomiell in n (und exponentiell in $100 = \frac{1}{\delta}$ und doppelt-exponentiell in $3 = G$).

Schritt 1: Wir notieren alle möglichen Bepackungen $\boxed{\text{eines}}$ Behälters.

Grobe Schätzung: von jedem Gewicht passen maximal 99 Stücke in einen Bin, weil alle $> \frac{1}{100}$ sind.

Die Anzahl verschiedener Vektoren mit 3 Einträgen jeweils ganzzahlig zwischen 0 und 99, ist $100 \cdot 100 \cdot 100 = 100^3$

Gew. 1. Gew 2. Gew. 3.

$$(\ 60 \ , \ 42 \ , \ 35 \ )$$

$$\left(\frac{1}{\delta}\right)^{G}$$

G38.

jede Bepackung <u>eines</u> Behälters kann eindeutig durch einen solchen Vektor repräsentiert werden.

(manche Vektoren repräsentieren ungültige Bepackungen, aber das macht nichts.)

Allgemein haben wir $\leq \left(\frac{1}{\delta}\right)^G$ Bepackungstypen für einen Bin.

[<u>Eine bessere Schätzung</u>: es passen <u>insgesamt</u> max 100 Stücke in den Bin. Die Anzahl der Vektoren mit 3 ganzzahligen Einträgen (≥ 0) so dass die <u>Summe</u> der Einträge maximum 100 ist, gleicht der Anzahl der Vektoren mit 4 ganzzahligen Einträgen (≥ 0) so dass die <u>Summe</u> der Einträge <u>genau</u> 100 ist. (Warum?)

$$
\begin{array}{cccc}
\text{Gew. 1} & \text{Gew. 2.} & \text{Gew. 3.} & \text{Rest von 100} \\
\left(\underbrace{3, \quad 7, \quad 70,}_{\text{Bepackung.}} \right. & & & \left. 20 \right)
\end{array}
$$

$\updownarrow$

$$
111 \; 0 \; 1111111 \; 0 \; \underbrace{11111 \quad 111}_{70} \; 0 \; \underbrace{11111 \; 111}_{20}
$$

ein solcher Vektor entspricht eindeutig einem 0-1 String aus genau 100 1-er und 3 0-en. Die Anzahl solcher Strings ist $\binom{103}{3}$, und allgemein $\binom{\frac{1}{\delta}+G}{G}$

auch exponentiell in G ←]

Schritt 2: Sei B die Anzahl möglicher Bepackungen eines Behälters. (Wir haben gesehen $B \leq \left(\frac{1}{\delta}\right)^G$ bzw. $\leq \binom{\frac{1}{\delta}+G}{G}$)

Grob: Wir haben höchstens n Behälter von jedem Bepackungstyp

$$\begin{array}{cccc} \text{Typ1} & \text{Typ2} \ldots & & \text{Typ } B \\ (\leq n & \leq n & & \leq n) \end{array}$$

$\Rightarrow$ es gibt höchstens $\sim n^B$ mögliche solche Vektoren und jede Bepackung in max n Bins entspricht einem solchen Vektor.

Es gibt also $\leq n^B$ mit $B \leq \left(\frac{1}{\delta}\right)^G$

$\rightarrow \leq n^{\left(\frac{1}{\delta}^G\right)}$ mögliche Bepackungen zu testen

— eine astronomische Zahl, aber immerhin polynomielle in n.

Empfindlichere Schätzung ergibt $\leq \binom{n+B}{B}$ Bepackungen in $\leq n$ Bins wobei $B \leq \binom{\frac{1}{\delta}+G}{G}$

selbst diese Schätzung ergibt viel zu hohen Grad für unser Polynom in der Laufzeit. Diese und die folgenden Ergebnisse sind rein theoretisch. Wir lernen jedoch allgemeine Methoden für den Entwurf eines PTAS kennen. Wir benutzen diese Idee um ein APTAS für beliebige Gewichte zu entwickeln.

Definition: Ein asymptotisches polynomielles Approximationsschema (APTAS) (für Minimierungsprobleme) ist eine Familie $(A_\varepsilon)_{\varepsilon>0}$ von Polynomialzeit-Algorithmen zusammen mit einer Konstante c, so dass jeder Algorithmus A_ε für jede Instanz I eine Lösung mit Wert höchstens $(1+\varepsilon)\,OPT(I)+c$ berechnet.

G40.

Ein APTAS für BIN PACKING

das für beliebige Gewichte $g_1 g_2 \dots g_n$ $(0 \leq g_i \leq 1)$ und $\varepsilon > 0$ eine Bepackung mit $(1+2\varepsilon)\,OPT(I) + 1$ Behältern ausgibt.

(Wir haben $(1+2\varepsilon)$ in der Approx. Faktor nur wegen einfacherer Notation während der Analyse. Sonst könnte man natürlich $\varepsilon' = \frac{\varepsilon}{2}$ im Algorithmus verwenden.)

Informelle Beschreibung:

- Wie erreichen wir, dass alle Gewichte g_i eine Mindestgröße δ haben?
  Unser δ wird das festgelegte ε sein.
  Anfangs legen wir alle Gewichte die $< \varepsilon$ sind an die Seite; am Ende werden wir sie First Fit in die Restkapazitäten (Lücken) der Bins einfüllen; falls es keine Lücken mehr gibt, sind alle Behälter bis $1-\varepsilon$ gefüllt, und wir dürfen neue öffnen.

- Wie erreichen wir, dass es konstant-viele verschiedene Gewichte gibt?
  Wir werden die Gewichte g_i aufrunden in G verschiedene Größen (Gewichtsklassen). Eine Bepackung der aufgerundeten Gewichte wird auch für die echten Gewichte eine legale Lösung sein. Wie werden die Gewichte gerundet?

In diesem Fall haben wir das Ziel, dass es für jede gerundete Größe gleichviele Gewichte gibt. (beim RUCKSACK Problem werden wir eine andere Methode für Runden sehen.)

Gewichte in der Eingabe |

die gerundeten Gewichte (Gewichtsklassen) ∘

wir haben G Gewichtsklassen, und in jeder Gewichtsklasse $E = \lceil \frac{n}{G} \rceil$ Gewichte

<u>Das APTAS für BIN-PACKING</u> ε sei vorgegeben

- lege Objekte mit Gewicht $\leq \varepsilon$ an die Seite
- zerlege die Menge der restlichen Objekte in G Gewichtsklassen mit jeweils $E = \lceil \frac{n}{G} \rceil$ Elementen (und eine Klasse mit $\leq E$)
- runde jedes Gewicht auf: setze ihn auf das größte Gewicht seiner Klasse
- jetzt gibt es nur G verschiedene Gewichte; löse BIN PACKING <u>exakt</u> für die <u>gerundeten</u> Gewichte in polynomieller Laufzeit $\binom{n+B}{B}$ wobei $B \leq \binom{\frac{1}{\varepsilon}+G}{G}$
und teile die <u>echten</u> Objekte genauso zu den Bins.
- schließlich füge die kleinen Objekte mit Gewicht $\leq \varepsilon$ in die Behälter (und ggf. in neue Behälter) mit First-Fit ein.

— Laufzeit: $n^{\left(\frac{1}{\varepsilon}\right)^G}$

Theorem: Dieses APTAS für BIN-PACKING benötigt höchstens $(1+2\varepsilon)\,OPT+1$ Behälter bei geeigneter Wahl von G.

Beweis:

1. Wir nehmen zuerst an, dass es gar keine kleine Objekte $\leq \varepsilon$ gibt in der Eingabe I.

Behauptung: Das APTAS öffnet höchstens $OPT(I)+E$ Behälter.

Warum? Das APTAS ist optimal für die gerundete Eingabe. Wir brauchen nur zu zeigen, dass $OPT(I)+E$ Behälter ausreichen für die gerundete Eingabe.
Wir packen die E Objekte in der größten (rechtesten) Gewichtsklasse je in einen Behälter. Wir zeigen, dass $OPT(I)$ Behälter reichen für die aufgerundeten Gewichte aller anderen Gewichtsklassen:
Betrachte eine optimale Bepackung der ungerundeten Objekte in $OPT(I)$ Behälter. In dieser Bepackung ersetze die E Objekte | in der größten Gewichtsklasse mit den aufgerundeten o der zweitgrößten Gewichtsklasse; die E Objekte | in der zweiten Gewichtsklasse mit den E aufgerundeten o aus der dritten Gewichtsklasse, usw.
Somit reichen $OPT(I)$ Behälter für alle aufgerundeten Gewichte, die nicht in der rechtesten Gewichtsklasse sind, und insgesamt reichen $OPT(I)+E$ Behälter für APTAS. □

$$APTAS \leq OPT+E$$

Wir brauchen nur noch, dass $OPT + E \leq (1+\varepsilon) OPT$,

d. h. $E \leq \varepsilon \cdot OPT$

→ wir brauchen eine Abschätzung von OPT; wir wissen nur, dass $OPT \geq n \cdot \varepsilon$, weil wir n Objekte der Größe mindestens ε haben.

Wir legen deshalb E so fest, dass $E \leq \varepsilon^2 n$ gilt,

und dann $OPT + E \leq OPT + \varepsilon^2 n \leq OPT + \varepsilon \cdot OPT = (1+\varepsilon) \cdot OPT$

⇒ setze $E = \lfloor n \cdot \varepsilon^2 \rfloor$

und $G = \frac{n}{E} \sim \left\lceil \frac{1}{\varepsilon^2} \right\rceil$ ⇝ Laufzeit: $n^{\left(\frac{1}{\varepsilon}^{\frac{1}{\varepsilon^2}}\right)}$

wobei n die Anzahl der Objekte mit Gewicht $\geq \varepsilon$ ist!

2. Wenn es in der Instanz auch kleine Objekte mit Gewicht $\leq \varepsilon$ gibt

- Der Algorithmus für die großen Objekte $(> \varepsilon)$ gibt eine Bepackung in maximal $(1+\varepsilon) OPT(\text{große}) \leq (1+\varepsilon) OPT(I)$ Bins aus.
- Die kleinen Objekte $(\leq \varepsilon)$ werden mit First-Fit anschließend in die Behälter gefüllt.
- Wenn keine weitere Behälter geöffnet werden, dann gilt $(1+\varepsilon) OPT$ weiterhin.
- Wenn weitere Behälter geöffnet werden, dann wird jeder Bin bis auf den letzten mindestens $1-\varepsilon$ Gesamtgewicht tragen. Es gilt

$$(APTAS(I) - 1)(1-\varepsilon) < \text{Gesamtgewicht} \leq OPT(I)$$

$$APTAS(I) < \frac{OPT(I)}{1-\varepsilon} + 1 \leq (1+2\varepsilon) OPT(I) + 1$$

falls $\varepsilon < \frac{1}{2}$ □

DYNAMISCHE PROGRAMMIERUNG

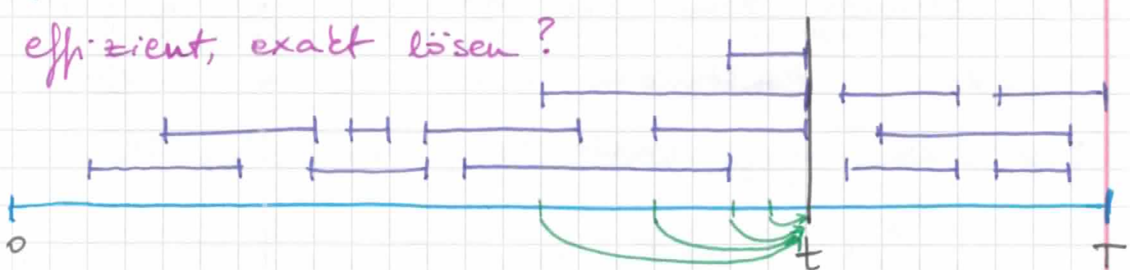
INTERVALL-SCHEDULING II.
(gewichtet)

Eingabe: Aufgaben $A_1\ A_2 \ldots A_i \ldots A_n$ mit ganzzahligen (nicht wichtig) Startpunkten $s_1\ s_2 \ldots s_i \ldots s_n$ und Endpunkten $e_1\ e_2 \ldots e_i \ldots e_n$

die Länge von Aufgabe i ist $l_i = e_i - s_i$

Ausgabe: ein Schedule der Aufgaben auf einem Prozessor, ohne Überlappen, mit maximaler Gesamtlänge der ausgeführten Aufgaben

Das Problem mit maximaler Anzahl der ausgeführten Aufgaben haben wir mit einem greedy Algorithmus gelöst. Wie kann man diese Variante des Problems effizient, exakt lösen?



- sei $[0,T]$ das ganze Zeitintervall $(T = e_{max})$
- Bezeichne $L(t)$ die Gesamtlänge in einem optimalen-Schedule-auf-dem-Intervall-$[0,t]$ für jede $t = 1,2,3,\ldots,T$
- $L(0) = 0$ ist trivial
- $L(T)$ ergibt den optimalen Zielwert für das ganze Intervall
- $L(t)$ kann man berechnen, wenn man $L(t-j)$ für alle (oder: bestimmte) $j \geq 1$ kennt:

DP.2.

- setze $L(0) = 0$

- FOR $t = 1$ to T DO

$$L(t) = \max \left\{ L(t-1), \max \left\{ l_i + L(s_i) \,\middle|\, \begin{array}{l} \text{über alle } i \\ \text{so dass } e_i = t \end{array} \right\} \right\}$$

der beste Zielwert wenn keine Aufgabe in t endet in der Lösung

der beste Zielwert so dass eine Aufgabe k genau in t endet $e_k = t$

1. Wir haben eine Rekursionsgleichung für $L(t)$

2. Wir berechnen alle Werte $L(0), L(1), L(2), \ldots, L(T)$, die Optima für die Teilprobleme von $[0, t]$, und speichern sie als Einträge einer "Tabelle". Wir benutzen diese Werte immer wieder für die Optimierung größerer Teilprobleme.

3. Wir berechnen also das Optimum $L(T)$ 'bottom-up'.

4. Wir können dann eine optimale Lösung (Schedule) 'top-down' konstruieren, mit Hilfe von zusätzlicher Information gespeichert in der bottom-up Phase

   d.h. welche / keine A_i mit $e_i = t$ für $L(t)$ verwendet wird.

Bemerkungen: - Es reicht für jede $t = s_i$ und $t = e_i$ den $L(t)$ zu berechnen (und $L(t-1)$ in der Gleichung entsprechend ersetzen). Dies geht auch wenn e_i und s_i nicht ganzzahlig sind. Die Laufzeit ist somit $O(n)$ (Warum??)

- der selbe Algorithmus löst das Problem mit beliebigen Gewichten w_i (statt l_i) für die Aufgaben A_i

Schulbeispiel für Dynamische Programmierung:

Das Pascalsche Dreieck

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}
$$

$$
\begin{array}{ccccccccccc}
 & & & & & \binom{0}{0} & & & & & \\
 & & & & \binom{1}{0} & & \binom{1}{1} & & & & \\
 & & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & & \\
 & & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & & \\
 & \binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4} & \\
\binom{5}{0} & & \binom{5}{1} & & \binom{5}{2} & & \binom{5}{3} & & \binom{5}{4} & & \binom{5}{5}
\end{array}
$$

- man könnte die (dumme) Idee haben den Binomialkoeffizienten $\frac{n!}{k!\,(n-k)!} = \binom{n}{k}$ mit Hilfe der Rekursionsgleichung:

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

zu berechnen. Wenn er schon so rechnen möchte,

Sollte er einen rekursiven Algorithmus, oder einen Algorithmus mit Dynamischer Programmierung verwenden?

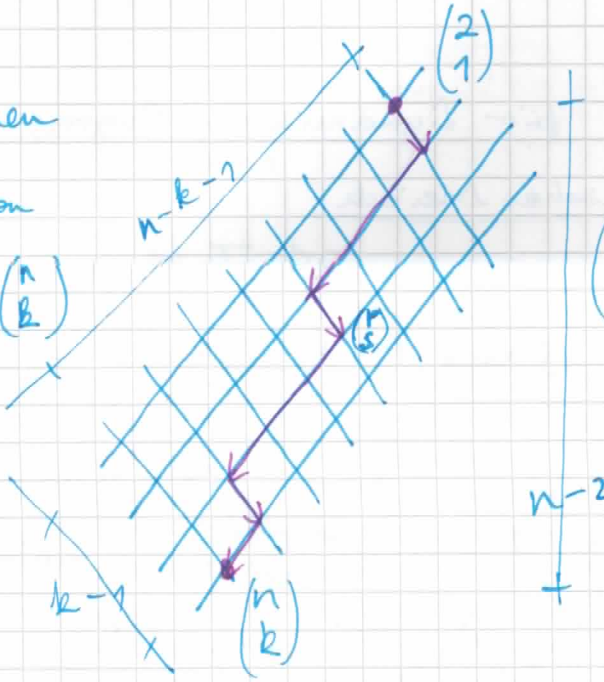
- Falls der Algorithmus rekursiv implementiert wird, werden die Werte der früheren Teilprobleme $\binom{n-i}{k-j}$ nicht gespeichert, und jedes mal wenn gebraucht, neu berechnet.

Beispiel:

Ungefähr wie oft wird $\binom{2}{1} = \binom{1}{0} + \binom{1}{1}$ berechnet, während der einzige Berechnung von $\binom{n}{k}$ mit der rekursiven Methode?

DP4.

für jeden solchen Weg im Gitter von $\binom{2}{1}$ nach unten zum $\binom{n}{k}$ (bzw. von $\binom{n}{k}$ durch rekursive Rufe nach $\binom{2}{1}$)



und solche Wege gibt es $\binom{n-2}{k-1} \sim \binom{n}{k}$

– ein Algorithmus mit dynamischer Programmierung würde stattdessen mit den einfachsten Teilproblemen anfangen, und die berechneten Werte für spätere Nutzung in der Tabelle (dh. im Pascalschen-Dreieck) speichern. ("bottom-up" entspricht in unserer Darstellung von oben nach unten). Er würde die $\sim (n-k)\cdot k$ nötigen Tabellenwerte $\binom{r}{s}$ (siehe oben) jeweils einmal mit der Rekursionsgleichung $\binom{r}{s} = \binom{r-1}{s-1} + \binom{r-1}{s}$ berechnen.

(Allgemein gilt:

'top' = die ganze (ursprüngliche) Problem-Instanz

'bottom' = die elementaren Teilprobleme)

DAS RUCKSACK Problem

Eingabe: eine Gewichtsschranke G, n Objekte mit Gewichten $g_1 g_2 \dots g_n$ $(g_i \leq G)$ und Werten $w_1 w_2 \dots w_n$

Ausgabe: finde eine Auswahl von Objekten mit maximalem Gesamtwert, so dass die Gewichtsschranke nicht überschritten wird.

Die Entscheidungsversion von RUCKSACK ist NP-vollständig

DP 5.

Ein exakter Algorithmus für ganzzahlige Gewichte $g_i \in \mathbb{N}$ mit dynamischer Programmierung

→ bezeichne $W(i, g)$ den größten Wert, so dass eine Bepackung aus den Objekten $\{1, 2, \ldots, i\}$ mit Gewicht <u>genau</u> g und diesem Wert existiert
(für jede $i = 0, 1, 2, \ldots, n$ und $g = 0, 1, 2 \ldots G$)

Wie berechnet man $W(i,g)$ aus früheren $W(i', g')$ Werten?
$i' \leq i$ und $g' \leq g$

→ Wir stellen eine Rekursionsgleichung für $W(i, g)$ auf:

$$W(i,g) = \max \{ \underbrace{W(i-1, g)}_{\text{Objekt } i \text{ nicht hinzugenommen}}, \underbrace{w_i + W(i-1, g-g_i)}_{\text{Objekt } i \text{ hinzugenommen}} \}$$

die Basisfälle sind

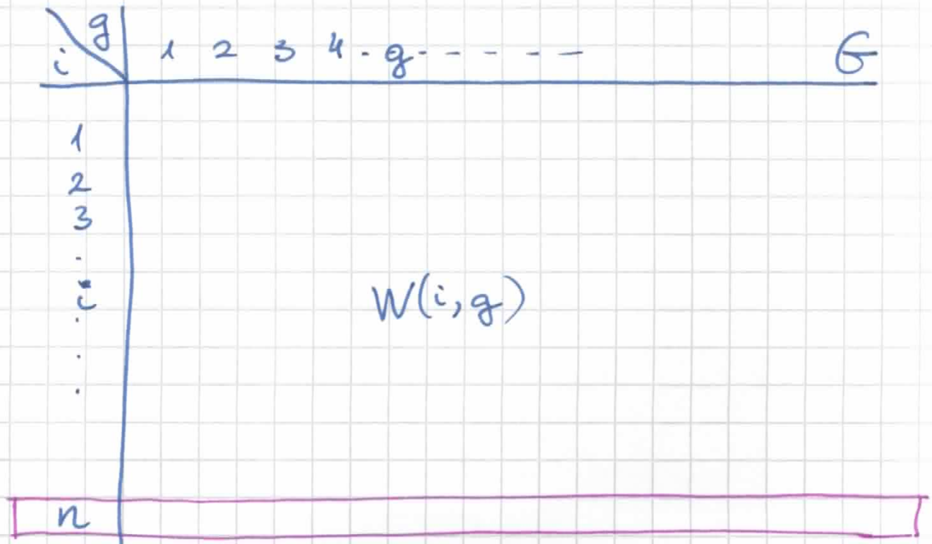
$$W(0,0) = 0$$

$$W(0, g) = -\infty \quad \text{wenn } g \neq 0$$

$$W(i, g) = -\infty \quad \text{für } g < 0$$

(setze $-\infty$ für unmögliche Fälle bei Maximierungsproblemen)

DP 6.



Der größte $W(n,g)$ Wert in der Zeile n ist optimal

→ die Tabelle von $W(i,g)$ Werten berechnet und speichert der Algorithmus von oben nach unten

~~und speichert sie in der Tabelle~~

- initialisiere
- FOR $i = 1$ to n DO

  FOR $g = 0$ to G DO

  $W(i,g) = \max\{W(i-1,g),\ w_i + W(i-1, g-g_i)\}$

→ Laufzeit: $O(\overset{G}{\cancel{W}} \cdot n)$

→ Wie wird eine optimale Bepackung berechnet?

Wenn mit jedem $W(i,g)$ noch ein Zeiger auf $W(i-1,g)$ oder auf $W(i-1, g-g_i)$ (um $W(i,g)$ zu realisieren) gesetzt wird, dann kann der Algorithmus am Ende vom maximalen $W(n,g)$ ausgehend, in $O(n)$ Schritten mit Backtracking die Objekte in einer optimalen Bepackung finden.

Beachte, dass erst am Ende klar wird, welche Zeiger/Objekte nützlich sind!