Analyse – Pfadkompression + Union by rank

Satz: $m\times$ find $+$ $n\times$ link brauchen Zeit $O(m\alpha_T(m,n))$ mit

$$\alpha_T(m,n) = \min\{i \geq 1 : A(i,\lceil m/n\rceil) \geq \log n\}$$

und

$$\begin{aligned} A(1,j) &= 2^j && \text{for } j \geq 1,\\ A(i,1) &= A(i-1,2) && \text{for } i \geq 2,\\ A(i,j) &= A(i-1,A(i,j-1)) && \text{for } i \geq 2 \text{ and } j \geq 2. \end{aligned}$$

Beweis: [Tarjan 1975, Seidel Sharir 2005]
A ist die Ackermannfunktion und α_T die inverse Ackermannfunktion.
$\alpha_T(m,n) = \omega(1)$ aber ≤ 5 für alle physikalisch denkbaren n, m.

Ackermannfunktion – Beispiele

$$A(1,j) = 2^j \qquad \text{for } j \geq 1,$$
$$A(i,1) = A(i-1,2) \qquad \text{for } i \geq 2,$$
$$A(i,j) = A(i-1,A(i,j-1)) \qquad \text{for } i \geq 2 \text{ and } j \geq 2.$$

$$A(2,1) = A(1,2) = 2^2$$
$$A(2,2) = A(1,A(2,1)) = 2^{2^2}$$
$$A(2,3) = A(1,A(2,2)) = 2^{2^{2^2}}$$
$$A(2,4) = A(1,A(2,3)) = 2^{2^{2^{2^2}}}$$
$$A(3,1) = A(2,2) = 2^{2^2}$$
$$A(3,2) = A(2,A(3,1)) = A(2,16) = ???$$
$$A(4,1) = A(3,2) = ???$$

Kruskal mit Union-Find

Sei $V = 1..n$

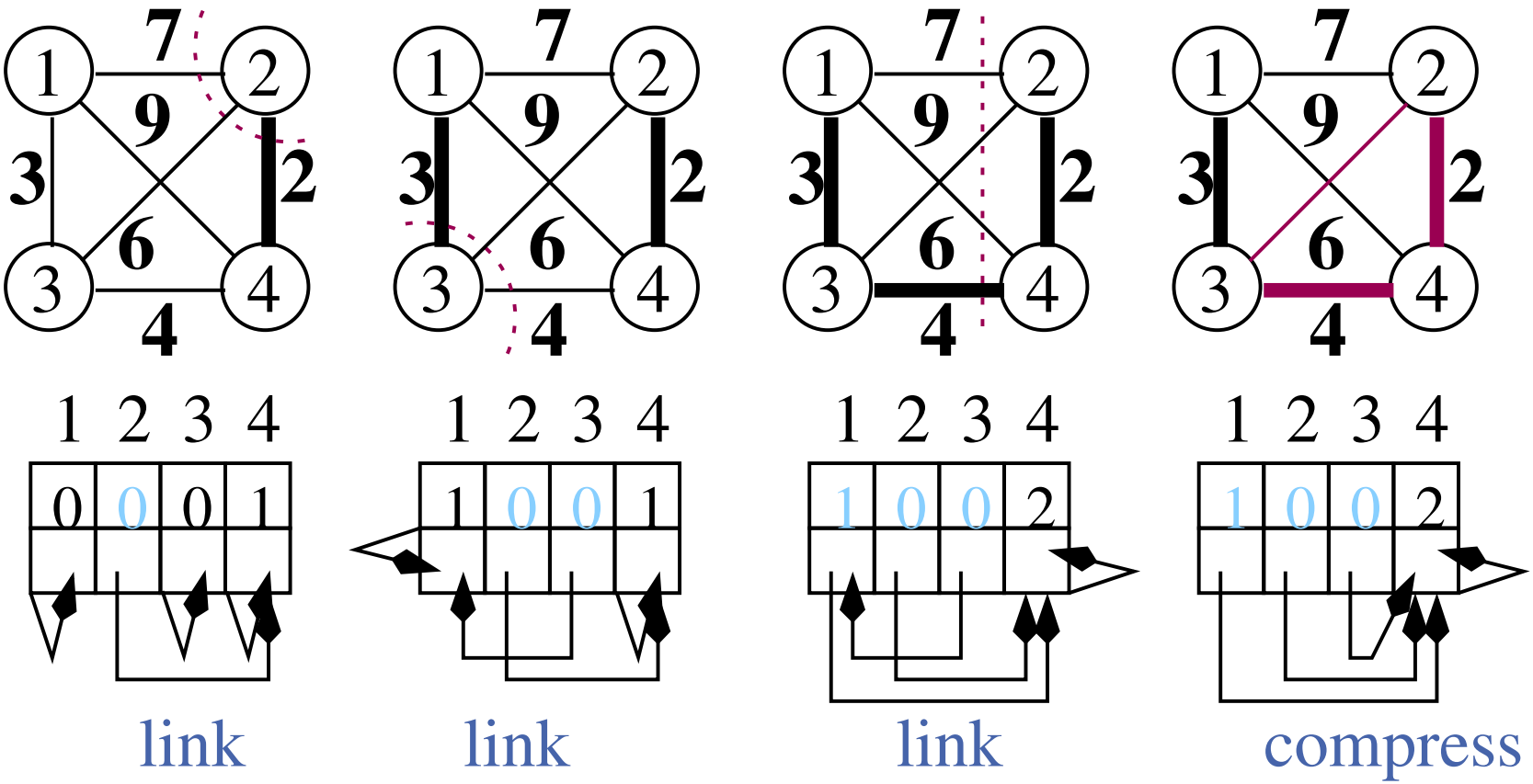
```
Tc : UnionFind(n)                          // encodes components of forest T
foreach (u, v) ∈ E in ascending order of weight do          // sort
    if Tc.find(u) ≠ Tc.find(v) then
        output {u, v}
        Tc.union(u, v)                                     // link reicht auch
```

Zeit $O(m \log m)$. Schneller für ganzzahlige Gewichte.

Graphrepräsentation: Kantenliste

Bäume im MSF $\leftrightarrow$ Blöcke in Partition $\rightarrow$ Wurzelbäume,
aber mit anderer Struktur als die Bäume im MSF!

Beispiel



Vergleich Jarník-Prim $\leftrightarrow$ Kruskal

Pro Jarník-Prim

- Asymptotisch gut für alle m, n
- Sehr schnell für $m \gg n$

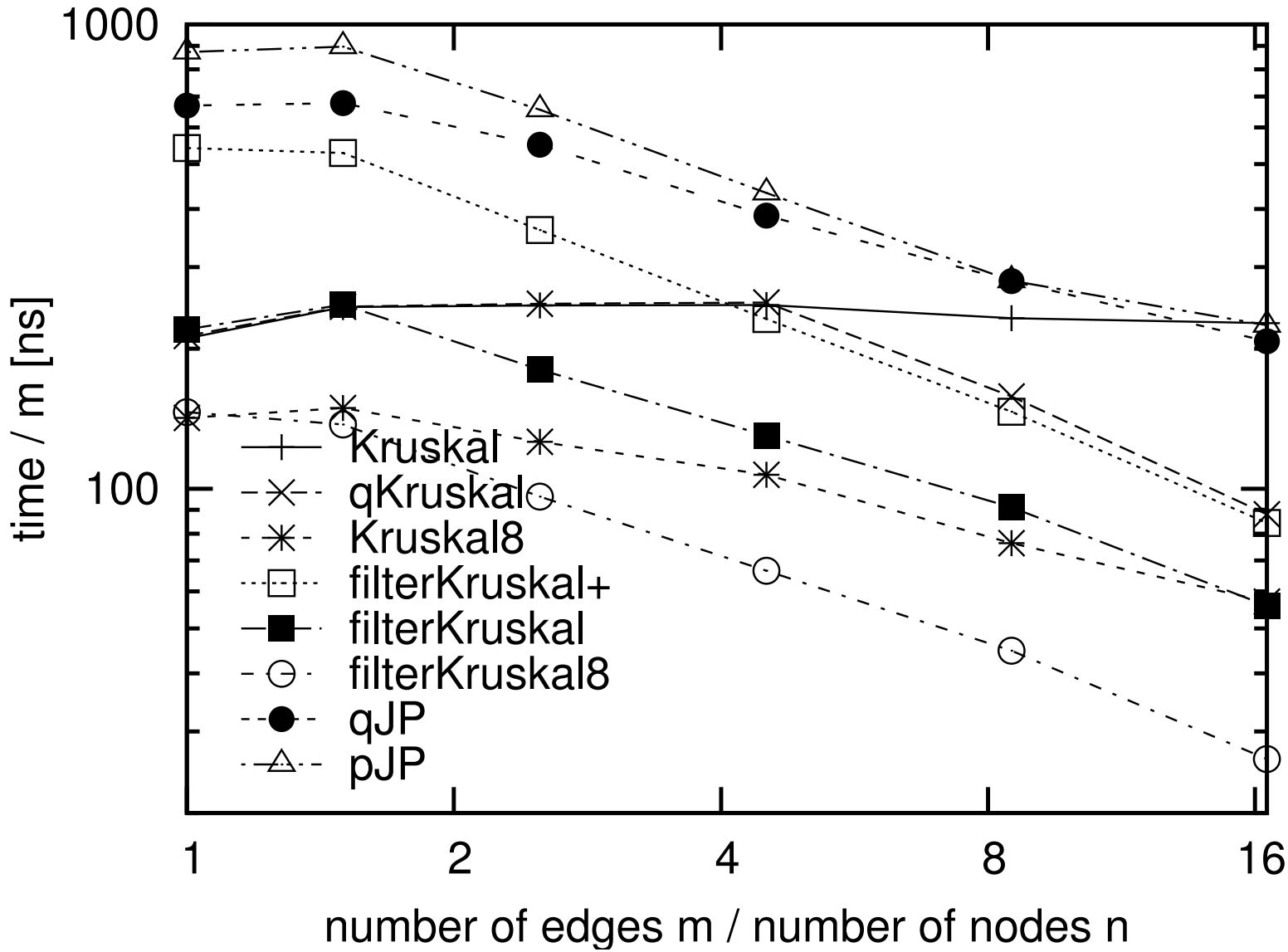
Pro Kruskal

- Gut für $m = \mathrm{O}(n)$
- Braucht nur Kantenliste
- Profitiert von schnellen Sortierern (ganzzahlig, parallel,...)
- Verfeinerungen auch gut für große m/n

Mehr MST-Algorithmen

- Zeit $O(m \log n)$ [Boruvka 1926]
  Zutat vieler fortgeschrittener Algorithmen
- Erwartete Zeit $O(m)$ [Karger Klein Tarjan 1995],
  parallelisierbar, externalisierbar
- Det. Zeit $O(m\alpha_T(m,n))$ [Chazelle 2000]
- "optimaler" det. Algorithmus [Pettie, Ramachandran 2000]
- "Invented at ITI":
  Praktikabler externer Algorithmus [Sanders Schultes Sibeyn 2004]
  Verbesserung von Kruskal (parallelisierbar, weniger Sortieraufwand).
  [Osipov Sanders Singler 2009]

Messungen, Zufallsgraph, $n = 2^{22}$



Zusammenfassung

- Schnitt- und Kreiseigenschaft als Basis für abstrakte Algorithmen.
  Entwurfsprinzip: benutze abstrakte Problemeigenschaften.
- Beweise mittels Austauschargumenten
- Implementierung braucht effiziente Datenstrukturen.
  Auch ein Entwurfsprinzip...

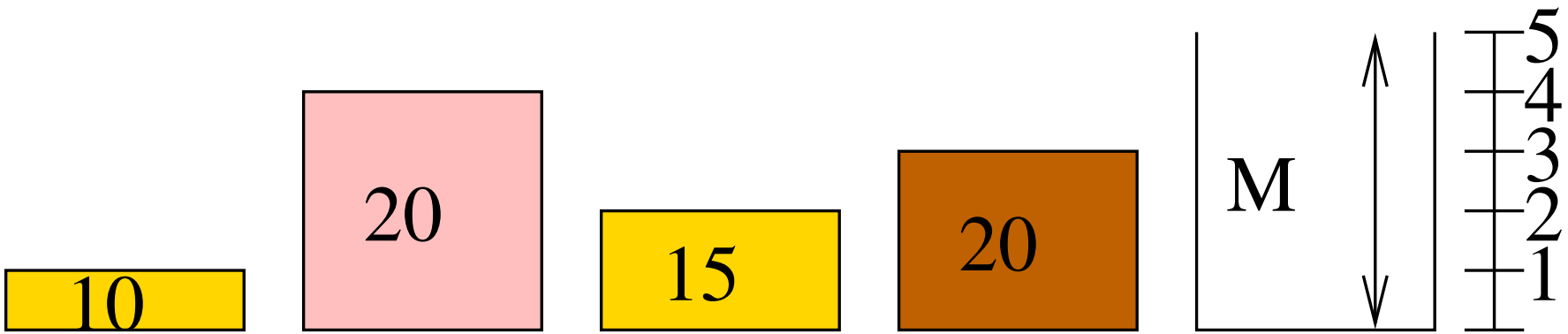
Zusammenfassung

- Schnitt- und Kreiseigenschaft als Basis für abstrakte Algorithmen.
  Entwurfsprinzip: benutze abstrakte Problemeigenschaften.
- Beweise mittels Austauschargumenten
- Implementierung braucht effiziente Datenstrukturen.
  Auch ein Entwurfsprinzip...
- Dijkstra $\approx$ JP.
  Noch ein Entwurfsprinzip:
  Greedy-Algorithmus effizient implementiert mittels Prioritätsliste
- Union-Find: effiziente Verwaltung von Partitionen mittels Pfadkompression und Union-by-rank.
  Beispiel für einfache Algorithmen mit nichttrivialer Analyse

Kap. 12: Generische Optimierungsansätze

- Black-Box-Löser
- Greedy
- Dynamische Programmierung
- Systematische Suche
- Lokale Suche
- Evolutionäre Algorithmen

Durchgehendes Beispiel: Rucksackproblem



- n Gegenstände mit Gewicht $w_i \in \mathbb{N}$ und Profit p_i
- Wähle eine Teilmenge $\mathbf{x}$ von Gegenständen derart, dass $\sum_{i \in \mathbf{x}} w_i \leq M$ und
- maximiere den Profit $\sum_{i \in \mathbf{x}} p_i$

Allgemein: Maximierungsproblem $(\mathscr{L}, f)$

- $\mathscr{L} \subseteq \mathscr{U}$: zulässige Lösungen
- $f : \mathscr{L} \to \mathbb{R}$ Zielfunktion
- $\mathbf{x}^* \in \mathscr{L}$ ist optimale Lösung falls $f(\mathbf{x}^*) \geq f(\mathbf{x})$ für alle $\mathbf{x} \in \mathscr{L}$

Minimierungsprobleme: analog

Problem: variantenreich, meist NP-schwer

Black-Box-Löser

- (Ganzzahlige) Lineare Programmierung
- Aussagenlogik
- Constraint-Programming $\approx$ Verallgemeinerung von beidem

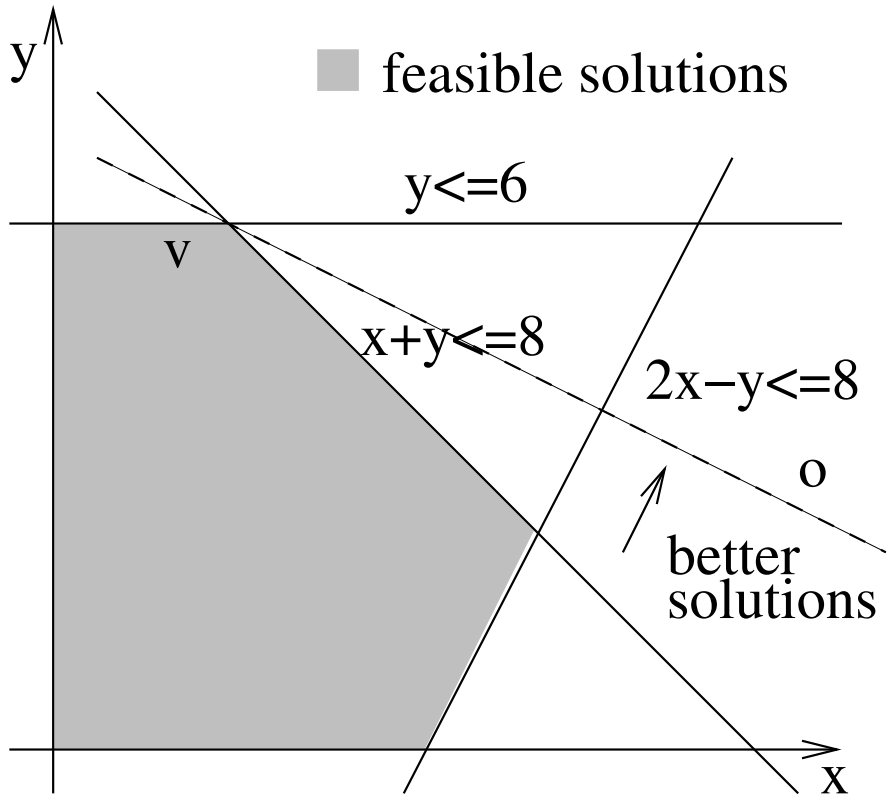
Lineare Programmierung

Ein lineares Programm mit n Variablen und m Constraints (NB) wird durch das folgende Minimierungs-/Maximierungsproblem definiert:

- Kostenfunktion $f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x}$
  $\mathbf{c}$ ist der Kostenvektor
- m Constraints der Form $\mathbf{a}_i \cdot \mathbf{x} \bowtie_i b_i$ mit $\bowtie_i \in \{\leq, \geq, =\}$, $\mathbf{a}_i \in \mathbb{R}^n$.
  Wir erhalten:

$$\mathscr{L} = \{\mathbf{x} \in \mathbb{R}^n : \forall j \in 1..n : x_j \geq 0 \wedge \forall i \in 1..m : \mathbf{a}_i \cdot \mathbf{x} \bowtie_i b_i\} \quad .$$

Ein einfaches Beispiel



Beispiel: Kürzeste Wege

maximiere $\sum_{v \in V} d_v$

so dass $d_s = 0$

$d_w \leq d_v + c(v,w)$ für alle $(v,w) \in E$

Eine Anwendung – Tierfutter

- n Futtersorten,
  Sorte i kostet c_i Euro/kg.
- m Anforderungen an gesunde Ernährung (Kalorien, Proteine, Vitamin C,...).
  Sorte i enthält a_{ji} Prozent des täglichen Bedarfs pro kg bzgl. Anforderung j
- Sei a_{ji} die i-te Komponente von Vektor $\mathbf{a}_j$.

Eine Anwendung – Tierfutter

- n Futtersorten,
  Sorte i kostet c_i Euro/kg.
- m Anforderungen an gesunde Ernährung (Kalorien, Proteine, Vitamin C,...).
  Sorte i enthält a_{ji} Prozent des täglichen Bedarfs pro kg bzgl. Anforderung j
- Sei a_{ji} die i-te Komponente von Vektor $\mathbf{a}_j$.
- Definiere x_i als
  zu beschaffende Menge von Sorte i
- LP-Lösung gibt eine kostenoptimale "gesunde" Mischung.

Verfeinerungen

- Obere Schranken (Radioaktivität, Cadmium, Kuhhirn, ...)
- Beschränkte Reserven (z. B. eigenes Heu)
- bestimmte abschnittweise lineare Kostenfunktionen (z. B. mit Abstand wachsende Transportkosten)

- Minimale Abnahmemengen
- die meisten nichtlinearen Kostenfunktionen
- Ganzzahlige Mengen (für wenige Tiere)
- Garbage in, Garbage out

Algorithmen und Implementierungen

- LPs lassen sich in polynomieller Zeit lösen [Khachiyan 1979]
- Worst case $\mathrm{O}\left(\max(m,n)^{\frac{7}{2}}\right)$
- In der Praxis geht das viel schneller
- Robuste, effiziente Implementierungen sind sehr aufwändig

$\rightsquigarrow$ Fertige freie und kommerzielle Pakete

Ganzzahlige Lineare Programmierung

ILP: Integer Linear Program, lineares Programm mit der zusätzlichen Bedingung $x_i \in \mathbb{N}$.
oft: 0/1 ILP mit $x_i \in \{0,1\}$

MILP: Mixed Integer Linear Program, lineares Programm bei dem einige Variablen ganzzahlig sein müssen.

Lineare Relaxation: Entferne die Ganzzahligkeitsbedingungen eines (M)ILP

Beispiel: Rucksackproblem

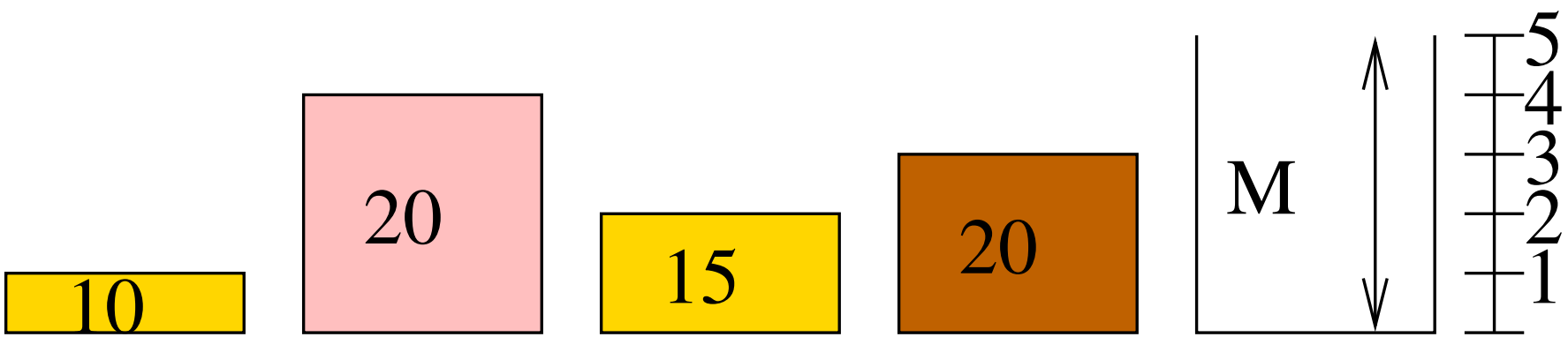
maximiere $\mathbf{p} \cdot \mathbf{x}$

derart, dass

$$\mathbf{w} \cdot x \leq M, x_i \in \{0,1\} \text{ for } 1 \leq i \leq n \ .$$

$x_i = 1$ gdw. Gegenstand i in den Rucksack kommt.

0/1 Variablen sind typisch für ILPs



Umgang mit (M)ILPs

- – NP-schwer
- \+ Ausdrucksstarke Modellierungssprache
- \+ Es gibt generische Lösungsansätze, die manchmal gut funktionieren
- \+ Viele Möglichkeiten für Näherungslösungen
- \+ Die Lösung der linearen Relaxation hilft oft, z. B. einfach runden.
- \+ Ausgefeilte Softwarepakete

Beispiel: Beim Rucksackproblem gibt es nur eine fraktionale Variable in der linearen Relaxation – Abrunden ergibt zulässige Lösung. Annähernd optimal, falls Gewichte und Profite $\ll$ Kapazität

Nie zurückschauen – Greedy-Algorithmen

(deutsch: gierige Algorithmen, wenig gebräuchlich)

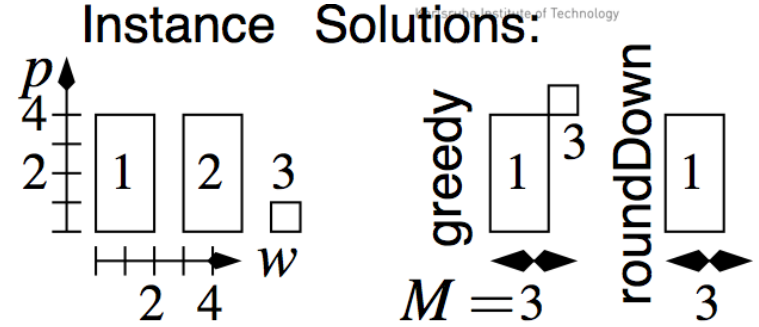
Idee: treffe jeweils eine lokal optimale Entscheidung

Optimale Greedy-Algorithmen

- Dijkstras Algorithmus für kürzeste Wege
- Minimale Spannbäume
  - Jarník-Prim
  - Kruskal
- Selection-Sort (wenn man so will)

Viel häufiger, z.T. mit Qualitätsgarantien.
Mehr: Vorlesungen Algorithmen II und
Approximations- und Onlinealgorithmen

Beispiel: Rucksackproblem

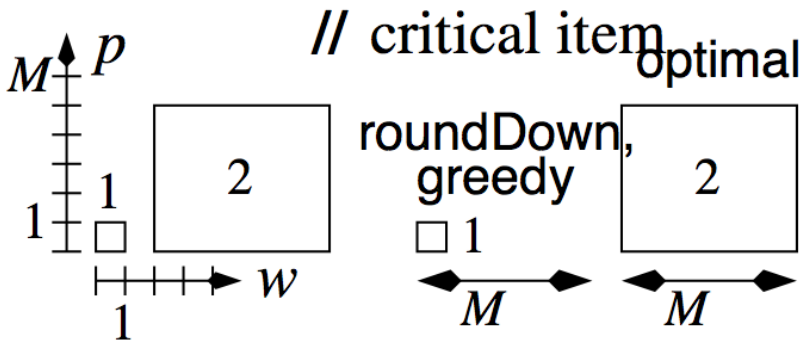


Procedure roundDownKnapsack

sort items by profit density $\frac{p_i}{w_i}$

find $\min\left\{j : \sum_{i=1}^{j} w_j > M\right\}$

output items $1..j-1$



Procedure greedyKnapsack

sort items by profit density $\frac{p_i}{w_i}$

for $i := 1$ **to** n **do**

if there is room for item i **then**

insert it into the knapsack