Kapitel 4:
Approximation Algorithms

Inhalt:

- Greedy Techniques
  - Load-Balancing Problem
  - Center Selection Problem
- Pricing Method
  - Vertex Cover Problem
- Linear Programming and Rounding
  - Vertex Cover Problem
  - Generalized Load-Balancing Problem
- Polynomial Time Approximation Scheme
  - Knapsack Problem

Generalized Load Balancing

Input. Set of m machines M; set of n jobs J.

- Job j must run contiguously on an authorized machine in $M_j \subseteq M$.
- Job j has processing time t_j.
- Each machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \Sigma_{j \in J(i)} t_j$.

Def. The makespan is the maximum load on any machine = $\max_i L_i$.

Generalized load balancing problem. Assign each job to an authorized machine to minimize makespan.

Generalized Load Balancing: Integer Linear Program and Relaxation

ILP formulation. x_{ij} = time machine i spends processing job j.

$$
\begin{array}{llllll}
(IP) & \min & L & & & \\
 & \text{s. t.} & \sum_i x_{ij} & = & t_j & \text{for all } j \in J \\
 & & \sum_j x_{ij} & \leq & L & \text{for all } i \in M \\
 & & x_{ij} & \in & \{0,\ t_j\} & \text{for all } j \in J \text{ and } i \in M_j \\
 & & x_{ij} & = & 0 & \text{for all } j \in J \text{ and } i \notin M_j
\end{array}
$$

LP relaxation.

$$
\begin{array}{llllll}
(LP) & \min & L & & & \\
 & \text{s. t.} & \sum_i x_{ij} & = & t_j & \text{for all } j \in J \\
 & & \sum_j x_{ij} & \leq & L & \text{for all } i \in M \\
 & & x_{ij} & \geq & 0 & \text{for all } j \in J \text{ and } i \in M_j \\
 & & x_{ij} & = & 0 & \text{for all } j \in J \text{ and } i \notin M_j
\end{array}
$$

Generalized Load Balancing: Lower Bounds

Lemma 1. Let L be the optimal value to the LP. Then, the optimal makespan $L^* \geq L$.

Pf. LP has fewer constraints than IP formulation.

Lemma 2. The optimal makespan $L^* \geq \max_j t_j$.

Pf. Some machine must process the most time-consuming job. ▪

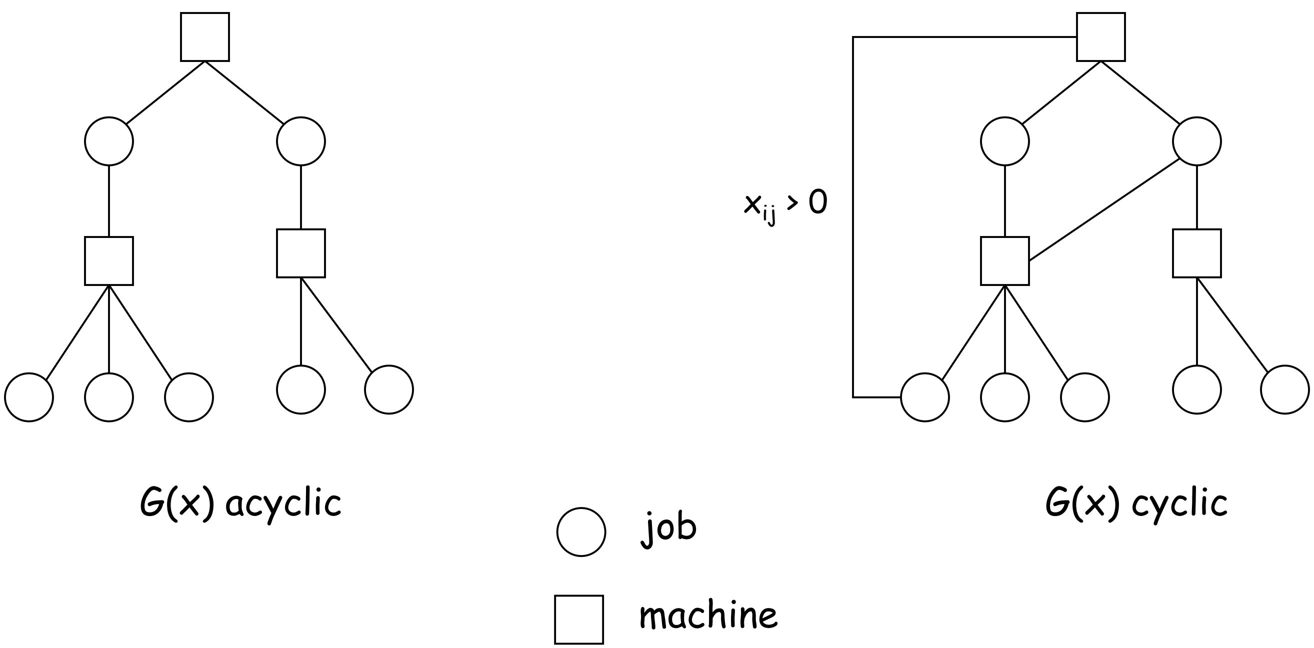
Probem: How can we do the rounding?

Generalized Load Balancing: Structure of LP Solution

Lemma 3. Let x be solution to LP. Let $G(x)$ be the graph with an edge from machine i to job j if $x_{ij} > 0$. Then $G(x)$ is acyclic.

can transform x into another LP solution where $G(x)$ is acyclic if LP solver doesn't return such an x

Pf. (deferred)



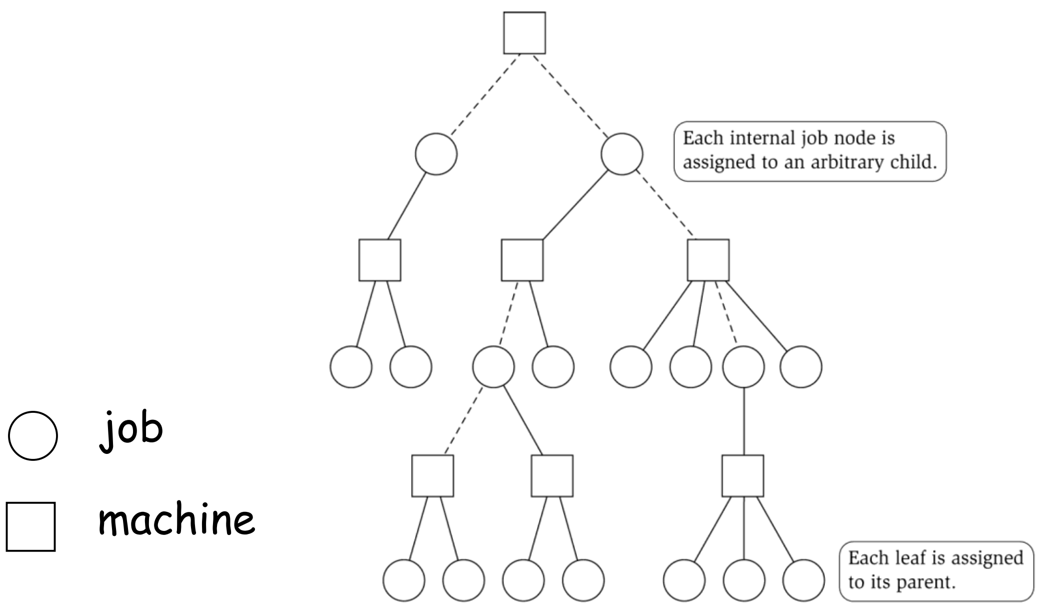
Generalized Load Balancing: Rounding

Rounded solution. Find LP solution x where G(x) is a forest. Root forest G(x) at some arbitrary machine node r.

- If job j is a leaf node, assign j to its parent machine i.
- If job j is not a leaf node, assign j to one of its children.

Lemma 4. Rounded solution only assigns jobs to authorized machines.

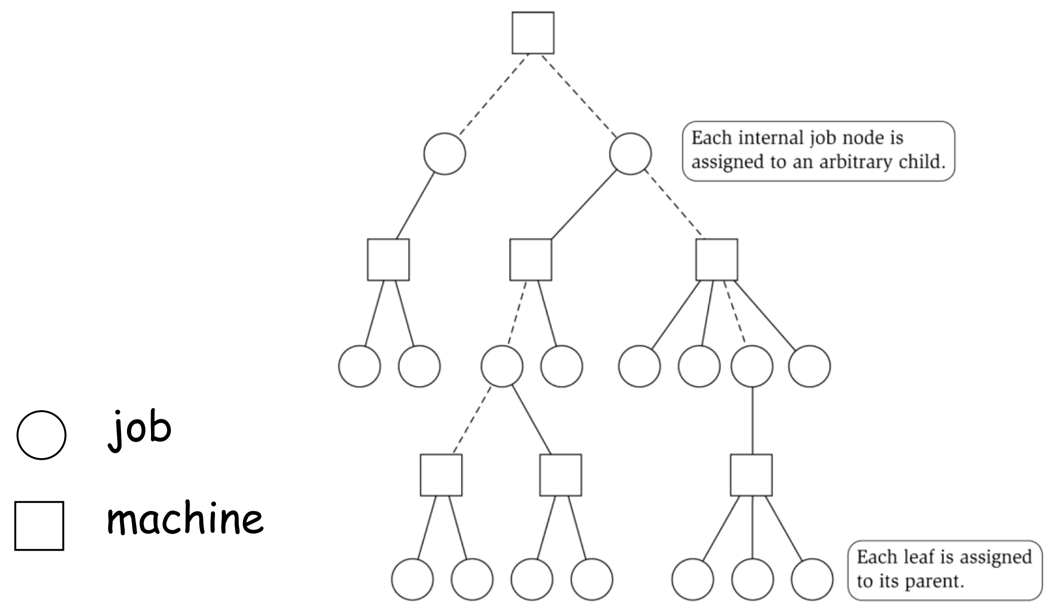
Pf. If job j is assigned to machine i, then $x_{ij} > 0$. LP solution can only assign positive value to authorized machines. ▪



Generalized Load Balancing: Analysis

Lemma 5. If job j is a leaf node and machine $i = \text{parent}(j)$, then $x_{ij} = t_j$.
Pf. Since j is a leaf, $x_{ij} = 0$ for all $i \neq \text{parent}(j)$. LP constraint guarantees $\Sigma_i\, x_{ij} = t_j$. ▪

Lemma 6. At most one non-leaf job is assigned to a machine.
Pf. The only possible non-leaf job assigned to machine i is parent(i). ▪



Generalized Load Balancing: Analysis

Theorem. [Lenstra, Shmoys, Tardos 1990] Rounded solution is a 2-approximation.

Pf.

- Let J(i) be the jobs assigned to machine i.
- By Lemma 5+6, the load L_i on machine i has two components:

  - leaf nodes

$$\sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} t_j \overset{\text{Lemma 5}}{=} \sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} x_{ij} \leq \sum_{j \in J} x_{ij} \overset{\text{LP}}{\leq} \underbrace{L}_{\text{optimal value of LP}} \overset{\text{Lemma 1 (LP is a relaxation)}}{\leq} L^*$$

  - parent(i)

$$t_{\text{parent}(i)} \overset{\text{Lemma 2}}{\leq} L^*$$

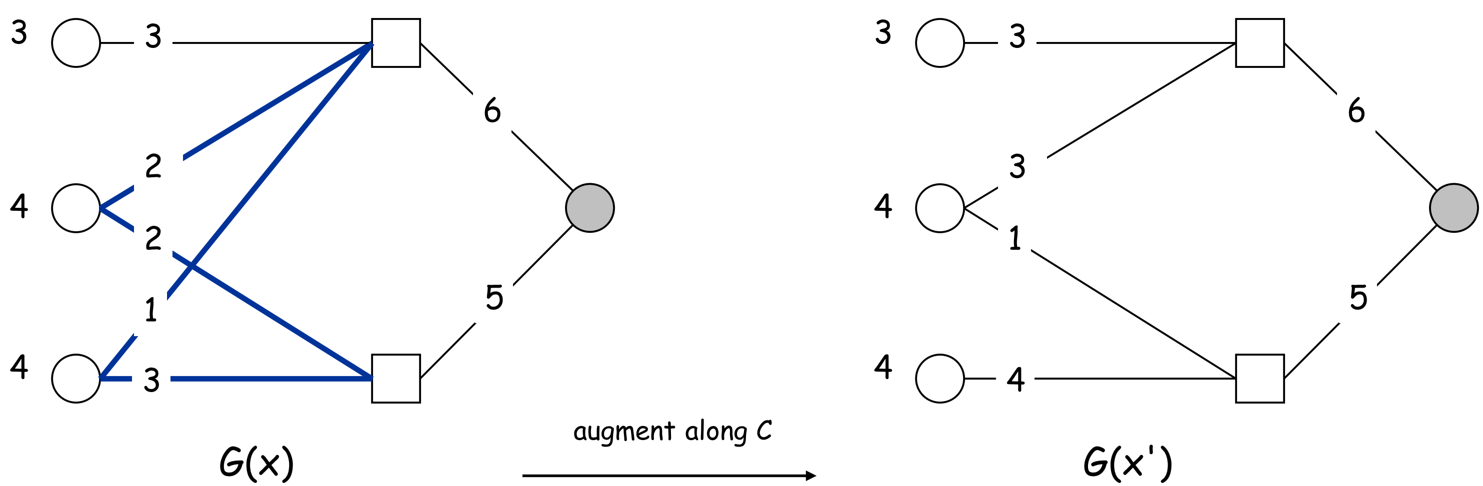
- Thus, the overall load $L_i \leq 2L^*$. ▪

Generalized Load Balancing: Structure of Solution

Lemma 3. Let (x, L) be solution to LP. Let $G(x)$ be the graph with an edge from machine i to job j if $x_{ij} > 0$. We can find another solution (x', L) such that $G(x')$ is acyclic.

Pf. Let C be a cycle in $G(x)$.

- Augment flow along the cycle C. ← flow conservation maintained
- At least one edge from C is removed (and none are added).
- Repeat until $G(x')$ is acyclic.



Conclusions

Running time. The bottleneck operation in our 2-approximation is solving one LP with $mn + 1$ variables.

Remark. Can solve LP using flow techniques on a graph with $m+n+1$ nodes: given L, find feasible flow if it exists. Binary search to find L^*.

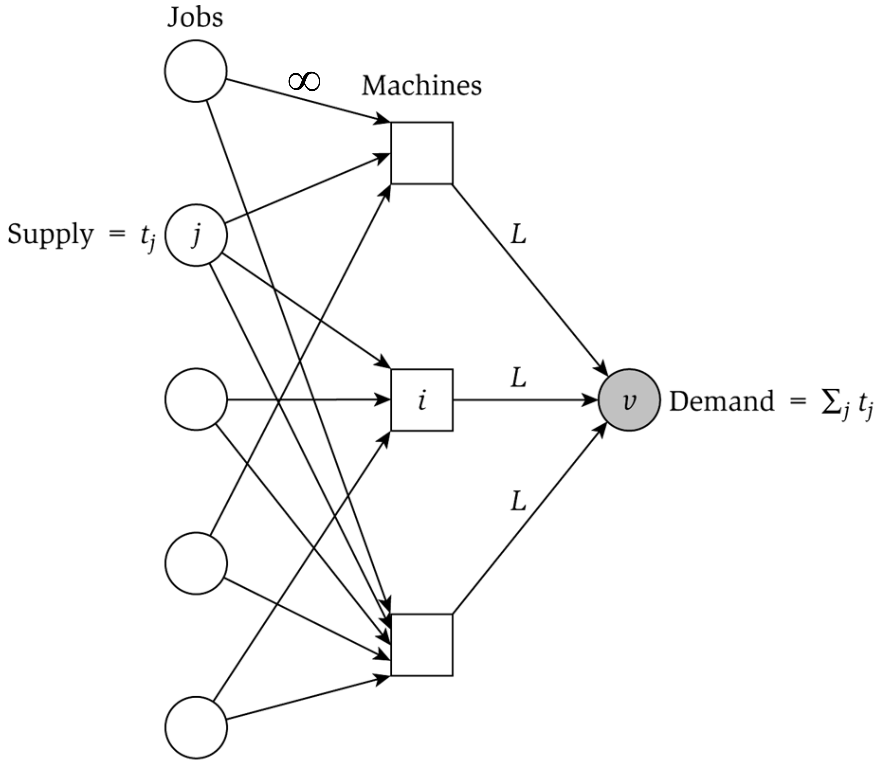
Extensions: unrelated parallel machines. [Lenstra-Shmoys-Tardos 1990]

- Job j takes t_{ij} time if processed on machine i.
- 2-approximation algorithm via LP rounding.
- No 3/2-approximation algorithm unless $P = NP$.

Generalized Load Balancing: Flow Formulation

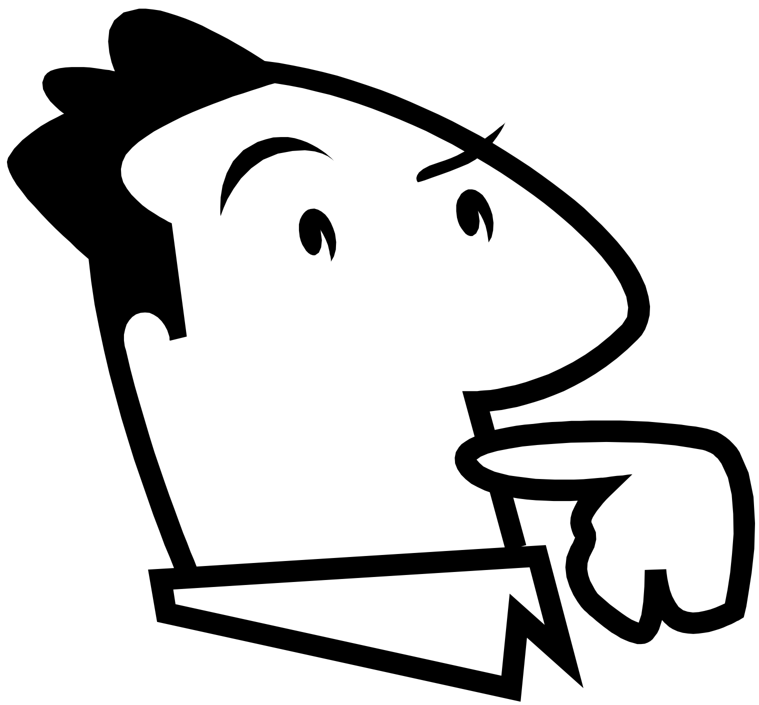
Flow formulation of LP.

$$\begin{array}{llll} \sum_i x_{ij} & = & t_j & \text{for all } j \in J \\ \sum_j x_{ij} & \leq & L & \text{for all } i \in M \\ x_{ij} & \geq & 0 & \text{for all } j \in J \text{ and } i \in M_j \\ x_{ij} & = & 0 & \text{for all } j \in J \text{ and } i \notin M_j \end{array}$$



Observation. Solution to feasible flow problem with value L are in one-to-one correspondence with LP solutions of value L.

Fragen?



Kapitel 4: Approximation Algorithms

Inhalt:

- Greedy Techniques
  - Load-Balancing Problem
  - Center Selection Problem
- Pricing Method
  - Vertex Cover Problem
- Linear Programming and Rounding
  - Vertex Cover Problem
  - Generalized Load-Balancing Problem
- Polynomial Time Approximation Scheme
  - Knapsack Problem

Polynomial Time Approximation Scheme

PTAS. $(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.

- Load balancing. [Hochbaum-Shmoys 1987]
- Euclidean TSP. [Arora 1996, Mitchell 1996]

Consequence. PTAS produces arbitrarily high quality solution, but trades off accuracy for time.

This section. PTAS for knapsack problem via rounding and scaling.

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i has value v_i > 0 and weights w_i > 0. ← we'll assume $w_i \leq W$
- Knapsack can carry weight up to W.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Problem: Dynamic Programming 1

Def. OPT(i, w) = max value subset of items 1,..., i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best of 1, …, i-1 using up to weight limit w
- Case 2: OPT selects item i.
  - new weight limit = $w - w_i$
  - OPT selects best of 1, …, i-1 using up to weight limit $w - w_i$

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{\, OPT(i-1, w),\ \ v_i + OPT(i-1, w-w_i)\,\} & \text{otherwise} \end{cases}$$

Running time. O(n W).

- W = weight limit.
- Not polynomial in input size!

Knapsack Problem: Dynamic Programming II

Def. OPT(i, v) = min weight subset of items 1, …, i that yields value exactly v.

- Case 1: OPT does not select item i.
  - OPT selects best of 1, …, i-1 that achieves exactly value v
- Case 2: OPT selects item i.
  - consumes weight w_i, new value needed = $v - v_i$
  - OPT selects best of 1, …, i-1 that achieves exactly value $v - v_i$

$$OPT(i,v) = \begin{cases} 0 & \text{if } v = 0 \\ \infty & \text{if } i = 0,\ v > 0 \\ OPT(i-1, v) & \text{if } v_i > v \\ \min\{ OPT(i-1, v),\ \ w_i + OPT(i-1, v - v_i) \} & \text{otherwise} \end{cases}$$

$V^* \leq n\, v_{max}$

Running time. $O(n V^*) = O(n^2 v_{max})$.

- V^* = optimal value = maximum v such that $OPT(n, v) \leq W$.
- Not polynomial in input size!

Knapsack: FPTAS

Intuition for approximation algorithm.

- Round all values up to lie in smaller range. $v_i \rightarrow \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$
- Run dynamic programming algorithm on rounded instance. $\hat{v}_i$
- Return optimal items in rounded instance.

Item	Value	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

W = 11

original instance

$\theta = 100{,}000$

Item	Value	Weight
1	2	1
2	7	2
3	19	5
4	223	6
5	284	7

W = 11

rounded instance

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$ $\quad \bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta$

- v_{max} = largest value in original instance
- ε = precision parameter
- θ = scaling factor = $\varepsilon \, v_{max} / n$

Observation. Optimal solution to problems with $\bar{v}$ or $\hat{v}$ are equivalent.

Intuition. $\bar{v}$ close to v, so optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ small and integral, so dynamic programming algorithm is fast.

Running time. $O(n^3 / \varepsilon)$.

- Dynamic program II running time is $O(n^2 \, \hat{v}_{\max})$, where

$$\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$ $\quad \bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta$ $\quad v_i \le \bar{v}_i \le v_i + \theta$

Theorem. If S is solution found by our algorithm and S* is any other feasible solution then $(1+\varepsilon) \sum_{i \in S} v_i \ \ge \ \sum_{i \in S^*} v_i$

Pf. Let S* be any feasible solution satisfying weight constraint.

$$
\begin{aligned}
\sum_{i \in S^*} v_i &\le \sum_{i \in S^*} \bar{v}_i && \text{always round up} \\
&\le \sum_{i \in S} \bar{v}_i && \text{solve rounded instance optimally} \\
&\le \sum_{i \in S} (v_i + \theta) && \text{never round up by more than } \theta \\
&\le \sum_{i \in S} v_i + n\theta && |S| \le n \\
&\le (1+\varepsilon) \sum_{i \in S} v_i && n\,\theta = \varepsilon\, v_{max} \text{ and } \underbrace{v_{max}}_{\text{DP alg can take } v_{max}} \le \Sigma_{i \in S}\, v_i
\end{aligned}
$$

Fragen?