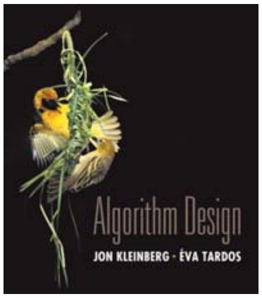


Approximation Algorithms

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

ρ-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is!

11.1 Load Balancing

Load Balancing

Input. m identical machines; n jobs; job j has processing time t_j.
- Job j must run contiguously on one machine.
- A 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 $L = \max_i L_i$.

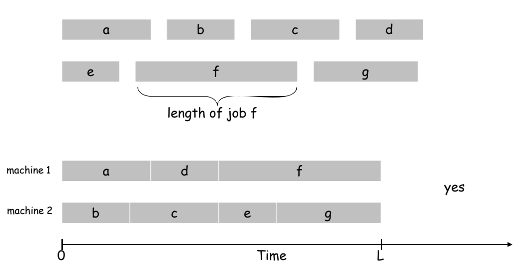
Load balancing. Assign each job to a machine to minimize the makespan.

Decision Version. Is the makespan bound by a number K?

Load Balancing on 2 Machines

Claim. Load balancing is hard even if only 2 machines.
Pf. NUMBER-PARTITION $\leq_P$ LOAD-BALANCE.

NP-complete



Load Balancing: Greedy Scheduling

Greedy-scheduling algorithm.
- Consider n jobs in some fixed order.
- Assign job j to machine whose load is smallest so far.

```
Greedy-Scheduling(m, n, t1,t2,…,tn) {
   for i = 1 to m {
      Li ← 0          ← load on machine i
      J(i) ← φ        ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink { Lk }     ← machine i has smallest load
      J(i) ← J(i) ∪ {j}      ← assign job j to machine i
      Li ← Li + tj           ← update load of machine i
   }
   return J(1), …, J(m)
}
```

Implementation: O(n log m) using a priority queue.

ρ-approximation

An algorithm for an optimization problem is a ρ-approximation if the solution found by the algorithm is always within a factor ρ of the optimal solution.

Minimization Problem: ρ = approximate-solution/optimal-solution

Maximization Problem: ρ = optimal-solution/approximate-solution

In general, $1 \le \rho$. If $\rho = 1$, then the solution is optimal.

Load Balancing: Greedy Scheduling Analysis

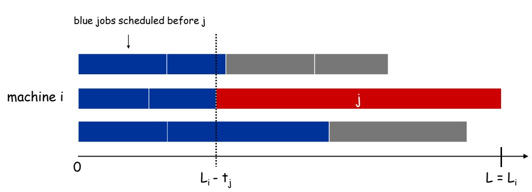
Theorem. [Graham, 1966] Greedy algorithm is a 2-approximation.
- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L*.

Lemma 1. The optimal makespan $L^* \ge \max_j t_j$.
Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. The optimal makespan $L^* \ \ge\ \frac{1}{m}\sum_j t_j$.
Pf.
- The total processing time is $\Sigma_j t_j$.
- One of m machines must do at least a 1/m fraction of total work. ▪

Load Balancing: Greedy Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.
Pf. Consider max load L_i of bottleneck machine i.
- Let j be the last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \ \Rightarrow\ L_i - t_j \le L_k$ for all $1 \le k \le m$.



Load Balancing: Greedy Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.
Pf. Consider max load L_i of bottleneck machine i.
- Let j be the last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \ \Rightarrow\ L_i - t_j \le L_k$ for all $1 \le k \le m$.
- Sum inequalities over all k and divide by m:

$$\begin{aligned} L_i - t_j &\le \tfrac{1}{m}\textstyle\sum_k L_k \\ &= \tfrac{1}{m}\textstyle\sum_k t_k \\ \text{Lemma 2} \longrightarrow \quad &\le L^* \end{aligned}$$

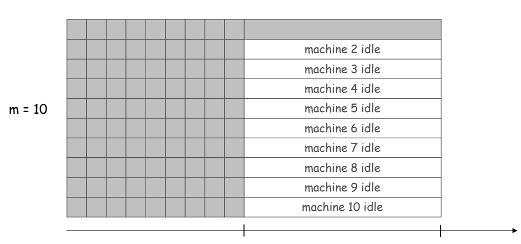
- Now $L_i = \underbrace{(L_i - t_j)}_{\le L^*} + \underbrace{t_j}_{\le L^*} \le 2L^*$. ▪

$\uparrow$ Lemma 1

Load Balancing: Greedy Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

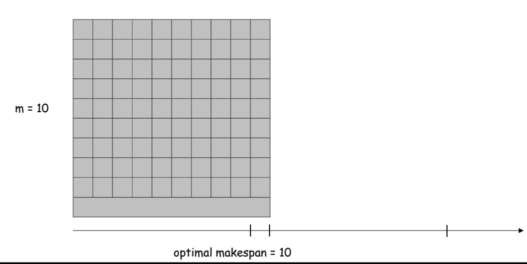
Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m
Greedy solution = 2m-1;



Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m
- Greedy solution = 2m-1; Optimal makespan = m; ρ = 2-1/m.



Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run Greedy scheduling algorithm.

```
LPT-Greedy-Scheduling(m, n, t1,t2,…,tn) {
   Sort jobs so that t1 ≥ t2 ≥ … ≥ tn

   for i = 1 to m {
      Li ← 0          ← load on machine i
      J(i) ← φ        ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk          ← machine i has smallest load
      J(i) ← J(i) ∪ {j}       ← assign job j to machine i
      Li ← Li + tj            ← update load of machine i
   }
   return J(1), …, J(m)
}
```


Load Balancing: LPT Rule

Observation. If at most m jobs, then greedy-scheduling is optimal.
Pf. Each job put on its own machine. ▪

Lemma 3. If there are more than m jobs, $L^* \geq 2\, t_{m+1}$.
Pf.
- Consider first m+1 jobs $t_1, \ldots, t_{m+1}$.
- Since the t_i's are in descending order, each takes at least t_{m+1} time.
- There are m+1 jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. ▪

Theorem. LPT rule is a 3/2 approximation algorithm.
Pf. Same basic approach as for the first greedy scheduling.

$$L_i \;=\; \underbrace{(L_i - t_j)}_{\leq L^*} \;+\; \underbrace{t_j}_{\leq \frac{1}{2}L^*} \;\leq\; \tfrac{3}{2}L^*. \quad ▪$$

Lemma 3
(by observation, can assume number of jobs > m)

Load Balancing: LPT Rule

Q. Is our 3/2 analysis tight?
A. No.

Theorem. [Graham, 1969] LPT rule is a 4/3-approximation.
Pf. More sophisticated analysis of the same algorithm.

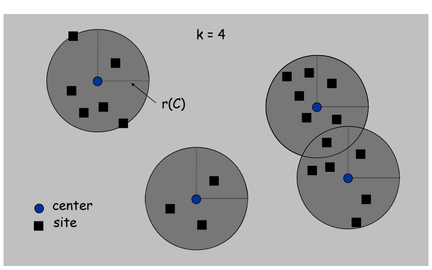
Q. Is Graham's 4/3 analysis tight?
A. Essentially yes.

11.2 Center Selection

Center Selection Problem

Input. Set of n sites $s_1, \ldots, s_n$ and integer k > 0.

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.



Center Selection Problem

Input. Set of n sites $s_1, \ldots, s_n$ and integer k > 0.

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.

Notation.
- dist(x, y) = distance between x and y.
- $dist(s_i, C) = \min_{c \in C} dist(s_i, c)$ = distance from s_i to closest center.
- $r(C) = \max_i dist(s_i, C)$ = smallest covering radius.

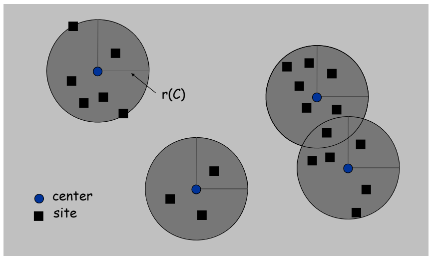
Goal. Find set of centers C that minimizes r(C), subject to |C| = k.

Distance function properties.
- dist(x, x) = 0 (identity)
- dist(x, y) = dist(y, x) (symmetry)
- dist(x, y) ≤ dist(x, z) + dist(z, y) (triangle inequality)

Center Selection Example

Ex: each site is a point in the plane, a center can be any point in the plane, dist(x, y) = Euclidean distance.

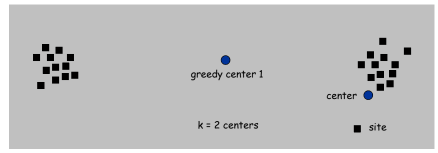
Remark: search can be infinite!



Greedy Algorithm: A False Start

Greedy algorithm. Put the first center at the best possible location for a single center, and then keep adding centers so as to reduce the covering radius each time by as much as possible.

Remark: arbitrarily bad!



Center Selection: Greedy Algorithm

Greedy algorithm. Repeatedly choose the next center to be the site farthest from any existing center.

```
Greedy-Center-Selection(k, n, s1,s2,…,sn) {

   C = { s1 }
   repeat k-1 times {
      Select a site si with maximum dist(si, C)
      Add si to C
   }                          site farthest from any center
   return C
}
```

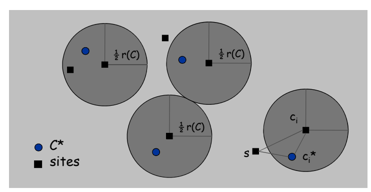
Observation. Upon termination all centers in C are pairwise at least $r(C)$ apart.
Pf. By construction of algorithm.

Center Selection: Analysis of Greedy Algorithm

Theorem. Let $C^* = \{c_i^*\}$ be an optimal set of centers. Then $r(C) \le 2r(C^*)$.
Pf. (by contradiction) Assume $r(C^*) < \frac{1}{2} r(C)$.

- For each site c_i in C, consider ball of radius $\frac{1}{2} r(C)$ around it.
- Exactly one c_i^* in each ball; let c_i be the site paired with c_i^*.
- Consider any site s and its closest center c_i^* in C^*.
- $\text{dist}(s, C) \le \text{dist}(s, c_i) \le \text{dist}(s, c_i^*) + \text{dist}(c_i^*, c_i) \le 2r(C^*)$.
- Thus $r(C) \le 2r(C^*)$. ▪

$\le r(C^*)$ since c_i^* is closest center



Center Selection

Theorem. Let C^* be an optimal set of centers. Then $r(C) \le 2r(C^*)$.

Theorem. Greedy algorithm is a 2-approximation for center selection problem.

Remark. Greedy algorithm always places centers at sites, but is still within a factor of 2 of best solution that is allowed to place centers anywhere.

e.g., points in the plane

Question. Is there hope of a 3/2-approximation? 4/3?

Theorem. Unless P = NP, there no ρ-approximation for center-selection problem for any $\rho < 2$.

Center Selection: Hardness of Approximation

Theorem. Unless P = NP, there is no ρ-approximation algorithm for metric k-center problem for any $\rho < 2$.

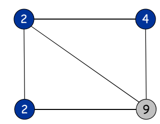
Pf. We show how we could use a $(2 - \varepsilon)$ approximation algorithm for k-center to solve DOMINATING-SET in poly-time.

- Let $G = (V, E)$, k be an instance of DOMINATING-SET. ← see Exercise 8.29
- Construct instance G' of k-center with sites V and distances
  - $d(u, v) = 1$ if $(u, v) \in E$
  - $d(u, v) = 2$ if $(u, v) \notin E$
- Note that G' satisfies the triangle inequality.
- Claim: G has dominating set of size k iff there exists k centers C^* with $r(C^*) = 1$.
- Thus, if G has a dominating set of size k, a $(2 - \varepsilon)$-approximation algorithm on G' must find a solution C^* with $r(C^*) = 1$ since it cannot use any edge of distance 2.

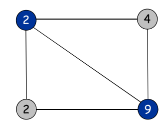
11.4 The Pricing Method: Weighted Vertex Cover

Weighted Vertex Cover

Weighted vertex cover. Given a graph G with vertex weights, find a vertex cover of minimum weight.



weight = 2 + 2 + 4



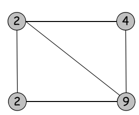
weight = 11

Pricing Method

Pricing method. Each edge must be covered by some vertex.
Edge e = (i, j) pays price $p_e \geq 0$ to use vertex i and j.

Fairness. Edges incident to vertex i should pay $\leq w_i$ in total.

$$\text{for each vertex } i: \sum_{e=(i,j)} p_e \leq w_i$$



Lemma. For any vertex cover S and any fair prices p_e: $\sum_e p_e \leq w(S)$.

Pf. ▪

$$\sum_{e \in E} p_e \;\leq\; \sum_{i \in S} \sum_{e=(i,j)} p_e \;\leq\; \sum_{i \in S} w_i \;=\; w(S).$$

each edge e covered by at least one node in S

sum fairness inequalities for each node in S

Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pe = 0
                                                    Σ_{e=(i,j)} p_e = w_i
   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pe as much as possible until i or j tight
   }

   S ← set of all tight nodes
   return S
}
```


Pricing Method

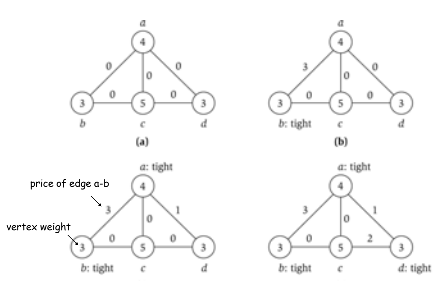


Figure 11.8

Pricing Method: Analysis

Theorem. Pricing method is a 2-approximation.
Pf.

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.
- Let S = set of all tight nodes upon termination of algorithm. S is a vertex cover: if some edge i-j is uncovered, then neither i nor j is tight. But then while loop would not terminate.
- Let S* be optimal vertex cover. We show w(S) ≤ 2w(S*).

$$w(S) = \sum_{i \in S} w_i \;=\; \sum_{i \in S} \sum_{e=(i,j)} p_e \;\leq\; \sum_{i \in V} \sum_{e=(i,j)} p_e \;=\; 2 \sum_{e \in E} p_e \;\leq\; 2w(S^*). \quad ▪$$

all nodes in S are tight

S ⊆ V, prices ≥ 0

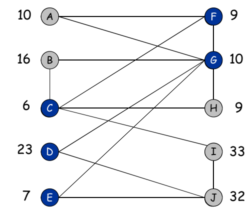
each edge counted twice

fairness lemma

11.6 LP Rounding: Weighted Vertex Cover

Weighted Vertex Cover

Weighted vertex cover. Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.



total weight = 55

32

Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.

Integer programming formulation.
- Model inclusion of each vertex i using a 0/1 variable x_i.

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

Vertex covers in 1-1 correspondence with 0/1 assignments: $S = \{i \in V : x_i = 1\}$

- Objective function: minimize $\Sigma_i w_i x_i$.
- For each edge (i, j), must take either i or j: $x_i + x_j \geq 1$.

33

Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Integer programming formulation.

$$\begin{array}{llllll} (ILP) & \min & \sum_{i \in V} w_i x_i & & & \\ & \text{s.t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\ & & x_i & \in & \{0,1\} & i \in V \end{array}$$

Observation. If x^* is optimal solution to (ILP), then $S = \{i \in V : x^*_i = 1\}$ is a minimum weight vertex cover.

34

Integer Programming

INTEGER-PROGRAMMING. Given integers a_{ij} and b_i, find integers x_j that satisfy:

$$\begin{array}{llll} \max & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & & \text{integral} \end{array}$$

$$\begin{array}{rlll} \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ x_j & \geq & 0 & 1 \leq j \leq n \\ x_j & & \text{integral} & 1 \leq j \leq n \end{array}$$

Observation. Vertex cover formulation proves that integer programming is NP-hard.

even if all coefficients are 0/1 and at most two variables per inequality

35

Linear Programming

Linear programming. Max/min linear objective function subject to linear inequalities.
- Input: integers c_j, b_i, a_{ij}.
- Output: real numbers x_j.

$$\begin{array}{llllll} (P) & \max & c^t x & & \\ & \text{s.t.} & Ax & \geq & b \\ & & x & \geq & 0 \end{array}$$

$$\begin{array}{lllllll} (P) & \max & \sum_{j=1}^{n} c_j x_j & & & \\ & \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ & & x_j & \geq & 0 & 1 \leq j \leq n \end{array}$$

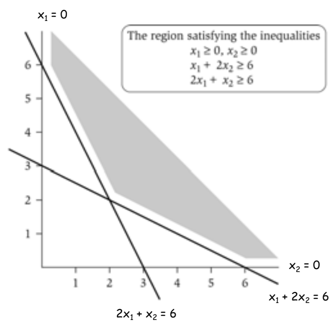
Linear. No x^2, xy, $\arccos(x)$, $x(1-x)$, etc.

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.
Ellipsoid algorithm. [Khachian 1979] Can solve LP in poly-time.

36

LP Feasible Region

LP geometry in 2D.



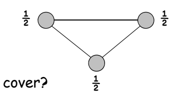
Weighted Vertex Cover: LP Relaxation

Weighted vertex cover. Linear programming formulation.

$$
\begin{array}{llrll}
(LP) & \min & \sum_{i \in V} w_i x_i & & \\
& \text{s. t.} & x_i + x_j & \geq 1 & (i,j) \in E \\
& & x_i & \geq 0 & i \in V
\end{array}
$$

Observation. Optimal value of (LP) is $\leq$ optimal value of (ILP).
Pf. LP has fewer constraints.

Note. LP is not equivalent to vertex cover.



Q. How can solving LP help us find a small vertex cover?
A. Solve LP and round fractional values.

Weighted Vertex Cover

Theorem. If x^* is optimal solution to (LP), then $S = \{i \in V : x^*_i \geq \frac{1}{2}\}$ is a vertex cover whose weight is at most twice the min possible weight.

Pf. [S is a vertex cover]
- Consider an edge $(i, j) \in E$.
- Since $x^*_i + x^*_j \geq 1$, either $x^*_i \geq \frac{1}{2}$ or $x^*_j \geq \frac{1}{2}$ $\Rightarrow$ (i, j) covered.

Pf. [S has desired cost]
- Let S^* be optimal vertex cover. Then

$$\sum_{i \in S^*} w_i \underbrace{\geq}_{\text{LP is a relaxation}} \sum_{i \in S} w_i x_i^* \underbrace{\geq}_{x^*_i \geq \frac{1}{2}} \frac{1}{2} \sum_{i \in S} w_i$$

Weighted Vertex Cover

Theorem. 2-approximation algorithm for weighted vertex cover.

Theorem. [Dinur-Safra 2001] If P $\neq$ NP, then no ρ-approximation for $\underbrace{\rho < 1.3607}_{10\sqrt{5} - 21}$, even with unit weights.

Open research problem. Close the gap.

11.8 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]

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

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 weighs $w_i > 0$. $\longleftarrow$ we'll assume $w_i \le W$
- 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 is NP-Complete

KNAPSACK: Given a finite set X, nonnegative weights w_i, nonnegative values v_i, a weight limit W, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \le W$$
$$\sum_{i \in S} v_i \ge V$$

SUBSET-SUM: Given a finite set X, nonnegative values u_i, and an integer U, is there a subset $S \subseteq X$ whose elements sum to exactly U?

Claim. SUBSET-SUM $\le_P$ KNAPSACK.
Pf. Given instance (u_1, …, u_n, U) of SUBSET-SUM, create KNAPSACK instance:

$$v_i = w_i = u_i \qquad \sum_{i \in S} u_i \le U$$
$$V = W = U \qquad \sum_{i \in S} u_i \ge U$$

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

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

Knapsack: PTAS

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

i = 0 or v = 0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
1	0															
2	0															
3	0															
4	0															
5	0															

Item	Value	Weight
1	1	1
2	1	2
3	3	5
4	4	6
5	6	7

W = 11

Knapsack: PTAS

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

i = 1 , v = …

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
1	0	1	×	×	×	×	×	×	×	×	×	×	×	×	×	×
2	0															
3	0															
4	0															
5	0															

Item	Value	Weight
1	1	1
2	1	2
3	3	5
4	4	6
5	6	7

W = 11

Knapsack: PTAS

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

i = 2 , v = ...

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	0	1	3	x	x	x	x	x	x	x	x	x	x	x	x	x
3	0															
4	0															
5	0															

Item	Value	Weight
1	1	1
2	1	2
3	3	5
4	4	6
5	6	7

W = 11

Knapsack: PTAS

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

i = 3 , v = ...

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	0	1	3	x	x	x	x	x	x	x	x	x	x	x	x	x
3	0	1	3	5	6	8	x	x	x	x	x	x	x	x	x	x
4	0															
5	0															

Item	Value	Weight
1	1	1
2	1	2
3	3	5
4	4	6
5	6	7

W = 11

Knapsack: PTAS

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

i = 4, v = ...

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	0	1	3	x	x	x	x	x	x	x	x	x	x	x	x	x
3	0	1	3	5	6	8	x	x	x	x	x	x	x	x	x	x
4	0	1	3	5	6	7	9	11	12	14	x	x	x	x	x	x
5	0															

Item	Value	Weight
1	1	1
2	1	2
3	3	5
4	4	6
5	6	7

W = 11

Knapsack: PTAS

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

i = 5, v = ...

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	0	1	3	x	x	x	x	x	x	x	x	x	x	x	x	x
3	0	1	3	5	6	8	x	x	x	x	x	x	x	x	x	x
4	0	1	3	5	6	7	9	11	12	14	x	x	x	x	x	x
5	0	1	3	5	6	7	7	8	10	12	13	14	16	18	19	21

Item	Value	Weight
1	1	1
2	1	2
3	3	5
4	4	6
5	6	7

W = 11

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

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

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

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

Knapsack: PTAS

Intuition for approximation algorithm.

- Round all values up to lie in smaller range.
- Run dynamic programming algorithm on rounded instance.
- Return optimal items in rounded instance.

Item	Value	Weight
1	934,221	1
2	5,956,342	2
3	17,810,013	5
4	21,217,800	6
5	27,343,199	7

W = 11

original instance

$\Rightarrow$

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

W = 11

rounded instance

Knapsack: PTAS

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

- 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: PTAS

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

Theorem. If S is solution found by our algorithm and S* is an optimal solution of the original problem, then $(1+\varepsilon) \sum_{i \in S} v_i \ \geq \ \sum_{i \in S^*} v_i$

Pf. Let S* be an optimal solution satisfying weight constraint.

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