11. 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.

\( \rho \)-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio \( \rho \) 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

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 \( L_i = \sum_{j \in J(i)} t_j \).

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 makespan.
Load Balancing: List Scheduling

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

```
List-Scheduling(m, n, t_1, t_2, ..., t_n) {
    for i = 1 to m {
        L_i = 0 ← load on machine i
        J(i) ← φ ← jobs assigned to machine i
    }
    for j = 1 to n {
        i = argmin_k L_k ← machine i has smallest load
        J(i) ← J(i) U {j} ← assign job j to machine i
        L_i ← L_i + t_j ← update load of machine i
    }
}
```

Implementation. $O(n \log n)$ using a priority queue.

Load Balancing - List Scheduling Analysis

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan $L^*$.

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

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

Implementation. $O(n \log n)$ using a priority queue.
Load Balancing: List Scheduling Analysis

**Theorem.** Greedy algorithm is a 2-approximation.

**Pf.** Consider load \( L_i \) of bottleneck machine \( i \).

- Let \( j \) be last job scheduled on machine \( i \).
- When job \( j \) assigned to machine \( i \), \( i \) had smallest load. Its load before assignment is \( L_i - t_j \) \( \Rightarrow L_i - t_j \leq L_k \) for all \( 1 \leq k \leq m \).
- Sum inequalities over all \( k \) and divide by \( m \):

\[
\begin{align*}
L_i - t_j &\leq \frac{1}{m} \sum_i L_i \\
&= \frac{1}{m} \sum_i t_i \\
\text{Lemma 1} &\Rightarrow \leq L^*.
\end{align*}
\]

**Now** \( L_i = (L_i - t_j) + t_j \leq 2L^* \).  

**Q.** Is our analysis tight?

**A.** Essentially yes.

**Ex:** \( m \) machines, \( m(m-1) \) jobs length 1 jobs, one job of length \( m \)

\[\begin{array}{cccccccccccc}
1 & 11 & 21 & 31 & 41 & 51 & 61 & 71 & 81 & 91 \\
2 & 12 & 22 & 32 & 42 & 52 & 62 & 72 & 82 & 92 \\
3 & 13 & 23 & 33 & 43 & 53 & 63 & 73 & 83 & 93 \\
4 & 14 & 24 & 34 & 44 & 54 & 64 & 74 & 84 & 94 \\
5 & 15 & 25 & 35 & 45 & 55 & 65 & 75 & 85 & 95 \\
6 & 16 & 26 & 36 & 46 & 56 & 66 & 76 & 86 & 96 \\
7 & 17 & 27 & 37 & 47 & 57 & 67 & 77 & 87 & 97 \\
8 & 18 & 28 & 38 & 48 & 58 & 68 & 78 & 88 & 98 \\
9 & 19 & 29 & 39 & 49 & 59 & 69 & 79 & 89 & 99 \\
\end{array}\]

\( m = 10 \), list scheduling makespan = 19

Load Balancing: LPT Rule

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

**LPT-List-Scheduling\( (m, n, t_1, t_2, \ldots, t_n) \) \{ \**

- **Sort** jobs so that \( t_i \geq t_j \geq \ldots \geq t_n \)
  
  **for** \( i = 1 \) to \( m \) \{
  \[
  L_i \leftarrow 0 \quad \text{load on machine } i \\
  J(i) \leftarrow \phi \quad \text{jobs assigned to machine } i
  \]
  \}

- **for** \( j = 1 \) to \( n \) \{
  \[
  i = \arg \min_k L_k \quad \text{machine } i \text{ has smallest load} \\
  J(i) \leftarrow J(i) \cup \{j\} \quad \text{assign job } j \text{ to machine } i \\
  L_i \leftarrow L_i + t_j \quad \text{update load of machine } i
  \]
  \}

**}**
Load Balancing: LPT Rule

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

Lemma 3. If there are more than m jobs, L* ≈ 2 tm+1.
Pf.
- Consider first m+1 jobs t1, ..., tm+1:
- Since the t's are in descending order, each takes at least tm+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 list scheduling.

\[
L_i = (L_i - t_j) + \frac{t_j}{L^*} \leq \frac{3}{2} L^*.
\]

\[ \text{by observation, can assume number of jobs } > m \]

11.2 Center Selection

Center Selection Problem

Input. Set of n sites s1, ..., sn.

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 $s_1, ..., s_n$.

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

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

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

**Distance function properties.**
- $\text{dist}(x, x) = 0$ (identity)
- $\text{dist}(x, y) = \text{dist}(y, x)$ (symmetry)
- $\text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y)$ (triangle inequality)

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 Example

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

**Remark:** search can be infinite!

**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, s_1, s_2, ..., s_n) {
    C = \emptyset
    repeat k times {
        Select a site $s_i$ with maximum $\text{dist}(s_i, C)$
        Add $s_i$ to $C$
    }  \text{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 $C^*$ be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.  

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 $C^*$.  
- $\text{dist}(s, C) \leq \text{dist}(s, c_i) \leq \text{dist}(s, c^*_i) + \text{dist}(c^*_i, c_i) \leq 2r(C^*)$.  
- Thus $r(C) \leq 2r(C^*)$. \( \Delta \)-inequality \( r(C^*) \) since $c_i^*$ is closest center

11.4 The Pricing Method: Vertex Cover

Center Selection

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

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 $\rho$-approximation for center-selection problem for any $\rho < 2$.

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 = 9
**Weighted Vertex Cover**

**Pricing method.** Each edge must be covered by some vertex. Edge \( e \) pays price \( p_e > 0 \) to use edge.

**Fairness.** Edges incident to vertex \( i \) should pay \( w_i \) in total.

\[
\text{for each vertex } i: \sum_{e \in \{i,j\}} p_e \leq w_i
\]

**Claim.** For any vertex cover \( S \) and any fair prices \( p_e: \sum e p_e = w(S) \).

**Proof.**

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

- each edge \( e \) covered by at least one node in \( S \)
- sum fairness inequalities for each node in \( S \)

---

**Primal-Dual Algorithm**

**Primal-dual algorithm.** Set prices and find vertex cover simultaneously.

```plaintext
Weighted-Vertex-Cover-Approx(G, w) {
    \text{foreach } e \in E
    \hspace{1em} p_e = 0
    \hspace{1em} \sum_{e \in \{i,j\}} p_e = w_i

    \text{while (3 edge } i-j \text{ such that either } i \text{ or } j \text{ are tight)}
    \hspace{1em} \text{select such an edge } e
    \hspace{1em} \text{increase } p_e \text{ without violating fairness}

    \hspace{1em} S = \text{set of all tight nodes}

    \text{return } S
}
```

---

**11.6 LP Rounding: Vertex Cover**

**Primal Dual Algorithm: Analysis**

**Theorem.** Primal-dual algorithm 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 either \( i \) or \( j \) is not tight. But then while loop would not terminate.

- Let \( S^* \) be optimal vertex cover. We show \( w(S) \leq 2w(S^*) \).

\[
w(S) = \sum_{i \in S} w_i \leq \sum_{i \in S} \sum_{e \in \{i,j\}} p_e \leq \sum_{e \in E} \sum_{e \in \{i,j\}} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*).
\]
**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 $S$.

![Graph example]

**Weighted vertex cover: IP Formulation**

**Weighted vertex cover.** Integer programming formulation.

\[
\text{(ILP)} \quad \begin{align*}
\min & \sum_{i \in V} w_i x_i \\
\text{s.t.} & \quad x_i + x_j \geq 1 \quad (i, j) \in E \\
& \quad x_i \in \{0, 1\} \quad i \in V
\end{align*}
\]

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

**Integer Programming**

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

\[
\begin{align*}
\sum_{j=1}^{n} a_{ij} x_j & \geq b_i \quad 1 \leq i \leq m \\
x_j & \geq 0 \quad 1 \leq j \leq n \\
x_j & \text{ integral } 1 \leq j \leq n
\end{align*}
\]

**Observation.** Vertex cover formulation proves that integer programming is NP-hard search problem (even if all coefficients are 0/1 and at most two nonzeros per inequality).

**Linear Programming**

**Linear programming.** Max/min linear objective function subject to linear inequalities.

- Input: integers $c_j$, $b_i$, $a_{ij}$.
- Output: real numbers $x_j$.

**Simplex algorithm.** [Dantzig 1947] Can solve LP in practice.

**Ellipsoid algorithm.** [Khachiyan 1979] Can solve LP in poly-time.
Weighted Vertex Cover: LP Relaxation

**Weighted vertex cover.** Linear programming formulation.

\[
(LP) \quad \min \sum_{i \in V} w_i x_i \\
\text{s.t.} \quad x_i + x_j \geq 1 \quad (i,j) \in E \\
x_i \geq 0 \quad i \in V
\]

**Observation.** Optimal value of (LP) is \( \leq \) optimal value of (ILP).

**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.

\[
\sum_{i \in S^*} w_i \geq \sum_{i \in S} w_i x^*_i \geq \frac{1}{2} \sum_{i \in S} w_i \\
\text{LP is a relaxation} \quad x^*_i \geq \frac{1}{2}
\]

**11.7 Load Balancing Reloaded**

**Good news.**
- 2-approximation algorithm is basis for most practical heuristics.
  - can solve LP with min cut \( \Rightarrow \) faster
  - primal-dual schema \( \Rightarrow \) linear time (see book)
- PTAS for planar graphs.
- Solvable in poly-time on bipartite graphs using network flow.

**Bad news.** [Dinur-Safra, 2001] If \( P \neq NP \), then no \( \rho \)-approximation for \( \rho < 1.3607 \), even with unit weights.
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 $L_i = \sum_{j \in J(i)} t_j$.

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

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

**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.

**Generalized Load Balancing: Integer Linear Program and Relaxation**

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

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

**LP relaxation.**

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

**Generalized Load Balancing: Structure of LP Solution**

**Lemma 3.** Let $x$ be an extreme point 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.

**Pf.** (we prove contrapositive)
- Let $x$ be a feasible solution to the LP such that $G(x)$ has a cycle.
- Define $y_j = \begin{cases} x_{ij} & (i, j) \in C \\ 0 & (i, j) \notin C \end{cases}$ for all $j \in J$.
- Define $z_j = \begin{cases} x_{ij} & (i, j) \in C \\ 0 & (i, j) \notin C \end{cases}$ for all $j \in J$.
- The variables $y$ and $z$ are feasible solutions to the LP.
- Observe $x = \frac{1}{2}y + \frac{1}{2}z$.
- Thus, $x$ is not an extreme point.

[Diagram of a graph with edges between machines and jobs, illustrating the concept of a cycle in the graph $G(x)$].
Generalized Load Balancing: Rounding

Rounded solution. Find extreme point LP solution $x$. 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

**Theorem.** Rounded solution is a 2-approximation.
**Pf.** Let $J(i)$ be the jobs assigned to machine $i$.
- By Lemma 6, the load $L_i$ on machine $i$ has two components:
  - leaf nodes
    \[
    \sum_{j \in J(i) \text{ leaf}} t_j = \sum_{j \in J(i) \text{ leaf}} x_{ij} t_j \leq \sum_{j \in J} x_{ij} t_j \leq L \leq L^* \]
    \[\text{Lemma 5 (LP is a relaxation)}\]
  - parent($i$)
    \[t_{\text{parent}(i)} \leq L^*\]
    \[\text{Lemma 2}\]
- Thus, the overall load $L_i \leq 2L^*$.

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

**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 $\text{parent}(i)$.

Conclusions

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

**Remark.** Possible to solve LP using max flow techniques. (see text)

**Extensions:** unrelated parallel machines. [Lenstra-Shmoys-Tardos 1990]
- Job $j$ takes $t_j$ time if processed on machine $i$.
- 2-approximation algorithm via LP rounding.
- No 3/2-approximation algorithm unless $P = NP$. 


11.8 Knapsack Problem

Knapsack Problem

- Given n objects and a "knapsack.
- Item i has value $v_i > 0$ and weighs $w_i > 0$. \( \text{we'll assume } w_i = W \)
- Knapsack can carry weight up to W.
- Goal: fill knapsack so as to maximize total value.

**Ex:** \{ 3, 4 \} has value 40.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

$W = 11$

Polynomial Time Approximation Scheme

**PTAS.** \((1 + \varepsilon)\)-approximation algorithm for any constant $\varepsilon > 0$.
- Load balancing. [Hochbaum-Shmoys 1987]
- Euclidean TSP. [Arora 1996]

**FPTAS.** PTAS that is polynomial in input size and $1/\varepsilon$.

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

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

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:

\[
\sum_{i \in S} w_i \leq W \\
\sum_{i \in S} v_i \geq V
\]

**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 \( \leq_p \) KNAPSACK.

**Pf.** Given instance $(u_1, \ldots, u_n, U)$ of SUBSET-SUM, create KNAPSACK instance:

\[
v_i = w_i = u_i \\
V = W = U \\
V = W = U
\]
Knapsack Problem: Dynamic Programming I

**Def.** \(OPT(i, w) = \text{max value subset of items } 1, \ldots, i \text{ with weight limit } w\).
- Case 1: OPT does not select item \(i\).
  - OPT selects best of \(1, \ldots, i-1\) using up to weight limit \(w\)
- Case 2: OPT selects item \(i\).
  - new weight limit = \(w - w_i\)
  - OPT selects best of \(1, \ldots, i-1\) 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.** \(O(nW)\).
- \(W\) = weight limit.
- **Not polynomial** in input size!

Knapsack: FPTAS

**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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>134,221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>656,342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1,810,013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22,217,800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28,343,199</td>
<td>7</td>
</tr>
</tbody>
</table>

\(W = 11\)  
original instance

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>29</td>
<td>7</td>
</tr>
</tbody>
</table>

\(W = 11\)  
rounded instance

Knapsack Problem: Dynamic Programming II

**Def.** \(OPT(i, v) = \text{min weight subset of items } 1, \ldots, i \text{ that yields value exactly } v\).
- Case 1: OPT does not select item \(i\).
  - OPT selects best of \(1, \ldots, i-1\) 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 \(1, \ldots, i-1\) that achieves exactly value \(v\)

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

**Running time.** \(O(nV^*) = O(n^2v_{max})\).
- \(V^*\) = optimal value = maximum \(v\) such that \(OPT(n, v) \leq W\).
- **Not polynomial** in input size!

Knapsack: FPTAS

**Knapsack FPTAS.** Round up all values: \(v_i = \left\lceil \frac{v_i}{\theta} \right\rceil\), \(\hat{v}_i = \left\lfloor \frac{v_i}{\varepsilon} \right\rfloor\)
- \(v_{max}\) = largest value in original instance
- \(\varepsilon\) = precision parameter
- \(\theta\) = scaling factor = \(\varepsilon v_{max} / n\)

**Observation.** Optimal solution to problems with \(\hat{V}\) or \(\hat{V}\) are equivalent.

**Intuition.** \(\hat{V}\) close to \(v\) so optimal solution using \(\hat{V}\) is nearly optimal; \(\hat{V}\) small and integral so dynamic programming algorithm is fast.

**Running time.** \(O(n^2 / \varepsilon)\).
- 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: \[
\tilde{v}_i = \frac{v_i}{\theta} \cdot \theta
\]

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

**Pf.** Let \( S^* \) be any feasible solution satisfying weight constraint.

\[
\begin{align*}
\sum_{i \in S^*} v_i &\leq \sum_{i \in S} \tilde{v}_i & \text{always round up} \\
&\leq \sum_{i \in S} \tilde{v}_i & \text{solve rounded instance optimally} \\
&\leq \sum_{i \in S} (v_i + \theta) & \text{never round up by more than } \theta \\
&\leq \sum_{i \in S} v_i + n\theta & |S| = n \\
&\leq (1+\epsilon) \sum_{i \in S} v_i & \text{DP alg can take } v_{\max}
\end{align*}
\]

---

**Center Selection: Hardness of Approximation**

**Theorem.** Unless \( P = NP \), there is no \((2 - \epsilon)\) approximation algorithm for \( k \)-center problem for any \( \epsilon > 0 \).

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

- Let \( G = (V, E) \), \( k \) be an instance of DOMINATING-SET.
- Construct instance \( G' \) of \( k \)-center with sites \( V \) and distances
  - \( d(u, v) = 2 \) if \( (u, v) \in E \)
  - \( d(u, v) = 1 \) 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 \((2 - \epsilon)\)-approximation algorithm on \( G' \) must find a solution \( C^* \) with \( r(C^*) = 1 \) since it cannot use any edge of distance 2.

---

**Knapsack: State of the Art**

**This lecture.**
- "Rounding and scaling" method finds a solution within a \((1 + \epsilon)\) factor of optimum for any \( \epsilon > 0 \).
- Takes \( O(n^3 / \epsilon) \) time and space.

**Ibarra-Kim (1975), Lawler (1979).**
- Faster FPTAS: \( O(n \log (1 / \epsilon) + 1 / \epsilon^4) \) time.
- Idea: group items by value into "large" and "small" classes.
  - run dynamic programming algorithm only on large items
  - insert small items according to ratio \( v_i / w_i \)
  - clever analysis