Approximation Algorithms: 
IP Practice and Knapsack Problem

CMSC 451, Summer 2009

Integer Programming

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

\[
\begin{align*}
\max \ & c^T x \\
\text{s.t.} \ & Ax \geq b \\
\ & x \text{ integral}
\end{align*}
\]

**Observation.** Vertex cover formulation proves that integer programming is an NP-hard search problem.

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

even if all coefficients are 0/1 and at most two variables 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.
  (Average case: poly-time)


**Example IP Problem: 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.

How do we prove our IP solves the problem? poly-time reduction
Example IP Covering/Packing Duality: Vertex Cover / Independent Set

**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 $S$.

**WEIGHTED INDEPENDENT SET.** Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a maximum weight subset of nodes $S$ such that for each edge at most one of its endpoints is in $S$.

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

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

## 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 accuracy for time.

Example running time: $O(n^{1/\varepsilon})$ where $\varepsilon$ is the given error parameter.

*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 $w_i > 0$. \( \longrightarrow \) 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.

<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>

$W = 11$
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.** $\text{SUBSET-SUM} \leq_p \text{KNAPSACK}$.
**Pf.** Given instance $(u_1, \ldots, u_n, U)$ of $\text{SUBSET-SUM}$, create $\text{KNAPSACK}$ instance:

$$v_i = w_i = u_i$$
$$\sum_{i \in S} u_i \leq U$$
$$V = W = U \sum_{i \in S} v_i \geq U$$

Knapsack Problem: Dynamic Programming 1

**Def.** $OPT(i, w) = \max$ value subset of items $1, \ldots, i$ 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 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.** $O(n V^*) = O(n^2 v_{\max})$.

- $V^*$ = optimal value = maximum $v$ such that $OPT(n, v) \leq W$.
- **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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>934,221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5,956,342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>17,810,013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>21,217,800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>27,343,199</td>
<td>7</td>
</tr>
</tbody>
</table>

<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>

$W = 11$

original instance

rounded instance
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.

Knapsack PTAS. Round up all values: $v_i = \left\lfloor \frac{v_i}{b} \right\rfloor b, \quad \hat{v}_i = \left\lfloor \frac{v_i}{b} \right\rfloor$

- $v_{\text{max}}$ = largest value in original instance
- $\epsilon$ = precision parameter
- $b$ = scaling factor = $\epsilon \cdot v_{\text{max}} / n$

Knapsack-Approx($\epsilon$) {
  $b \leftarrow (\epsilon/(2n)) \cdot v_{\text{max}}$
  $S \leftarrow$ solve Knapsack with values $\hat{v}_i$
  return $S$
}

Example Knapsack

Knapsack PTAS. Round up all values: $v_i = \left\lfloor \frac{v_i}{b} \right\rfloor b, \quad \hat{v}_i = \left\lfloor \frac{v_i}{b} \right\rfloor$

Knapsack-Approx($\epsilon$) {
  $b \leftarrow (\epsilon/(2n)) \cdot v_{\text{max}}$
  $S \leftarrow$ solve Knapsack with values $\hat{v}_i$
  return $S$
}

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>934,221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5,956,342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>17,810,013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>21,217,800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>27,343,199</td>
<td>7</td>
</tr>
</tbody>
</table>

$W = 11$

$\varepsilon = 1/10$

$b = (\epsilon/(2 \cdot 5)) \cdot 27343199$

$= (1/100) \cdot 27343199$

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>66</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>78</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>7</td>
</tr>
</tbody>
</table>

$W = 11$
Knapsack: PTAS

**Knapsack PTAS.** Round up all values: 
\[ \bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil, \quad \hat{v}_i = \left\lfloor \frac{v_i}{\theta} \right\rfloor \]

- \( v_{\text{max}} \) = largest value in original instance
- \( \varepsilon \) = precision parameter
- \( \theta \) = scaling factor = \( \varepsilon \frac{v_{\text{max}}}{n} \)

**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 \( O(n^2\hat{v}_{\text{max}}) \), where

\[ \hat{v}_{\text{max}} = \left\lfloor \frac{v_{\text{max}}}{\theta} \right\rfloor = \left\lfloor \frac{n}{\varepsilon} \right\rfloor \]

---

**Final Exam Topics**

**Greedy Algorithms** (Interval Scheduling, Shortest Path, Minimum Spanning Tree, Union-Find, Clustering, and Huffman Coding)

**Divide and Conquer** (Merge Sort, Inversion Counting, Closest Pair of Points, Multiplication, and Matrix Multiplication)

**Dynamic Programming** (Weighted Interval Scheduling, Knapsack Problem, and Sequence Alignment)

**Network Flow** (Max Flow/Min Cut, Capacity Scaling, Bipartite Matching, Extensions, and Airline Scheduling)

**Intractability** (Poly-time reductions, NP, NP-Complete)

**Approximation Algorithms** (Load Balancing, k-Center Problem, Pricing Method, Integer and Linear Programming)

**Advanced Topics** (next week)

**Registrar’s Problem**