Dynamic Programming

Example I: The Knapsack Problem
The Knapsack Problem

Input: set of $n$ items $I$, values $v_i$, sizes $s_i$, capacity $B$ (all integers)

Task: find $S \subseteq I$ with $\sum_{i \in S} s_i \leq B$, maximizing value $\sum_{i \in S} v_i$
A dynamic program

Idea: store all “good” subsets of \{1, \ldots, j\} in \(A(j)\)
A dynamic program

Idea: store all “good” subsets of \{1, \ldots, j\} in \(A(j)\)

Dominance: \(X \succeq Y \iff s(X) \leq s(Y)\) and \(v(X) \geq v(Y)\)

We don’t need \(Y\) if we have \(X\) ...
Idea: store all “good” subsets of \{1, \ldots, j\} in \(A(j)\)

Dominance: \(X \succeq Y \iff s(X) \leq s(Y) \text{ and } v(X) \geq v(Y)\)

We don’t need \(Y\) if we have \(X\) ...

Algorithm 1 (DP for Knapsack)

1. \(A(0) := \{\emptyset\}\)
2. for \(j := 1\) to \(n\)
   \(A(j) := A(j - 1)\)
   for each \(X \in A(j)\)
     if \(s(X) + s_j \leq B\) then
       add \(X \cup \{j\}\) to \(A(j)\)
     while (\(\exists X, Y \in A(j)\) with \(X \succeq Y\))
       remove \(Y\) from \(A(j)\)
3. return \(X \in A(n)\) maximizing \(v(X)\)
An approximation scheme

**Idea:** make $V$ smaller by scaling all $v_i$ down (and rounding)
Idea: make $V$ smaller by scaling all $v_i$ down (and rounding)

Let’s try to get a $(1 - \varepsilon)$-approximation for some $\varepsilon > 0$. 

Algorithm 2 (FPTAS for Knapsack)

1. $M := \max_i v_i$
2. $\mu := \varepsilon M$
3. $v'_i := \left\lfloor \frac{v_i}{\mu} \right\rfloor$ for all $i \in [n]$
4. Solve instance with $v'_i$ instead of $v_i$, using Algorithm 1.

Polynomial-time Approximation Scheme (PTAS): $(1 - \varepsilon)$-approximation for every $\varepsilon > 0$

Fully Polynomial-time Approximation Scheme (FPTAS): $(1 - \varepsilon)$-approximation for every $\varepsilon > 0$, running time polynomial in encoding and $1/\varepsilon$.
An approximation scheme

**Idea:** make $V$ smaller by scaling all $v_i$ down (and rounding)

Let’s try to get a $(1 - \varepsilon)$-approximation for some $\varepsilon > 0$.

**Algorithm 2 (FPTAS for Knapsack)**

1. $M := \max_i v_i, \quad \mu := \frac{\varepsilon M}{n}$
2. $v'_i := \lfloor v_i / \mu \rfloor$ for all $i \in [n]$
3. Solve instance with $v'$ instead of $v$, using Algorithm 1.
An approximation scheme

Idea: make $V$ smaller by scaling all $v_i$ down (and rounding)

Let’s try to get a $(1 - \varepsilon)$-approximation for some $\varepsilon > 0$.

**Algorithm 2 (FPTAS for Knapsack)**

1. $M := \max_i v_i$, $\mu := \frac{\varepsilon M}{n}$
2. $v'_i := \lfloor v_i / \mu \rfloor$ for all $i \in [n]$  
3. Solve instance with $v'$ instead of $v$, using Algorithm 1.

**Polynomial-time Approximation Scheme (PTAS):**
$(1 - \varepsilon)$-approximation for every $\varepsilon > 0$

**Fully Polynomial-time Approximation Scheme (FPTAS):**
$(1 - \varepsilon)$-approximation for every $\varepsilon > 0$, running time polynomial in encoding and $1/\varepsilon$
Dynamic Programming
Example II: Scheduling on Identical Parallel Machines
**Input:** \( m \) identical machines, 
\( n \) jobs with processing times \( p_1, \ldots, p_n \)

**Task:** assign each job \( j \in [n] \) to a machine \( \sigma(j) \in [m] \)
minimizing \( C_{\text{max}} := \max_{i \in [m]} \sum_{j : \sigma(j)=i} p_j \)
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.

2. For $j := 1$ to $n$
   
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a $4/3$-approximation for $P||C_{\text{max}}$. 
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a $4/3$-approximation for $P\|C_{\text{max}}$. 

Last week ...
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a $4/3$-approximation for $P\|C_{\text{max}}$. 
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   - Assign $j$ to machine $i$ with lowest load.

Theorem
LPT List Scheduling is a $4/3$-approximation for $P||C_{max}$.
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   Assign $j$ to machine $i$ with lowest load.

Theorem
LPT List Scheduling is a $4/3$-approximation for $P\|C_{\text{max}}$. 
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a $4/3$-approximation for $P||C_{\text{max}}$. 
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a $4/3$-approximation for $P||C_{\text{max}}$. 
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a 4/3-approximation for $P\|C_{\text{max}}$. 
Algorithm (Longest Processing Time first)

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to $n$
   Assign $j$ to machine $i$ with lowest load.

Theorem

LPT List Scheduling is a $4/3$-approximation for $P\|C_{\max}$. 
Algorithm $A(k)$

1. Compute optimal schedule for long jobs.
2. For each short job $j$
   
   Assign $j$ to machine $i$ with lowest load.

\[ j' \text{ long: } p_{j'} \geq \frac{1}{km} \sum_{j \in [n]} p_j \]
An approximation scheme

Algorithm $A(k)$

1. Compute optimal schedule for long jobs.
2. For each short job $j$
   Assign $j$ to machine $i$ with lowest load.

$$j' \text{ long: } p_{j'} \geq \frac{1}{km} \sum_{j \in [n]} p_j$$
**Algorithm** \( A(k) \)  

1. Compute optimal schedule for long jobs.
2. For each short job \( j \)
   
   Assign \( j \) to machine \( i \) with lowest load.

\[ j' \text{ long: } p_{j'} \geq \frac{1}{km} \sum_{j \in [n]} p_j \]
An approximation scheme

**Algorithm** $A(k)$

1. Compute optimal schedule for long jobs.
2. For each short job $j$
   
   Assign $j$ to machine $i$ with lowest load.

**Theorem** $A(k)$ is a $(1 + 1/km)$-approximation with running time $O(m + km + n)$. 

**Equation**

$$j' \text{ long: } p_j' \geq \frac{1}{km} \sum_{j \in [n]} p_j$$
Algorithm $A(k)$

1. Compute optimal schedule for long jobs.
2. For each short job $j$
   Assign $j$ to machine $i$ with lowest load.

$j'$ long: $p_{j'} \geq \frac{1}{km} \sum_{j \in [n]} p_j$
Algorithm $A(k)$

1. Compute optimal schedule for long jobs.
2. For each short job $j$
   - Assign $j$ to machine $i$ with lowest load.

\[ j' \text{ long: } p_{j'} \geq \frac{1}{km} \sum_{j \in [n]} p_j \]

**Theorem**

$A(k)$ is a $(1 + 1/k)$-approximation with running time $O(m^{km} + n)$. 
Algorithm $A'(k, T)$

1. Schedule long jobs within $(1 + 1/k)T$
   (or find out that $T < \text{OPT}$ and stop).

2. For each short job $j$
   Assign $j$ to machine $i$ with lowest load.

$j$ long: $p_j \geq \frac{T}{k}$
An approximation scheme

**Algorithm** $A'(k, T)$

1. Schedule long jobs within $(1 + 1/k)T$ (or find out that $T < \text{OPT}$ and stop).
2. For each short job $j$
   - Assign $j$ to machine $i$ with lowest load.

$j$ long: $p_j \geq \frac{T}{k}$

**Theorem 1**

If $A'(k, T)$ computes a schedule, then it has makespan $(1 + 1/k)T$. 
Algorithm $A'(k, T)$

1. Schedule long jobs within $(1 + 1/k)T$ (or find out that $T < \text{OPT}$ and stop).

2. For each short job $j$
   Assign $j$ to machine $i$ with lowest load.

Theorem 1
If $A'(k, T)$ computes a schedule, then it has makespan $(1 + 1/k)T$. 
Subroutine for long jobs

**Algorithm** $B(k, T)$

- **1** $p'_j := \left\lfloor \frac{k^2 p_j}{T} \right\rfloor \cdot \frac{T}{k^2}$ for each $j$
- **2** Compute $m' := \min$ number of machines needed to schedule rounded long jobs within makespan $T$.
- **3** If $m' \leq m$ then return corresponding schedule.
- **4** Otherwise return “failed”.

**Lemma 1**

If $B(k, T)$ computes a schedule, then it has makespan $(1 + 1/k)T$. If $B(k, T)$ returns “failed”, then $\text{OPT} > T$. 

only long jobs: $p_j \geq \frac{T}{k}$
Subroutine for long jobs

**Algorithm** $B(k, T)$

1. $p'_j := \left\lfloor \frac{k^2 p_j}{T} \right\rfloor \cdot \frac{T}{k^2}$ for each $j$
2. Compute $m' := \min$ number of machines needed to schedule rounded long jobs within makespan $T$.
   \[ \rightarrow \text{DP} \]
3. If $m' \leq m$ then return corresponding schedule.
4. Otherwise return “failed”.

**Lemma 1**

If $B(k, T)$ computes a schedule, then it has makespan $(1 + 1/k)T$. If $B(k, T)$ returns “failed”, then $\text{OPT} \geq T$.

**Lemma 2**

$B(k, T)$ runs in time $O(n^{k^2}(k + 1)^{k^2})$. 
Idea:

- only $k^2$ different job types ($p_j' \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}$)
Idea:

- only $k^2$ different job types ($p'_j \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}$)
- $n_i$: number of jobs of type $i$
Dynamic program: Idea

Idea:

- only $k^2$ different job types ($p_j' \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}$)
- $n_i$: number of jobs of type $i$
- $M(n_1, \ldots, n_{k^2}) := \#$machines needed to schedule instance defined by $n_1, \ldots, n_{k^2}$
Dynamic program: Idea

Idea:

- only $k^2$ different job types ($p'_j \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}$)
- $n_i$: number of jobs of type $i$
- $M(n_1, \ldots, n_{k^2}) := \#$machines needed to schedule instance defined by $n_1, \ldots, n_{k^2}$

$\#\text{instances} \leq n^{k^2}$
**Dynamic program: Idea**

**Idea:**

- only $k^2$ different job types ($p_j' \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}$)
- $n_i$: number of jobs of type $i$
- $M(n_1, \ldots, n_{k^2}) := \#\text{machines needed to schedule instance defined by } n_1, \ldots, n_{k^2}$
- $M(n_1, \ldots, n_{k^2}) = 1 + \min_{s \in C} M(n_1 - s_1, \ldots, n_{k^2} - s_{k^2})$
- $C := \text{configurations (schedule } s_i \text{ jobs of type } i \text{ on the machine)}$

$$\#\text{instances} \leq n^{k^2}$$
Dynamic program: Idea

Idea:

- only \( k^2 \) different job types \( (p'_j \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}) \)
- \( n_i \): number of jobs of type \( i \)
- \( M(n_1, \ldots, n_{k^2}) := \# \text{machines needed to schedule instance defined by } n_1, \ldots, n_{k^2} \)
- \( M(n_1, \ldots, n_{k^2}) = 1 + \min_{s \in C} M(n_1 - s_1, \ldots, n_{k^2} - s_{k^2}) \)
- \( C := \text{configurations (schedule } s_i \text{ jobs of type } i \text{ on the machine)} \)

\[ \#\text{instances} \leq n^{k^2} \quad \text{and} \quad |C| \leq (k + 1)^{k^2} \]
Idea:

- only $k^2$ different job types ($p_j' \in \{i \cdot \frac{T}{k^2} : i \in [k^2]\}$)
- $n_i$: number of jobs of type $i$
- $M(n_1, \ldots, n_{k^2}) := \#$ machines needed to schedule instance defined by $n_1, \ldots, n_{k^2}$
- $M(n_1, \ldots, n_{k^2}) = 1 + \min_{s \in C} M(n_1 - s_1, \ldots, n_{k^2} - s_{k^2})$
- $C := \text{configurations (schedule } s_i \text{ jobs of type } i \text{ on the machine)}$

$$\#\text{instances} \leq n^{k^2} \quad \text{and} \quad |C| \leq (k + 1)^{k^2}$$

dynamic program: $n^{k^2}(k + 1)^{k^2}$
Algorithm DP

1. $M(0, \ldots, 0) := 0$, $Q := \{(0, \ldots, 0)\}$
2. while ($Q \neq \emptyset$)
   
   Let $q \in Q$ such that $M(q)$ is minimum.
   
   for each $s \in \mathcal{C}$ with $q_i + s_i \leq n$ for all $i$
   
   if $M(q) + 1 < M(q + s)$ then
     
     $M(q + s) := M(q) + 1$
     
     $Q := Q \cup \{q + s\}$
   
   end if
   
   end for

   $Q := Q \setminus \{q\}$
An approximation scheme

**Algorithm** $A'(k, T)$

1. Schedule long jobs within $(1 + 1/k)T$ (or find out that $T < \text{OPT}$ and stop).
2. For each short job $j$
   Assign $j$ to machine $i$ with lowest load.

$j$ long: $p_j \geq \frac{T}{k} \rightarrow B(k, T)$

**Theorem**
If $A'(k, T)$ computes a schedule, then it has makespan $(1 + 1/k)T$. $A'(k, T)$ runs in time $O(n^{k^2}(k + 1)^{k^2})$. 
An approximation scheme

**Algorithm** $A'(k, T)$

1. Schedule long jobs within $(1 + 1/k)T$
   (or find out that $T < \text{OPT}$ and stop).
2. For each short job $j$
   Assign $j$ to machine $i$ with lowest load.

**Theorem**

If $A'(k, T)$ computes a schedule, then it has makespan $(1 + 1/k)T$. $A'(k, T)$ runs in time $O(n^{k^2} (k + 1)^{k^2})$.

$A'(k, T)$ is a $(1 + 1/k)$-relaxed decision procedure.

Can be turned into a $(1 + 1/k)$-approximation algorithm. (Exercise)

**PTAS**: choose $k := \lceil 1/\varepsilon \rceil$