Every Optimization Problem $P$ is defined in terms of the following 4 important parameters:

- $I_P$: the set of instances of $P$.
- $SOL_P$: a function that associates to any instance of $x$ of $P$ the set of feasible solutions of $x$.
- $m_P$: the measure function defined for pairs $(x,y)$ that for $x$ in $I_P$ and $y$ in $SOL_P(x)$. For each such pair $m_P(x,y)$ provides a positive integer which is the value of the feasible solution $y$.
- $goal_P$ in $\{\text{max, min}\}$ specifies whether it is a maximization or a minimization problem.
NPO

An optimization problem \( \mathcal{P} = (I, \text{SOL}, m, \text{goal}) \) belongs to the class NPO if:
- The set \( I \) is recognizable in polynomial time.
- There exists a polynomial \( q \) such that, given an instance \( x \) in \( I \), for any \( y \) in \( \text{SOL}(x) \), the length of \( y \), \( |y| \), is bounded by \( q(|x|) \) and besides, for any \( y \) such that \( |y| = q(|x|) \), it is decidable in polynomial time whether \( y \) is in \( \text{SOL}(x) \).
- The measure function \( m \) is computable in polynomial time.

Example

Minimum Vertex Cover belongs to NPO:
- The set of instances (any undirected graph) is clearly \textit{recognizable} in polynomial time.
- Since any feasible solution is a subset of the vertex set its size is less than vertex set. Moreover, that a subset of vertices is a vertex cover can be done in polynomial time. (how?).
- Given a feasible solution, we can trivially apply the measure function to it in polynomial time (measuring the size).
Important:

- Theorem: For any optimization problem $P$ in NPO, the corresponding decision problem $P_D$ belongs to NP. (See page 28 of the text book for proof).

Class PO

- An optimization problem $P$ belongs to class PO if it is in NPO and there exists a polynomial-time computable algorithm $A$ that, for any instance $x$ in $I$, returns optimal solution $y$ in $SOL^*(x)$, together with value $m^*(x)$. 
Dynamic Programming (DP):

- An algorithmic technique that in some cases, allows us to reduce the size of the search space while looking for an optimal solution.
- Roughly speaking DP can be applied to any problem for which an optimal solution of the problem can be derived by composing optimal solutions of a limited set of “sub-problems”, regardless of how these solutions have been obtained (*principle of optimality*).
- The Top-Down description of the problem is reduced to Bottom-Up implementation.

FPTAS for Knapsack Problem.

- Remember In Knapsack problem:
  - **Input**: Finite set $X$ of $n$ objects. Object $x_i$ in $X$ has value $p_i$ in $\mathbb{Z}^+$ and size $a_i$, and a positive integer $b$.
  - **Solution**: a subset of item $Y$ such that
    \[
    \sum_{x_i \in Y} a_i \leq b
    \]
  - **Measure ($m$)**: Total value of chosen item in $Y$
    \[
    \sum_{x_i \in Y} p_i
    \]
  - **Objective**: maximize Measure over all Solutions
To Apply DP:

- **Sub-problems:**
  - For any $k$ in $\{1, \ldots, n\}$, and for any $p$ in $\{0, \ldots, P\}$, (where $P = p_1 + \ldots + p_n$), we will consider the problem of finding a subset of $\{x_1, \ldots, x_k\}$ which minimizes the total size among all those subsets having total profit equal to $p$ and total size at most $b$.
  - We denote by $M^*(k, p)$ an optimal solution of this problem and $S^*(k, p)$ the corresponding optimal size, we assume that whenever $M^*(k, p)$ is not defined: $S^*(k, p) = \infty$.

Clearly::

- **Input:** $n$ objects. Object $i$ has weight $w_i$ and value $v_i$, with $w_i \leq C$, $\forall 1 \leq i \leq n$.
- **Goal:** Find a subset $S$ of the $n$ elements that does not exceed capacity $C$.
- **Objective:** maximize $\sum_{i \in S} v_i$.
Clearly!

- $M^*(1,0)$ is empty.
- $M^*(1,p_1) = \{x_1\}$.
- And $M^*(1,p)$ is infinity for every other value of $p$.
- Moreover for any $k$ in $\{2, \ldots, n\}$ and for any value of $p$ with $0 \leq p \leq \sum p_i$,

The following relationship holds (the principle of optimality):

\[
M^*(k,p) = \begin{cases} 
M^*(k-1,p-p_k) \cup \{x_k\} & \text{if conditions in (I) hold} \\
M^*(k-1,p) & \text{Otherwise}
\end{cases}
\]

- The conditions (I) are:
  - $p_k \leq p$,
  - $M^*(k-1,p-p_k)$ is defined,
  - $S^*(k-1,p)$ is at least $S^*(k-1,p-p_k)+a_k$ and,
  - $S^*(k-1,p-p_k)+a_k \leq b$.

i.e.

- The best subset of $\{x_1, \ldots, x_k\}$ that has total profit $p$ is either the subset of $\{x_1, \ldots, x_{k-1}\}$ that has total profit $p-p_k$ plus item $x_k$ or the the best subset of $\{x_1, \ldots, x_{k-1}\}$ that has total profit $p$.
- Since the best subset of $\{x_1, \ldots, x_k\}$ that has total profit $p$ must either contain $x_k$ or not one of these two conditions must be satisfied.
- See algorithm on page 71 of the book.
- The running time will be $O\left(n \sum_{1 \leq i \leq n} p_i\right)$.
Knapsack Approximation Scheme

INPUT: Set $X$ of $n$ items, with values $p_i$, costs $a_i$

OUTPUT: Set $Y \subseteq X$ such that $\sum_{a_i \in Y} a_i \leq b$

Begin

$p_{\text{max}} \leftarrow \text{maximum among values of } p_i$;

$t \leftarrow \left\lceil \log \left( \frac{r-1}{r} \frac{p_{\text{max}}}{n} \right) \right\rceil$

$x' \leftarrow \text{Instance with profits } p'_i = \left\lfloor p_i / 2' \right\rfloor$

$Y \leftarrow \text{Solution returned by previous DP algorithm}$

Return $Y$

End.

Why is it a PTAS

Given an instance of Knapsack, the previous algorithm returns a solution in time $O(rn^3/(r-1))$ whose measure $m_{\text{AS}}(x,r)$ satisfies the inequality $m^*(x)/m_{\text{AS}}(x,r) \leq r$.
Linear Programming

- IS a powerful tool for many optimization problems.
- A linear program on \( n \) real variables \( \{x_1, \ldots, x_n\} \) requires to minimize (or maximize) the value of an objective function defined as a linear combination of variables, subject to \( m \) linear inequalities to equalities.

For example

- Given \( c \) in \( \mathbb{R}^n \) and \( b \) in \( \mathbb{R}^m \) and \( m.n \) real matrix \( A \) we can represent the linear programming problem in the following form:

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{S.T.} & \quad Ax \geq b \\
& \quad x \geq 0
\end{align*}
\]
The General Form

\[
\begin{align*}
\min & \quad c'x \\
\text{S.T.} & \quad Ax \geq b \\
& \quad A'x = b \\
& \quad x_i \geq 0, x_2 \geq 0, \ldots, x_k \geq 0
\end{align*}
\]

- Not all constraints are equality.
- Not all variables are conditioned to be non-negative.
- Vice versa, we can convert all inequalities to equality using surplus variables, and change to unconstrained with the subtraction of two non-negative variables.

Equivalently:

- A LP problem can be formulated as maximization problem of the form:
  \[
  \begin{align*}
  \max & \quad c'x \\
  \text{S.T.} & \quad Ax \leq b \\
  & \quad x \geq 0
  \end{align*}
  \]

- Any maximization LP can be reformulated as a minimization problem of the form.
Primal dual Problems:

- Given a minimization linear program, called primal, it is possible to define a related maximization linear program, called dual as follows:
  - For each constraint in primal, there is a variable in dual,
  - For each variable in the primal there is a constraint in dual:

Primal-Dual Programs:

<table>
<thead>
<tr>
<th>PRIMAL</th>
<th>DUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constr. $i$: $\sum_{j \in \text{sets}} a_{iy} x_j = b_i$</td>
<td>Var $i$: $y_i$ not constrained</td>
</tr>
<tr>
<td>Constr. $i$: $\sum_{j \in \text{sets}} a_{iy} x_j \geq b_i$</td>
<td>Var $i$: $y_i \geq 0$</td>
</tr>
<tr>
<td>Var $j$: $x_j$ not constrained</td>
<td>Constr. $j$: $\sum_{i \in \text{sets}} a_{ji} y_i = c_j$</td>
</tr>
<tr>
<td>Var $j$: $x_j \geq 0$</td>
<td>Constr. $j$: $\sum_{i \in \text{sets}} a_{ji} y_i \leq c_j$</td>
</tr>
<tr>
<td>minimize: $c^T x$</td>
<td>maximize $b^T y$</td>
</tr>
</tbody>
</table>
Duality Theorem

If the primal and dual problems are both feasible then the value of optimal solutions coincide, i.e.:

\[
\min \{ c'x \mid Ax \geq b, x \geq 0 \} = \max \{ b'y \mid y'A \leq c, y \geq 0 \}
\]

Integer Programming

Is very much similar to LP except a restriction that the variables have to be integer values. Here is a sample formulation for a \{0,1\}-integer programming problem:

\[
\begin{align*}
\max \quad & c'x \\
\text{S.T.} \quad & Ax \leq b \\
\text{and} \quad & x \in \{0,1\}^n
\end{align*}
\]
Relaxation of Integer Programs:

- Given an integer LP we can associate a LP, obtained by relaxing the integrality constraints on variables, for example:
  \[
  \begin{align*}
  \text{max} & \quad c^T x \\
  \text{S.T.} & \quad Ax \leq b \\
  & \quad 0 \leq x_i \leq 1, \forall i \in \{1, \ldots, n\}
  \end{align*}
  \]

- Clearly, \( m^*(ILP) \leq m^*(LP) \), and such an approach allows us to establish a bound on the optimal measure of ILP.

- The maximization problem is similar.

Example:

- Consider the weighted version of vertex-cover Problem:
  - There is non-negative weight \( c_i \) associated with every vertex \( v_i \) in the graph.
  - We are looking for a vertex cover (VC) of minimum weight (of vertices in VC).
ILP Formulation

Given a vertex-weighted graph $G$, the VC problem can be formulated as following integer linear program ($ILP_{VC}(G)$):

$$\min \sum_{v_i \in V} c_i x_i$$

S.T. $x_i + x_j \geq 1, \forall (v_i, v_j) \in E$

$x_i \in \{0,1\}, \forall v_i \in V$.

LP relaxation and rounding:

- By removing the integrality condition we will get a $LP_{VC}$ version of VC problem.
- Let $x^*(G)$ denote the optimal solution for $LP_{VC}$.
- Given a solution for this problem we will round up the entries of vector $x^*(G)$, i.e. include all the vertices corresponding to entries with value at least 0.5.
The algorithm

Begin
Let $ILP_{VC}$ be the LP formulation of the problem
Let $LP_{VC}$ be the obtained from $ILP_{VC}$ by relaxing
Let $x^*(G)$ be the optimal solution for $LP_{VC}$

$V' \leftarrow \{v_i | x^*_i(G) \geq 0.5\}$
return $V'$
End

It’s a 2-approximation

- Let $V'$ be the solution returned by the algorithm.
- Observe that $V'$ is Feasible for $ILP_{VC}$.
- $m^*(G) \geq m^*_{LP}(G)$.
- And we have the following chain:

$$\sum_{v_i \in V'} c_i \leq 2 \sum_{v_i \in V} c_i x^*_i(G) = 2m^*_{LP}(G) \leq 2m^*(G).$$