Definitions

- Optimization Problem: Given an Optimization Function and a set of constraints, find an optimal solution.

Optimal Solution: A feasible solution for which the optimization function has the best possible value.

Feasible Solution: Solution that satisfies the constraints.

Example

- Printer problem: The constraint is to print all the jobs nonpreemptively (one at a time), and the objective is to minimize the average finish time.

- Container Loading problem: The constraint is that the container loaded have total weight ≤
the cargo weight capacity, and the objective function is to find a largest set of containers to load.

- Coin Changing: Give change using the least number of coins.

  Greedy Method (Chapter 10.1)

- Attempt to construct an optimal solution in stages.

  At each stage we make a decision that appears to be the best (under some criterion) at the time (local optimum).

  A decision made in one stage is not changed in a later stage, so each decision should assure feasibility

- Greedy criterion: criterion used to make the greedy decision at each stage.
Container Loading

- Large ship is to be loaded with cargo.

Cargo is in equal size containers

Container $i$ has weight $w_i$.

The cargo weight capacity is $c$ (and every $w_i \leq c$).

- Load the ship with maximum number of containers without exceeding the cargo weight capacity.

- Find values $x_i \in \{0, 1\}$ such that

$$\sum_{i=1}^{n} w_i \cdot x_i \leq c,$$

and the optimization function $(\sum_{i=1}^{n} x_i)$ is maximized.
Example: $n = 8$

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>50</td>
<td>90</td>
<td>150</td>
<td>50</td>
<td>20</td>
<td>80</td>
<td>400</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>400</td>
</tr>
</tbody>
</table>

Is it an optimal solution? No! Why? One can take out object 2 and add objects 7 and 8. That would be a better solution.

A solution is optimal iff one cannot trade two objects for one (in the solution) while maintaining feasibility. The “if” holds, but not the “only if”. See the following non-optimal solution.

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>240</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>240</td>
</tr>
</tbody>
</table>
Algorithm

- Load ship in stages, one container per stage.
  
  At each stage we need to decide which container to load.

- Greedy criterion: From the remaining containers, select the one with least weight.

Example

\[\begin{array}{cccccccccc}
w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & c \\
20 & 50 & 50 & 80 & 90 & 100 & 150 & 200 & 400 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 390 \\
\end{array}\]
Correctness Proof

Theorem: The above greedy algorithm generates an optimal set of containers to load.

- Proof Idea: No matter which feasible solution (Y) you start with, it is possible to transform it to the solution generated by the algorithm without decreasing the objective function value.

- Assume without loss of generality (wlog) \( w_1 \leq w_2 \leq \ldots \leq w_n; \)

- Let \( X = (x_1, x_2, \ldots x_n) \) be the solution generated by the algorithm

Let \( Y = (y_1, y_2, \ldots y_n) \) be any feasible solution such that \( \sum w_i y_i \leq c. \)

Transform \( Y \) to \( X \) in several steps without decreasing the objective function value.
- From the way the algorithm works, there is a \( k \in [0, n] \) s.t. \( x_i = 1 \) for \( i \leq k \), and \( x_i = 0 \) for \( i > k \). (i.e., \( X = 1, 1, ..., 1, 0, 0, ..., 0 \)).

- Transformation: Let \( j \) be the smallest integer in \([1, n]\), s.t. \( x_j \neq y_j \).

- So either:

  1. No such \( j \) exists in which case \( Y = X \).
  2. \( j \leq k \) (as otherwise \( Y \) is not feasible).

So, \( x_j = 1 \) and \( y_j = 0 \).

Change \( y_j \) to 1

- If \( Y \) is infeasible then there is an \( l \) in \([j + 1, n]\) s.t. \( y_l = 1 \), \( y_l \) is changed to 0, and the new \( Y \) is feasible (because \( w_j \leq w_l \)).

- No matter what the new \( Y \) is, it has at least as many 1s (or more) as before.

- Apply the transformation until you get \( Y = X \).
Example for Proof

Solution Generated by our algorithm $(X)$.

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>50</td>
<td>50</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Consider the following feasible solution $(Y)$

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>50</td>
<td>50</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Transformations as in the proof of the previous theorem. $(j$ in the proof is 1, then 3, 4, 5, 6).
<table>
<thead>
<tr>
<th>(w_1)</th>
<th>(w_2)</th>
<th>(w_3)</th>
<th>(w_4)</th>
<th>(w_5)</th>
<th>(w_6)</th>
<th>(w_7)</th>
<th>(w_8)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>50</td>
<td>50</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>270</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>320</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>400</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>290</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>390</td>
</tr>
</tbody>
</table>

**Implementation**

- Remaining objects are stored in a heap ordered with respect to their weight (smallest on top of the heap).
- Algorithm talks \(O(n \log n)\) time (\(n\) deletes from a heap with initially \(n\) objects, creating the heap takes \(O(n)\) time (CMPSC 130A)).
- Alg takes \(O(n)\) time if optimal sol has very few \((n/\log n)\) objects?
Linear Time Implementation!!

- Assume all weights are different, if there are repeated weights then a similar algorithm exists for the solution of the problem.
- We use an algorithm (which we will describe when we cover divide-and-conquer algorithms) that finds the middle object of \( n \) objects (i.e., find the element such that there are exactly \( \lceil n/2 \rceil \) objects smaller or equal to it) in \( O(n) \) time.
• Our algorithm works by doing a binary Search type of search on the unsorted weights \( W \).

• Let \( S \) be the smallest \( \lceil n/2 \rceil \) objects (in \( W \)) and let \( t \) be their total weight. There are three cases:

  (1) If \( t > c \) then search for a solution in \( S \) only with the same capacity \( c \).

  (2) If \( t = c \), then add all the objects in \( S \) to the solution and end the procedure;

  (3) Else, add all the elements in \( S \) to the solution and set the the remaining capacity to \( c - t \) and now try to add as many objects as possible from \( W - S \).

• Repeat the above step until there are no objects left.

• Time complexity is

\[
c_1n + c_1n/2 + c_1n/4 + \ldots = c_2n
\]
Example 1

- Suppose that \( W \) is: \( \{6, 10, 8, 15, 22, 19, 5, 9\} \), and \( c = 25 \).

  The middle object is 9, \( S = \{6, 8, 5, 9\} \) and \( t = 28 \). The objects in \( S \) do not fit.

- The new \( W \) is: \( \{6, 8, 9, 5\} \), and \( c = 25 \).

  The middle object is 6, \( S = \{6, 5\} \), and \( t = 11 \). The objects in \( S \) fit and are added to the solution.

- The new \( W \) is: \( \{8, 9\} \), and \( c = 14 \).

  The middle object is 8, \( S = \{8\} \), and \( t = 8 \). The object in \( S \) fit and are added to the solution.

- The new \( W \) is: \( \{9\} \), and \( c = 6 \).

  The middle object is 9, \( S = \{9\} \), and \( t = 9 \). The object in \( S \) does not fit.

- There are no objects left and we are done. The solution are the objects with weight 5, 6 and 8.
Example 2

- Suppose $W$ is: $\{6, 10, 8, 15, 22, 19, 5, 9\}$, and $c = 53$.

  The middle object is 9, $S = \{6, 8, 5, 9\}$, and $t = 28$. The objects in $S$ fit and are added to the solution.

- The new $W$ is: $\{10, 15, 22, 19\}$, and $c = 25$.

  The middle object is 15, $S = \{10 + 15\}$, and $t = 25$. The objects in $S$ fit exactly and the algorithm finishes.

- The solution are the objects with weight 6, 8, 5, 9, 10, and 15.
Deterministic Scheduling

Printer Scheduling with Complete Information

- Problem is identical to the one in Section 10.1.1.
- At time zero there are $n$ tasks to be printed.
- Tasks are denoted by $T_1, T_2, \ldots, T_n$ with execution time requirements $t_1, t_2, \ldots, t_n$
- Once a task starts printing it will continue printing until it terminates (i.e., preemptions are not allowed).
Example: $t_1 = 2, t_2 = 1, t_3 = 4, t_4 = 9$. Two schedules:

Let $f_i(S)$ be the finish time for task $T_i$ in $S$.

The Average Finish Time (AFT) for $S$ is $\frac{1}{n} \sum f_i(S)$.

The AFT for $S_1$ is $\frac{2+16+6+15}{4} = 9.75$, and the AFT for $S_2$ is $\frac{7+5+4+16}{4} = 8.00$.

Objective Function: Find a schedule with minimum AFT.

Shortest Processing Time First (SPT): Assign the tasks to the printer from smallest to largest.
**Theorem:** SPT schedules are optimal wrt AFT.

**Proof:** By contradiction.

Suppose that there is a problem instance $I$ such that schedule $S'$ (which is not an SPT schedule) is an optimal schedule wrt AFT, i.e.,

$$
\frac{\sum_{i} f_i(S')}{n} < \frac{\sum_{i} f_i(SPT)}{n}
$$

Since $S'$ is not an SPT schedule there exist two tasks $(T_i, T_j)$ such that they are scheduled one after the other in $S'$ (first $T_i$ and then $T_j$) such that $t_i > t_j$. 
Construct schedule $S''$ from $S'$ by interchanging task $T_i$, and $T_j$.

\[
\begin{array}{c|c|c}
S' & T_i & T_j \\
\hline
& & \downarrow \\
S' & T_j & T_i \\
\end{array}
\]

Since $t_i > t_j$ we know that $f_i(S') > f_j(S'')$, $f_j(S') = f_i(S'')$, and the finish time of the remaining tasks in both schedules is identical.

Therefore, $\frac{\sum f_i(S')}{n} > \frac{\sum f_i(S'')}{n}$. This contradicts that $S'$ is an optimal schedule.