Algorithm Design Methods

- Greedy method.
- Divide and conquer.
- Dynamic Programming.
- Backtracking.
- Branch and bound.
Some Methods Not Covered

• Linear Programming.
• Integer Programming.
• Simulated Annealing.
• Neural Networks.
• Genetic Algorithms.
• Tabu Search.
Optimization Problem

A problem in which some function (called the optimization or objective function) is to be optimized (usually minimized or maximized) subject to some constraints.
Machine Scheduling

Find a schedule that minimizes the finish time.

- optimization function ... finish time
- constraints
  - each job is scheduled continuously on a single machine for an amount of time equal to its processing requirement
  - no machine processes more than one job at a time
Bin Packing

Pack items into bins using the fewest number of bins.

- optimization function … number of bins
- constraints
  - each item is packed into a single bin
  - the capacity of no bin is exceeded
Min Cost Spanning Tree

Find a spanning tree that has minimum cost.

- optimization function \( \text{sum of edge costs} \)
- constraints
  - must select \( n-1 \) edges of the given \( n \) vertex graph
  - the selected edges must form a tree
Feasible And Optimal Solutions

A **feasible solution** is a solution that satisfies the constraints.

An **optimal solution** is a feasible solution that optimizes the objective/optimization function.
Greedy Method

- Solve problem by making a sequence of decisions.
- Decisions are made one by one in some order.
- Each decision is made using a greedy criterion.
- A decision, once made, is (usually) not changed later.
Machine Scheduling

LPT Scheduling.

• Schedule jobs one by one and in decreasing order of processing time.
• Each job is scheduled on the machine on which it finishes earliest.
• Scheduling decisions are made serially using a greedy criterion (minimize finish time of this job).
• LPT scheduling is an application of the greedy method.
LPT Schedule

- LPT rule does not guarantee minimum finish time schedules.
- \((\text{LPT Finish Time}) / (\text{Minimum Finish Time}) \leq 4/3 - 1/(3m)\)
  where \(m\) is number of machines
- Minimum finish time scheduling is NP-hard.
- In this case, the greedy method does not work.
- The greedy method does, however, give us a good heuristic for machine scheduling.
Container Loading

- Ship has capacity $c$.
- $m$ containers are available for loading.
- Weight of container $i$ is $w_i$.
- Each weight is a positive number.
- Sum of container weights $> c$.
- Load as many containers as is possible without sinking the ship.
Greedy Solution

• Load containers in increasing order of weight until we get to a container that doesn’t fit.

• Does this greedy algorithm always load the maximum number of containers?

• Yes. May be proved using a proof by induction (see text).
Container Loading With 2 Ships

Can all containers be loaded into 2 ships whose capacity is \(c\) (each)?

- Same as bin packing with 2 bins.
  - Are 2 bins sufficient for all items?
- Same as machine scheduling with 2 machines.
  - Can all jobs be completed by 2 machines in \(c\) time units?
- NP-hard.
0/1 Knapsack Problem
0/1 Knapsack Problem

• Hiker wishes to take $n$ items on a trip.
• The weight of item $i$ is $w_i$.
• The items are to be carried in a knapsack whose weight capacity is $c$.
• When sum of item weights $\leq c$, all $n$ items can be carried in the knapsack.
• When sum of item weights $> c$, some items must be left behind.
• Which items should be taken/left?
0/1 Knapsack Problem

- Hiker assigns a profit/value $p_i$ to item $i$.
- All weights and profits are positive numbers.
- Hiker wants to select a subset of the $n$ items to take.
  - The weight of the subset should not exceed the capacity of the knapsack. (constraint)
  - Cannot select a fraction of an item. (constraint)
  - The profit/value of the subset is the sum of the profits of the selected items. (optimization function)
  - The profit/value of the selected subset should be maximum. (optimization criterion)
0/1 Knapsack Problem

Let \( x_i = 1 \) when item \( i \) is selected and let \( x_i = 0 \) when item \( i \) is not selected.

\[
\text{maximize } \sum_{i=1}^{n} p_i x_i
\]

\[
\text{subject to } \sum_{i=1}^{n} w_i x_i \leq c
\]

and \( x_i = 0 \) or \( 1 \) for all \( i \)
Greedy Attempt 1

Be greedy on capacity utilization.
- Select items in increasing order of weight.

\[ n = 2, \quad c = 7 \]
\[ w = [3, 6] \]
\[ p = [2, 10] \]

only item 1 is selected
profit (value) of selection is 2
not best selection!
Greedy Attempt 2

Be greedy on profit earned.
  - Select items in decreasing order of profit.

\( n = 3, \ c = 7 \)
\[ w = [7, 3, 2] \]
\[ p = [10, 8, 6] \]

only item 1 is selected
profit (value) of selection is 10
not best selection!
Greedy Attempt 3

Be greedy on profit density \((p/w)\).

- Select items in decreasing order of profit density.

\(n = 2, \ c = 7\)

\(w = [1, 7]\)

\(p = [10, 20]\)

only item 1 is selected

profit (value) of selection is 10

not best selection!
Greedy Attempt 3

Be greedy on profit density ($p/w$).

- Works when selecting a fraction of an item is permitted
- Select items in decreasing order of profit density, if next item doesn’t fit take a fraction so as to fill knapsack.

$n = 2$, $c = 7$

$w = [1, 7]$

$p = [10, 20]$

item 1 and $6/7$ of item 2 are selected
0/1 Knapsack Greedy Heuristics

- Select a subset with \( \leq k \) items.
- If the weight of this subset is \( > c \), discard the subset.
- If the subset weight is \( \leq c \), fill as much of the remaining capacity as possible by being greedy on profit density.
- Try all subsets with \( \leq k \) items and select the one that yields maximum profit.
Subset Heuristic Examples

• So let’s say your set of items is [1,2,3,4,5,6]
• If k=0, then you just do same as 2 slides ago
• If k=1, you start with one item already in your bag, then do same as 2 slides ago.
• If k=2, you start with two items already in your bag, then do same as 2 slides ago.
Subset Heuristic Examples

• With \( k = 1 \), you start with each of the following: \([1], [2], [3], [4], [5], \text{ and } [6] \).

• With \( k = 2 \), you start with each of the following: \([1,2], [1,3], [1,4], [1,5], [1,6], [2,3], [2,4], [2,5], [2,6], [3,4], [3,5], \text{ etc.} \)

• Throw away any subsets where the weight is greater than the capacity of your bag.

• Otherwise, fill rest based on decreasing profit density.
0/1 Knapsack Greedy Heuristics

- \((\text{best value - greedy value})/\text{(best value)} \leq 1/(k+1)\)

<table>
<thead>
<tr>
<th>(k)</th>
<th>0%</th>
<th>1%</th>
<th>5%</th>
<th>10%</th>
<th>25%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>239</td>
<td>390</td>
<td>528</td>
<td>583</td>
<td>600</td>
</tr>
<tr>
<td>1</td>
<td>360</td>
<td>527</td>
<td>598</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>483</td>
<td>581</td>
<td>600</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of solutions (out of 600) within \(x\)% of best.
0/1 Knapsack Greedy Heuristics

- First sort into decreasing order of profit density.
- There are $O(n^k)$ subsets with at most $k$ items.
- Trying a subset takes $O(n)$ time.
- Total time is $O(n^{k+1})$ when $k > 0$.
- $(\text{best value} - \text{greedy value})/(\text{best value}) \leq 1/(k+1)$