Optimization Problems

- A class of problems in which we are asked to find a **set** (or a **sequence**) of “**items**” that satisfy some constraints and simultaneously optimize (i.e., **maximize** or **minimize**) some **objective function**
Example: Bin Packing
Example: Bin Packing
Example: Bin Packing
Example: Bin Packing
Example: Bin Packing
Example: Bin Packing

Number of bins = 3
Example: Bin Packing

(Optimal Solution) Number of bins = 2
The Greedy Method

- Applied to optimization problems
- Adds items to the solution one-by-one
- Builds up towards the final solution
- No backtracking

- Not necessarily optimal!
Activity Selection

a.k.a. Task Scheduling
Activity Selection Problem

- Given a set of activities $S = \{a_1, a_2, \ldots, a_n\}$ where each activity $i$ has a start time $s_i$ and a finish time $f_i$, where $0 \leq s_i < f_i < \infty$. An activity $a_i$ happens in the half-open time interval $[s_i, f_i)$. Two activities are said to be compatible if they do not overlap. The problem is to find a maximum-size compatible subset, i.e., a one with the maximum number of activities.
Example of Activity Selection

\[ a_{3}[0,6) \]

\[ a_{10}[2,14) \]

\[ a_{1}[1,4) \]

\[ a_{9}[8,12) \]

\[ a_{5}[3,9) \]

\[ a_{4}[5,7) \]

\[ a_{8}[8,11) \]

\[ a_{2}[3,5) \]

\[ a_{7}[6,10) \]

\[ a_{11}[12,16) \]

\[ a_{6}[5,9) \]
A Solution

a3[0,6]

a10[2,14]

a1[1,4]  a9[8,12]

a5[3,9]

a4[5,7]  a8[8,11]

a2[3,5]  a7[6,10]  a11[12,16]

a6[5,9]
A Better Solution

a3[0,6)
a10[2,14)
a1[1,4) a9[8,12)
a5[3,9)
a4[5,7) a8[8,11)
a2[3,5] a7[6,10] a11[12,16)
a6[5,9)
An Optimal (Best) Solution

- $a_3[0,6)$
- $a_{10}[2,14)$
- $a_1[1,4)$
- $a_9[8,12)$
- $a_5[3,9)$
- $a_4[5,7)$
- $a_8[8,11)$
- $a_2[3,5)$
- $a_7[6,10)$
- $a_{11}[12,16)$
- $a_6[5,9)$
Another Optimal Solution

\[ a_3[0,6) \]
\[ a_{10}[2,14) \]
\[ a_{11}[12,16) \]
\[ a_{1}[1,4) \]
\[ a_9[8,12) \]
\[ a_{5}[3,9) \]
\[ a_{4}[5,7) \]
\[ a_8[8,11) \]
\[ a_{2}[3,5) \]
\[ a_{7}[6,10) \]
\[ a_{6}[5,9) \]
“Greedy” Strategies

- Longest first
- Shortest first
- Early start first
- Early finish first
- …
Early Finish Greedy Strategy

1. Sort activities by finish time
2. Schedule the first activity
3. Remove all incompatible activities
4. If there are more activities, repeat 2
Early Finish

- $a_3[0,6)$
- $a_{10}[2,14)$
- $a_{1}[1,4)$
- $a_{9}[8,12)$
- $a_{5}[3,9)$
- $a_{4}[5,7)$
- $a_{8}[8,11)$
- $a_{2}[3,5)$
- $a_{7}[6,10)$
- $a_{11}[12,16)$
- $a_{6}[5,9)$
Early Finish

a3[0,6]

a1[1,4]
a5[3,9]
a4[5,7]
a2[3,5]
a6[5,9]
a10[2,14]
a9[8,12]
a8[8,11]
a7[6,10]
a11[12,16]
Early Finish

- a1[1,4]
- a4[5,7]
- a7[6,10]
- a6[5,9]
- a8[8,11]
- a9[8,12]
- a11[12,16]
Early Finish
Early Finish

a1[1,4]  a9[8,12]

a4[5,7]  a8[8,11]

a11[12,16]
Early Finish

- $a_1[1,4)$
- $a_4[5,7)$
- $a_8[8,11)$
- $a_9[8,12)$
- $a_{11}[12,16)$
Early Finish

- $a_1[1,4]$
- $a_4[5,7]$
- $a_8[8,11]$
- $a_{11}[12,16]$
Early Finish

- a1[1,4]
- a4[5,7]
- a8[8,11]
- a11[12,16]
Optimality of the Greedy Choice

- To prove optimality of the greedy choice, we have to prove the following two properties

1. **Greedy Choice:** The greedy choice is part of the answer

2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem
Greedy Choice

- Let $A \subseteq S$ be an optimal solution. Let $a_j$ be the first element in $A$ and $a_m$ be the first element in $S$.
- We want to prove that $a_m$ is part of an optimal solution.
- If $a_m = a_j$ then we are done.
- Otherwise, we prove that there is another optimal solution $A' = A - \{a_j\} \cup \{a_m\}$.
- Is $A'$ a solution? yes.
- Is $A'$ optimal? yes.
Optimal Substructure

We want to prove that, if \( A \subseteq S \) is an optimal solution to \( S \), then \( A - \{a_m\} \) is an optimal solution to \( S' = S - \{a_j: s_j > f_1\} \).

Proof by contradiction

Assume that \( A - \{a_m\} \) is not optimal.

Then, there is another solution \( B \) to \( S' \) such that \( |B| > |A - \{a_m\}| \geq |A| \).

If this is the case, then the subset \( A' = \{a_m\} \cup B \) is also a solution to \( S \).

\[ |A'| > |B| \Rightarrow |A'| > |A|, \] which means that \( A \) is not optimal which is a contradiction.
Knapsack Problem
0-1 Knapsack Problem

- $4,500
  - ¥15LBs

- $1,500
  - ¥3LBs

- $800
  - ¥2LBs

- $3,000
  - ¥10LBs

- $4,000
  - ¥20LBs

- 45LBs
0-1 Knapsack Problem

- $4,500 for 15LBs
- $1,500 for 3LBs
- $800 for 2LBs
- $3,000 for 10LBs
- $4,000 for 20LBs
- 45LBs

General Mills Cereals:
- Unit price: $0.263 per oz
- Price: $3.95

Other items:
- $800 for an item weighing 2LBs
- $1,500 for an item weighing 3LBs

UCR logo
0-1 Knapsack Problem

- $4,500 15LBs
- $300/LB
- $1,500 3LBs
- $500/LB
- $800 2LBs
- $400/LB
- $3,000 10LBs
- $300/LB
- $4,000 20LBs
- $200/LB

45LBs
Fractional Knapsack Problem

- $4,500
- 15 oz
- $300/oz

- $1,500
- 3 oz
- $500/oz

- $800
- 2 oz
- $400/oz

- $3,000
- 10 oz
- $300/oz

- $4,000
- 20 oz
- $200/oz

45 oz
Problem Formulation

- Given a set $S[1..n]$ of items where each item $i$ has a weight $w_i$ and a value $v_i$, we would like to find the amount $x_i$ of each item that we can take to maximize the total value

$$V = \sum_{i=1}^{n} v_i \left( \frac{x_i}{w_i} \right)$$

- Under the following two conditions
  - $0 \leq x_i \leq w_i$
  - $\sum_{i=1}^{n} x_i \leq W$
Pseudo-code

› Fractional-Knapsack
  › Compute the value-per-weight $y_i = \frac{v_i}{w_i}$ for all items
  › Sort items by $y_i$
  › Set $L = 0$
  › While ($L < W$)
    › Select the item $j$ with the highest $y_j$
    › Set $x_j = \text{Min}(W - L, w_j)$
    › Remove item $j$
    › Set $L = L + x_j$
  › Set all remaining $x_i$’s to 0
Greedy-choice Property

- Given the set $S$ ordered by the value-per-weight ($y$), taking as much as possible $x_j$ from the item $j$ with the highest value-per-weight will lead to an optimal solution.

- Assume we have an optimal solution $X'$ where we take less amount of item $j$, say $x_j' < x_j$.

- We prove that there is another solution $X''$ where we take $x_j$ of item $j$ and get a similar or a higher total value $V$. 

Greedy-choice Property

- Since $x_j` < x_j$, there must be another item $k$ which was taken in an amount that accounts for the difference, i.e., $x_k`$
- We create another solution $X``$ by doing the following changes in $X`$
- Reduce the amount of item $k$ by a value $z$
  \[ x``_k = x_k` - z \]
- Increase the amount of item $j$ by a value $z$
  \[ x``_j = x_j` + z = x_j \] ($x_j$ is the greedy choice)
Greedy Choice Property

\[ \sum x'_{i} = \sum x''_{i} \implies \text{(both are valid solutions)} \]
\[ V' = \sum x'_{i}y_{i}, \quad V'' = \sum x''_{i}y_{i} \]
\[ V'' - V' = (y_{j}x''_{j} + y_{k}x''_{k}) - (y_{j}x'_{j} + y_{k}x'_{k}) \]

(All other items are the same)
\[ V'' - V' = y_{j}(x''_{j} - x'_{j}) - y_{k}(x'_{k} - x''_{k}) \]

But, \( x''_{j} - x'_{j} = x'_{k} - x''_{k} = z \)
\[ V'' - V' = (y_{j} - y_{k}) z \]

Since \( y_{j} \geq y_{k} \) and \( z = x''_{j} - x_{j} > 0 \)
\[ V'' \geq V' \], hence, \( X'' \) is also optimal
Greedy Choice Illustration

Ordered by $y$

$S$

$X'$

$X''$

Input

Optimal answer that does not have the greedy choice

Optimal answer that has the greedy choice

$S$

$w_j$

$X'j$

$x'_k$

$x''_k$

$x''_j = x_j$
Optimal Sub-structure

- Given the problem $S$ with an optimal solution $X$, we want to prove that the solution $X' = X - \{x_j\}$ is optimal to the problem $S'$ after removing the item $j$ and updating the capacity $W' = W - x_j$

- Proof by contradiction
  - Assume that $X'$ is not optimal to $S'$
  - There is another solution $X''$ to $S$ that has a higher total value $V'' > V'$
  - This means we can have a better solution to the problem $S$ which is impossible because $X$ is optimal
Optimal substructure

Input

Ordered by $y$

$S$

Ordered by $w_j$

$X$

Optimal answer for $S$

$S'$

Reduced problem

$X'$ Should be an optimal answer for $S'$
Optimal Substructure

- Input problem $S$ with an optimal answer $X$ of total value $V$
- $X' = X - \{x_j\}$, $V' = V - x_jy_j$
- Assume $X''$ is an optimal solution to $S'$
- $W'' = \sum x''_i \leq (W - x_j)$ and $V'' > V'$
- $X'' + \{x_j\}$ is a valid solution to $S$ because $W'' + x_j \leq W$
- $V'' = V'' + x_jy_j > V' + x_jy_j > V$
- This is a contradiction with the assumption that $X$ is an optimal solution for $S$
Huffman Codes
Encoding

- How data is represented?
- Fixed-size codes, e.g., ASCII
  - A: 1000001
  - B: 1000010
- Variable-size codes, e.g., Morse Codes
  - A: ●▬
  - B: ▬●●●
  - E: ●
  - T: ─
Example: Morse Code

A • •
B • • • •
C • • • •
D • • •
E •
F • • • •
G • • •
H • • • • •
I • • •
J • • • • • •
K • • •
L • • • • •
M • •
N • •
O • • • • • •
P • • • • •
Q • • • •
R • • •
Prefix Codes

- No code is allowed to be a prefix of another code
- To encode, simply concatenate all the codes
- Decoding does not entail any ambiguity
- Example:
  - Message ‘JAVA’
  - a = “0”, j = “11”, v = “10”
  - Encoded message “110100”
  - Decoding “110100”
Trie

- We can use a trie to find prefix codes
- The characters are stored at the external nodes
- A left child (edge) means 0
- A right child (edge) means 1

A = 010
B = 11
C = 00
D = 10
R = 011
Example of Decoding

» encoded text:
  01011011010000101001011011010

» text: ABRACADABRA

» ASCII: 88 bits

» Our encoding: 29 bits

A = 010
B = 11
C = 00
D = 10
R = 011
Another Encoding

- Message: ‘ABRACADABRA’
- Encoded message: ‘001011000100001100101100’
- Length: 24 bits
Optimal Encoding Problem

- Given a set $C$ of $n$ characters, for each character $c \in C$. Let $c.freq$ be the frequency of $c$ in the file. We would like to find a prefix encoding for each $c \in C$ with a length $d_T(c)$ such that we minimize the total cost

$$B = \sum_{c \in C} c.freq \times d_T(c)$$

- Solution: Huffman Codes
Example

“ABRACADABRA”

A,5  B,2  C,1  D,1  R,2
Example

“ABRACADABRA”
Example

“ABRACADABRA”

A,5  B,2

C,1  D,1

4

2

R,2
Example

“ABRACADABRA”
Example

"ABRACADABRA"
Example

"ABRACADABRA"
Example

"ABRACADABRA"

A, 5

A = 0

B, 2

B = 10

C, 1

C = 1100

D, 1

D = 1101

R, 2

R = 111
Encoding

“ABRACADABRA”

0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 1 0

Length = 23
Optimal!

A,5
A=0

B,2
B=10

C,1
C=1100

D,1
D=1101

R,2
R=111
“ABRACADABRA”

A = 0

R = 10

B = 111

C = 1100

D = 1101

Length = 23

Optimal!
Construction of Huffman Tree

- **Huffman(C)**
  - n=|C|
  - Q=C
  - for i = 1 to n-1
    - allocate a new node z
    - z.left = x = Extract-Min(Q)
    - z.right = y = Extract-Min(Q)
    - z.freq = x.freq + y.freq
    - Insert(Q, z)
  - return Extract-Min(Q) // Root of the tree

Note: Can also be done in linear time

\[ T(n) = \Theta(n \log n + |M|) \]
Optimality of Huffman Codes

- **Greedy-choice**
  - The greedy choice yields an optimal solution.

- **Optimal sub-structure**
  - The optimal solution for the bigger problem contains the optimal solution of the sub-problem.

- Detailed proof in the textbook
Greedy Choice

- Greedy choice: Choose the characters \( x \) and \( y \) with the highest frequencies and merge them under one internal node.
- We need to prove that the optimal tree has \( x \) and \( y \) as siblings under one common node.
- Assume there is an optimal tree \( T \) where \( x \) and \( y \) are not siblings.
- In the same tree, the two siblings at the deepest level are other characters \( a \) and \( b \).
- Assume \( x.freq \leq y.freq \) and \( a.freq \leq b.freq \).
Greedy Choice