Greedy Algorithms
Chapter 16
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

2
4
3
5
6

10
Example: Bin Packing
Example: Bin Packing

5

6

3

4

2
Example: Bin Packing
Example: Bin Packing

5

5

6

1

3

4

2
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 $S = \{a_1, a_2, ..., a_n\}$ 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
An Optimal (Best) Solution

- a3[0,6]
- a1[1,4]
- a10[2,14]
- a5[3,9]
- a4[5,7]
- a2[3,5]
- a7[6,10]
- a6[5,9]
- a9[8,12]
- a8[8,11]
- a11[12,16]
Another Optimal 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)
“Greedy” Strategies

- Longest first
- Shortest first
- Early start first
- Early finish first
- ...

[Image 616x446 to 720x540]
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

- 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]
Early Finish

\[ a_3(0,6) \]

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

\[ a_1(1,4) \]

\[ a_9(8,12) \]

\[ a_5(3,9) \]

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

\[ a_4(5,7) \]

\[ a_8(8,11) \]

\[ a_2(3,5) \]

\[ a_7(6,10) \]
Early Finish

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

- \(a_1[1,4]\)
- \(a_4[5,7]\)
- \(a_6[5,9]\)
- \(a_7[6,10]\)
- \(a_8[8,11]\)
- \(a_9[8,12]\)
- \(a_{11}[12,16]\)
Early Finish

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

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

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

a11[12,16]
Early Finish

\[ a_1(1,4) \]

\[ a_4(5,7) \quad a_8(8,11) \]

\[ a_{11}(12,16) \]
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 $A^\prime = A - \{a_j\} \cup \{a_m\}$
- Is $A^\prime$ a solution? yes
- Is $A^\prime$ 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 $S' = \{a_i: a_i \in S \land s_i > f_m\}$

Proof by contradiction

Assume that $A - \{a_m\}$ is not optimal

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

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
  - 15LBs

- $1,500
  - 3LBs

- $800
  - 2LBs

- $3,000
  - 10LBs

- $4,000
  - 20LBs

- 45LBs

- 15LBs

- 3LBs

- 2LBs

- 10LBs

- 20LBs

- 45LBs
0-1 Knapsack Problem

Total value = $9,800
There isn’t a known solution to the 0-1 Knapsack problem using a greedy algorithm.
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 constraints

- $0 \leq x_i \leq w_i$
- $\sum_{i=1}^{n} x_i \leq W$
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 $x_j = Min(W - L, w_j)$
    ▶ 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 there is 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 $z$
    $$x''_j = x'_j + z = x_j \ (x_j \text{ is the greedy choice})$$
Greedy Choice Property

\[ \sum x'_i = \sum x''_i \Rightarrow \text{(both are valid solutions)} \]

\[ V' = \sum x'_i y_i, \ V'' = \sum x''_i y_i \]

\[ V'' - V' = (y_j x''_j + y_k x''_k) - (y_j x'_j + y_k x'_k) \]

(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 \( z = x''_j - x_j > 0 \)

\[ V'' \geq V', \text{ hence, } X'' \text{ is also optimal} \]
Greedy Choice Illustration

Ordered by $y$

$S$

$j$

$w_j$

$X'$

$j$

$x'_j$

$x'_k$

$X''$

$j$

$x''_j = x_j$

$x''_k$

Optimal answer that does not have the greedy choice

Optimal answer that has the greedy choice

Input
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 $w_j$

Optimal answer for $S$

$X'$

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 -----
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’
  - \(a = \text{"0"}, \ j = \text{"11"}, \ v = \text{"10"}\)
  - 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 $c.f\text{req}$ 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

\[
B = \sum_{c \in C} c.f\text{req} \times d_T(c)
\]

- 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

R,2

4

2
Example

“ABRACADABRA”
Example

"ABRACADABRA"
Example

"ABRACADABRA"
Example

```
A,5
A=0

B,2
B=10

C,1
C=1100

D,1
D=1101

R,2
R=111

“ABRACADABRA”
```
Encoding

“ABRACADABRA”

0 10 111 0 1100 0 1101 0 10 111 0

Length = 23

Optimal!

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

A=0
B=10
C=1100
D=1101
R=111
Encoding

“ABRACADABRA”

0 111 10 0 1100 0 1101 0 111 10 0

Length = 23

Optimal!
Construction of Huffman Tree

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

   \[ T(n) = \Theta(n \log n + |M|) \]

- Note: Can also be done in linear time
- \( \Theta(n \log n) \)
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

$T$

$T'$

$T''$