

# Greedy Algorithms

## Definitions

### Fractional Knapsack

- Some selection criteria
- Highest  $v/w$
- 0-1 Knapsack

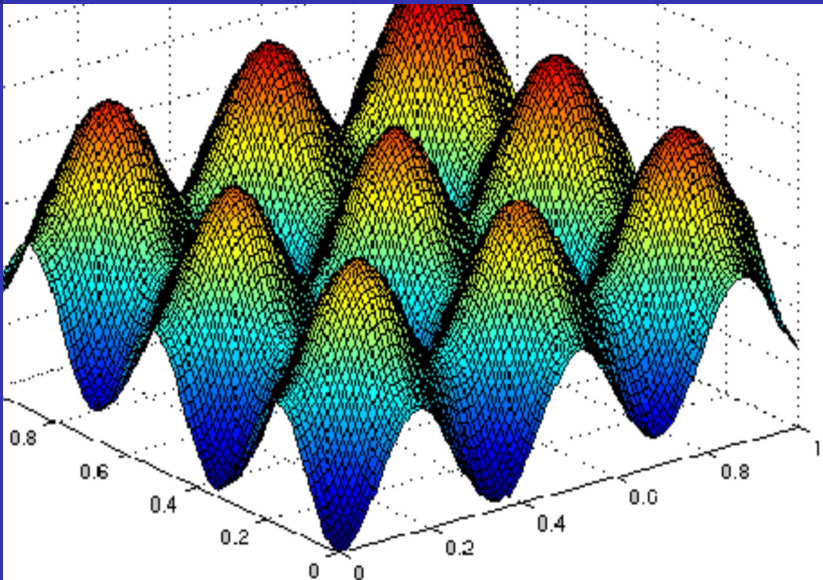
### Scheduling

- Interval scheduling
- Weighted activity selection
- Minimizing lateness

### Optimal prefix codes

- data compression
- prefix codes
- Huffman code

### Approximation algorithms



# Greedy Algorithms

- Greedy algorithms are mainly designed to solve **combinatorial optimization problems**:

Given an input, we want to compute an optimal solution according to some **objective function**.

- The solutions are formed by a sequence of elements.
- For example: Given a graph  $G = (V, E)$  and two vertices  $u, v \in V$ , we want to find a path from  $u$  to  $v$  having the minimum number of edges.

The solution is a sequence of vertices or edges.

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# Greedy Algorithms

A greedy algorithm obtains an optimal solution to a combinatorial optimization problem by making a sequence of choices (without backtracking).

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

- Greedy algorithms make locally optimal **myopic choices** to construct incrementally a global solution.
- In some cases this will lead to a globally optimal solution.
- Often **easy** greedy algorithms are used to obtain **quickly** solutions to optimization problems, even though they do not always yield optimal solutions.
- For many problems the greedy technique yields good heuristics, or even good approximation algorithms.

# Greedy Algorithms

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

- At each step we choose the best (myopic) choice at the **moment** for the corresponding component of the solution, and then solve the subproblem that arise by taking this decision.
- The choice may depend on previous choices, but not on future choices.
- At each choice, the algorithm reduces the problem into a smaller one, and obtains one component of the solution.
- A greedy algorithm never backtracks.

# Greedy Algorithms

For the greedy strategy to work correctly, it is necessary that the problem under consideration has two characteristics:

- **Greedy choice property:** We can arrive to the global optimum by selecting a local optimums.
- **Optimal substructure:** After making some local decision, it must be the case that there is an optimal solution to the problem that contains the partial solution constructed so far.

In many cases, the local criteria for selecting a part of the solution allow us to define a global order that directs the greedy algorithm.

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# The FRACTIONAL KNAPSACK problem

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

## Optimal prefix codes

data compression  
prefix codes  
Huffman code

## Approximation algorithms

**FRACTIONAL KNAPSACK:** Given as input a set of  $n$  items, where item  $i$  has weight  $w_i$  and value  $v_i$ , together with a maximum total weight  $W$  permissible. We want to select a set of items or fractions of item, to maximize the profit, within allowed weight  $W$ .

Observe that from each item we can select any arbitrary fraction of its weight.

**Example.**  $n = 5$  and  $W = 100$

Item	1	2	3	4	5
$w$	10	20	30	40	50
$v$	20	30	66	40	60



# FRACTIONAL KNAPSACK: GREEDY SCHEMA

**GreedyFKnapsack** ( $n, v, w, W$ )

$O = \{1, \dots, n\}; S = \emptyset; Val = 0; i = 0;$

**while**  $W > 0$  **do**

Let  $i \in O$  be the item with **property P**

**if**  $w[i] \leq W$  **then**

$S = S \cup \{(i, 1)\}; W = W - w[i]; Val = Val + v[i];$

**else**

$S = S \cup \{(i, W/w[i])\}; W = 0;$

$Val = Val + v[i] * W/w[i];$

**end if**

Remove  $i$  from  $O$ .

**end while**

**return**  $S$

Definitions

**Fractional Knapsack**

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

**Optimal prefix codes**

data compression

prefix codes

Huffman code

**Approximation algorithms**

# GreedyFKnapsack: most valuable object

*Example.*  $n = 5$  and  $W = 100$

Item	1	2	3	4	5
$w$	10	20	30	40	50
$v$	20	30	66	40	60

Item	1	2	3	4	5
Selected	0	0	1	0.5	1

Total selected weight **100** and total value **146**

Selecting the most valuable object is a correct greedy rule?

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms



# GreedyFKnapsack: the lighter object

*Example.*  $n = 5$  and  $W = 100$

Item	1	2	3	4	5
$w$	10	20	30	40	50
$v$	20	30	66	40	60

Item	1	2	3	4	5
Selected	1	1	1	1	0

Total selected weight **100** and total value **156**

Selecting the **most valuable object** does not provide a correct solution.

Selecting the **lighter object** is a correct greedy rule?

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# GreedyFKnapsack: the highest ratio value/weight

*Example.*  $n = 5$  and  $W = 100$

Item	1	2	3	4	5
$w$	10	20	30	40	50
$v$	20	30	66	40	60

Item	1	2	3	4	5
ratio	2.0	1.5	2.2	1.0	1.2
Selected	1	1	1	0	0.8

Total selected weight **100** and total value **164**

Selecting the **lighter object** does not provide a correct solution.

**Highest ratio value/weight is a correct greedy rule?**

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# GreedyFKnapsack: highest ratio value/weight

## Theorem

*The GreedyFKnapsack selecting the item with the best ratio value/weight always finds an optimal solution to the FRACTIONAL KNAPSACK problem*

## Proof.

Assume that the  $n$  items are sorted so that

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \dots \geq \frac{v_n}{w_n}$$

### Definitions

#### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

prefix codes

Huffman code

### Approximation algorithms

# GreedyFKnapsack: highest ratio value/weight

Let  $X = (x_1, \dots, x_n)$ ,  $x_i \in [0, 1]$ , be the portions of items selected by the algorithm.

- If  $x_i = 1$ , for all  $i$ , the computed solution is optimal.  
We take all!
- Otherwise, let  $j$  be the smallest value for which  $x_j < 1$ .
- According with the algorithm,  
 $x_i = 1$ , for  $i < j$ , and  
 $x_i = 0$ , for  $i > j$ .
- Furthermore,  $\sum_{i=1}^n x_i w_i = W$

## Definitions

### Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

prefix codes

Huffman code

### Approximation algorithms

# GreedyFKnapsack: highest ratio value/weight

Let  $Y = (y_1, \dots, y_n)$ ,  $y_i \in [0, 1]$ , be the portions of items selected in a **feasible** solution, i.e.,

$$\sum_{i=1}^n y_i w_i \leq W$$

- We have,  $\sum_{i=1}^n y_i w_i \leq W = \sum_{i=1}^n x_i w_i$
- So,  $0 \leq \sum_{i=1}^n x_i w_i - \sum_{i=1}^n y_i w_i = \sum_{i=1}^n (x_i - y_i) w_i$
- Then, the value difference can be expressed as

$$\begin{aligned} v(X) - v(Y) &= \sum_{i=1}^n x_i v_i - \sum_{i=1}^n y_i v_i = \sum_{i=1}^n (x_i - y_i) v_i \\ &= \sum_{i=1}^n (x_i - y_i) w_i \frac{v_i}{w_i} \end{aligned}$$

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# GreedyFKnapsack: highest ratio value/weight

We want to bound  $v(x) - v(y) = \sum_{i=1}^n (x_i - y_i) w_i \frac{v_i}{w_i}$ .

- If  $i < j$ ,  $x_i = 1$ , so  $x_i - y_i \geq 0$  but, as  $\frac{v_i}{w_i} \geq \frac{v_j}{w_j}$ ,

$$(x_i - y_i) \frac{v_i}{w_i} \geq (x_i - y_i) \frac{v_j}{w_j}$$

- If  $i > j$ ,  $x_i = 0$ , so  $x_i - y_i \leq 0$  but, as  $\frac{v_i}{w_i} \leq \frac{v_j}{w_j}$ ,

$$(x_i - y_i) \frac{v_i}{w_i} \geq (x_i - y_i) \frac{v_j}{w_j}$$

- The same inequality in both cases.

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# GreedyFKnapsack: highest ratio value/weight

- Using the derived inequalities, we have

$$\begin{aligned} v(x) - v(y) &= \sum_{i=1}^n (x_i - y_i) w_i \frac{v_i}{w_i} \\ &\geq \sum_{i=1}^n (x_i - y_i) w_i \frac{v_j}{w_j} \geq \frac{v_j}{w_j} \sum_{i=1}^n (x_i - y_i) w_i \geq 0 \end{aligned}$$

- So,  $v(X) - v(Y) \geq 0$ , and **x is an optimal solution.**

End Proof

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# GreedyFKnapsack: highest ratio value/weight

**GreedyFKnapsack** ( $n, v, w, W$ )

$O = \{1, \dots, n\}; S = \emptyset; Val = 0; i = 0;$

**while**  $W > 0$  **do**

Let  $i \in O$  be an item with highest value/weight

**if**  $w[i] < W$  **then**

$S = S \cup \{(i, 1)\}; W = W - w[i]; Val = Val + v[i];$

**else**

$S = S \cup \{(i, W/w[i])\}; W = 0;$

$Val = Val + v[i] * W/w[i];$

**end if**

Remove  $i$  from  $O$

**end while**

**return**  $S$

**Cost?**  $O(n^2)$  a better implementation?

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms



# FRACTIONAL KNAPSACK

**GreedyFKnapsack** ( $n, v, w, W$ )

Sort the items in decreasing value of  $v_i/w_i$

$S = \emptyset$ ;  $Val = 0$ ;  $i = 0$ ;

**while**  $W > 0$  and  $i < n$  **do**

**if**  $w[i] < W$  **then**

$S = S \cup \{(i, 1)\}$ ;  $W = W - w[i]$ ;  $Val = Val + v[i]$ ;

**else**

$S = S \cup \{(i, W/w[i])\}$ ;  $W = 0$ ;

$Val = Val + v[i] * W/w[i]$ ;

**end if**

$++i$ ;

**end while**

**return**  $S$

This algorithm has cost of  $T(n) = O(n \log n)$ .

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# FRACTIONAL KNAPSACK

## Theorem

*The FRACTIONAL KNAPSACK problem can be solved in time  $O(n \log n)$ .*

### Definitions

#### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

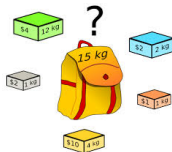
prefix codes

Huffman code

### Approximation algorithms

# 0-1 KNAPSACK

0-1 KNAPSACK Given as input a set of  $n$  items, where item  $i$  has weight  $w_i$  and value  $v_i$ , together with a maximum total weight  $W$  permissible. We want to select a set of items to maximize the profit, within allowed weight  $W$ .



Items cannot be fractioned, you have to take all or nothing.

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

The greedy algorithm for the fractional version does not work for 0-1 KNAPSACK

#### Definitions

#### Fractional Knapsack

Some selection criteria

Highest  $v/w$

#### 0-1 Knapsack

#### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

#### Optimal prefix codes

data compression

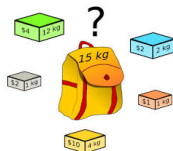
prefix codes

Huffman code

#### Approximation algorithms

Example:  $n = 3$  and  $W = 50$

Item	1	2	3
$w$	10	20	30
$v$	60	100	120
$v/w$	6	5	4



The algorithm will select item 1, with value 60. This is not an optimal solution, as 2 and 3 form a better solution, with value 220.

But, 0-1 KNAPSACK is known to be NP-hard.

# Tasks or Activities Scheduling problems

## General Setting:

- Given: A set of  $n$  tasks (with different characteristics) to be processed by a single/multiple processor system (according to different constraints).
- Provide a schedule, (when and where a (each) task must be executed), so as to optimize some objective criteria.

### Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# Some mono processor scheduling problems

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

- 1 INTERVAL SCHEDULING problem:** Tasks have **start** and **finish** times. The objective is to make an executable selection with **maximum** size.
- 2 WEIGHTED INTERVAL SCHEDULING problem:** Tasks have **start** and **finish** times and its execution produce **profits**. The objective is to make an executable selection giving **maximum** profit.
- 3 JOB SCHEDULING problem (Lateness minimization):** Tasks have **processing time** (could start at any time) and a **deadline**, define the lateness of a task as the time from its deadline to its starting time. **Find an executable schedule, including all the tasks, that minimizes the total lateness.**

# The INTERVAL SCHEDULING problem

The INTERVAL SCHEDULING (aka ACTIVITY SELECTION problem)

- Given a set of  $n$  tasks where, for  $i \in [n]$ , task  $i$  has a **start time  $s_i$**  and a **finish time  $f_i$** , with  $s_i < f_i$ .
- The processor is a single machine, that can process only one task at a time.
- A task must be processed completely from its starting time to its finish time.
- We want to find a set of **mutually compatible tasks**, where activities  $i$  and  $j$  are compatible if  $[s_i, f_i) \cap (s_j, f_j] = \emptyset$ , with **maximum size**.

A solution is a set of mutually compatible activities, and the objective function to maximize is the cardinality of the solution set.

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

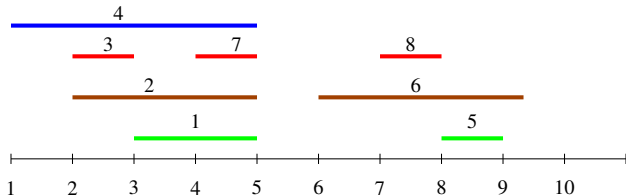
### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# Example: one input

Task :	1	2	3	4	5	6	7	8
Start (s):	3	2	2	1	8	6	4	7
Finish (f):	5	5	3	5	9	9	5	8



Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms



# Designing a greedy algorithm

To apply the greedy technique to a problem, we must take into consideration the following,

- A **local criteria** to allow the selection,
- having in mind a **property** ensuring that a partial solution can be completed to an optimal solution.

As for the FRACTIONALKNAPSACK problem, the selection criteria might lead to a sorting criteria. In such a case, greedy processes the input in this particular order.

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# The Interval Scheduling problem: Earlier finish time

**IntervalScheduling( $A$ )**

$S = \emptyset$ ;  $T = \{1, \dots, n\}$ ;

**while**  $T \neq \emptyset$  **do**

    Let  $i$  be the task that finishes earlier among those in  $T$

$S = S \cup \{i\}$ ;

    Remove from  $T$ ,  $i$  and all tasks  $j \in T$  with  $s_j \leq t_i$

**end while**

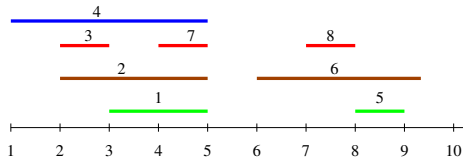
**return**  $S$ .

$task :$     3 4 2 7 8 5 6

$s :$         3 1 2 4 8 5 6

$f :$         3 5 5 5 8 9 9

**SOL: 3 1 8 5**



Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# IntervalScheduling: correctness

## Theorem

*The **IntervalScheduling** algorithm produces an optimal solution to the INTERVAL SCHEDULING problem.*

## Proof.

We want to prove that:

There is an optimal solution that includes the task with the earlier finishing time.

We will assume that this is not the case and reach contradiction.

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# IntervalScheduling: correctness

- Let  $i$  be a task that finishes at the earliest finish time.
- Let  $S$  be an optimal solution with  $i \notin S$ .  
Let  $k \in S$  be the task with the earlier finish time among those in  $S$ .
- Any task in  $S$  finishes after time  $A[k].f$ , so they start also after  $A[k].f$ . As  $A[i].f \leq A[k].f$ ,  $S' = (S - \{k\}) \cup \{i\}$  is a set of mutually compatible tasks.
- As  $|S'| = |S|$ ,  $S'$  is an optimal solution that includes  $i$ .

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# IntervalScheduling: correctness

## Optimal substructure

After each greedy choice, we are left with an optimization subproblem, of the same form as the original. In the subproblem we removed the selected task and all tasks that overlap with the selected one.

An optimal solution to the original problem is formed by the selected task (one that finishes earliest possible) and an optimal solution to the corresponding subproblem.

End Proof

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Interval Scheduling: cost

**IntervalScheduling( $A$ )**

$S = \emptyset$ ;  $T = [n]$ ;  $O(n)$

**while**  $T \neq \emptyset$  **do**

Let  $i$  be the task that finishes earlier among those in  $T$

$O(n)$

$S = S \cup \{i\}$ ;

Remove  $i$  and all tasks overlapping  $i$  from  $T$   $O(n)$

**end while**

**return**  $S$ .

It takes  $O(n^2)$  Too slow, a better implementation?

We have to find a fastest way to select  $i$  and discard  $i$  and the overlapping tasks.

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# The Interval Scheduling problem: algorithm 2

## IntervalScheduling2( $A$ )

Sort  $A$  in increasing order of  $A.f$

$S = \{0\}$

$j = 0$  {pointer to last task in solution}

**for**  $i = 1$  **to**  $n - 1$  **do**

**if**  $A[i].s \geq A[j].f$  **then**

$S = S \cup \{i\}; j = i;$

**end if**

**end for**

**return**  $S$ .

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# IntervalScheduling2: correctness

## Theorem

*The **IntervalScheduling2** algorithm produces an optimal solution to the INTERVAL SCHEDULING problem in time  $O(n \log n)$*

## Proof.

- A task that does not verify  $A[i].s \geq A[j].f$  overlaps with task  $j \in S$ . It starts before  $j$  and finishes after  $j$  finishes. Therefore, it cannot be part of a solution together with  $j$ .
- As the tasks are sorted by finish time at each step, we select, among those tasks that start later than  $j$ , the one that finishes earlier.

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms



# IntervalScheduling2: correctness

- **IntervalScheduling2** makes the same greedy choice as **IntervalScheduling**, therefore it computes an optimal solution.
- The most costly step in **IntervalScheduling2** is the sorting, which can be done in  $O(n \log n)$  time using Merge sort.

End Proof

Definitions

Fractional  
Knapsack

Some selection  
criteria  
Highest v/w  
0-1 Knapsack

Scheduling

Interval scheduling  
Weighted activity  
selection  
Minimizing lateness

Optimal prefix  
codes

data compression  
prefix codes  
Huffman code

Approximation  
algorithms

# IntervalScheduling2: particular case

If we know that the tasks start and finish time are given in seconds within a day (24 hours),

**IntervalScheduling2** can be implemented with cost  $O(n)$

## Definitions

### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

## Scheduling

### Interval scheduling

Weighted activity selection

Minimizing lateness

## Optimal prefix codes

data compression

prefix codes

Huffman code

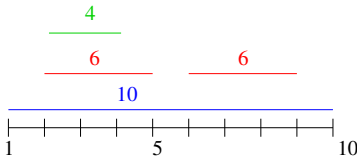
## Approximation algorithms

# Adding weights: greedy choice does not always work.

## WEIGHTED ACTIVITY SELECTION problem:

Given a set of  $n$  activities to be processed by a single machine, where each activity  $i$  has a **start time**  $s_i$  and a **finish time**  $f_i$ , with  $s_i < f_i$ , and a **weight**  $w_i$ .

We want to find a set  $S$  of mutually compatible activities so that  $\sum_{i \in S} w_i$  is **maximum** among all such sets.



**IntervalScheduling2** selects the green and the second red activity with weight 10 which is not an optimal solution.

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# What about maximizing locally the selected weight?

## WeightedAS-max-weight ( $A$ )

$S = \emptyset; T = [n];$

**while**  $T \neq \emptyset$  **do**

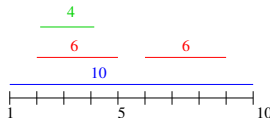
Let  $i$  be the task with highest weight among those in  $T$ .

$S = S \cup \{i\}$

Remove  $i$  and all tasks overlapping  $i$  from  $T$

**end while**

**return**  $S$



The algorithm chooses the blue task with weight 10, and the optimal solution is formed by the two red intervals with total weight of 12.

Definitions

Fractional  
Knapsack

Some selection  
criteria  
Highest v/w  
0-1 Knapsack

Scheduling

Interval scheduling  
Weighted activity  
selection  
Minimizing lateness

Optimal prefix  
codes

data compression  
prefix codes  
Huffman code

Approximation  
algorithms

# Greedy approach

- Easy to come up with one or more greedy algorithms
- Easy to analyze the running time.
- Hard to establish correctness.
- Most greedy algorithms we came up are not correct on all inputs.

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

## Scheduling

Interval scheduling  
**Weighted activity selection**  
Minimizing lateness

## Optimal prefix codes

data compression  
prefix codes  
Huffman code

## Approximation algorithms

# A Job Scheduling problem

## LATENESS MINIMIZATION problem.

- We have a **single processor** and  $n$  tasks (or jobs) to be processed.
- Once a task starts to be processed it continues using the processor until its completion.
- Processing task  $i$  takes **time  $t_i$** . Furthermore, task  $i$  has a **deadline  $d_i$** .
- The goal is to schedule **all the tasks**, i.e., determine the time at which to start processing each tasks.
- We want to minimize, over all the tasks, the maximum amount of time that the finish time of a tasks exceeds its deadline.

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Minimize Lateness: a more formal formulation

## Definitions

## Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

## Optimal prefix codes

data compression  
prefix codes  
Huffman code

## Approximation algorithms

- We have a single processor
- We have  $n$  jobs such that job  $i$ :
  - requires  $t_i > 0$  units of processing time,
  - it has to be finished by time  $d_i$ ,
  - A schedule will determine a finish time  $f_i$
- Under this schedule lateness of  $i$  is:

$$L_i = \begin{cases} 0 & \text{if } f_i \leq d_i, \\ f_i - d_i & \text{otherwise.} \end{cases}$$

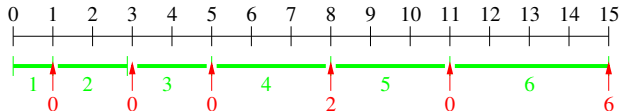
- The **lateness** of a valid schedule is  $\max_i L_i$ .

**Goal: find a schedule with minimum lateness**

# Minimize Lateness: an example

We must assign starting time  $s_i$  to each  $i$ , making sure that the processor only processes a job at a time, in such a way that  $\max_i L_i$  is minimum.

6 tasks:  $t$ : 1 2 2 3 3 4     $d$ : 9 8 15 6 14 9





# Minimize Lateness

We can try different **task selection criteria** to schedule the jobs following a generic greedy algorithm.

## Definitions

### Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

## Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

## Optimal prefix codes

data compression

prefix codes

Huffman code

## Approximation algorithms

### **LatenessXX** ( $A$ )

Sort  $A$  according to  $XX$

$S[0] = 0$ ;  $t = A[0].t$ ;  $L = \max(0, t - A[0].d)$ ;

**for**  $i = 1$  **to**  $n - 1$  **do**

$S[i] = t$

$t = t + A[i].t$

$L = \max(L, \max(0, t - A[i].d))$

**end for**

**return**  $(S, L)$

# Minimize Lateness: selection criteria

Process jobs with short time first

$i$	$t_i$	$d_i$
1	1	6
2	5	5

1 at time 0 and 2 at time 1 lateness 1, but  
2 at time 0 and 1 at time 5 has lateness 0.  
It does not work.

Process first jobs with smaller  $d_i - t_i$  time

$i$	$t_i$	$d_i$	$d_i - t_i$
1	1	2	1
2	10	10	0

2 should start at time 0, that  
does not minimize lateness.

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# Process urgent jobs first

Sort in increasing order of  $d_i$ .

**LatenessUrgent** ( $A$ )

Sort  $A$  by increasing order of  $A.d$

$S[0] = 0$ ;  $t = A[0].t$ ;

$L = \max(0, t - A[0].d)$ ;

**for**  $i = 1$  **to**  $n - 1$  **do**

$S[i] = t$

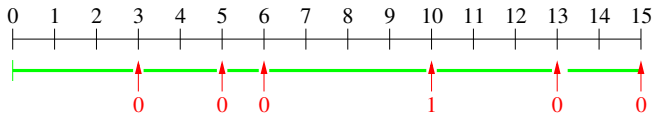
$t = t + A[i].t$

$L = \max(L, \max(0, t - A[i].d))$

**end for**

**return** ( $S, L$ )

$i$	$t$	$d$	pos. sorted by $d$
1	1	9	3
2	2	8	2
3	2	15	6
4	3	6	1
5	3	14	5
6	4	9	4



Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# Process urgent jobs first: Complexity

## LatenessUrgent ( $A$ )

Sort  $A$  by increasing order of  $A.d$

$S[0] = 0; t = A[0].t; L = \max(0, t - A[0].d);$

**for**  $i = 1$  **to**  $n - 1$  **do**

$S[i] = t$

$t = t + A[i].t$

$L = \max(L, \max(0, t - A[i].d))$

**end for**

**return** ( $S, L$ )

Time complexity

Running-time of the algorithm without sorting  $O(n)$

Total running-time:  $O(n \lg n)$

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

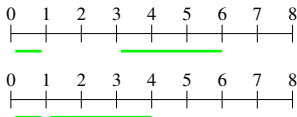
Approximation  
algorithms

# Process urgent jobs first: Correctness

## Lemma

*There is an optimal schedule minimizing lateness that does not have idle steps.*

From a schedule with idle steps, we always can eliminate gaps to obtain another schedule with the same or better lateness:



**LatenessUrgent** has no idle steps.

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# Inversions and exchange argument

A schedule  $S$  has **an inversion** if  $S(i) < S(j)$  and  $d_j < d_i$ .

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

## Lemma

*Exchanging two adjacent inverted jobs reduces the number of inversions by 1 and does not increase the max lateness.*

## Proof.

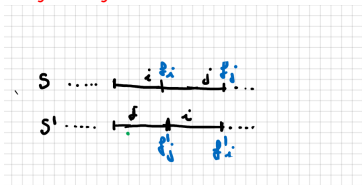
Assume that in schedule  $S$ ,  $i$  is scheduled just before  $j$  and that they form an inversion.

Let  $S'$  be the schedule obtained from  $S$  interchanging  $i$  with  $j$ .

- $S[k] = S'[k]$  for  $k \neq i$  and  $k \neq j$ .
- Thus, only  $i$  and  $j$  can change lateness.

# Inversions and exchange argument: Cont.

- Let  $L_i, L_j$  and  $L'_i, L'_j$  be the lateness of jobs  $i$  and  $j$  in  $S$  and  $S'$ , respectively. Recall  $d_j < d_i$ .
- Let  $f_i, f_j$  and  $f'_i, f'_j$  be the finish times of jobs  $i$  and  $j$  in  $S$  and  $S'$ , respectively.
- We have  $f_i < f_j$ ,  $f'_i < f'_j$ ,  $f'_i = f_j$ , and  $f'_j < f_j$ .
- If  $f_j < d_j$ ,



$$L'_i = L'_j = 0 = L_i = L_j$$

Both schedules have the same latency.

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

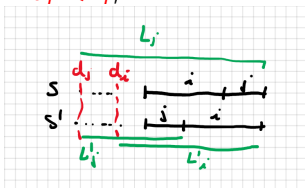
prefix codes

Huffman code

Approximation algorithms

# Inversions and exchange argument: Cont.

- Let  $L_i, L_j$  and  $L'_i, L'_j$  be the lateness of jobs  $i$  and  $j$  in  $S$  and  $S'$ , respectively. Recall  $d_j < d_i$ .
- Let  $f_i, f_j$  and  $f'_i, f'_j$  be the finish times of jobs  $i$  and  $j$  in  $S$  and  $S'$ , respectively.
- We have  $f_i < f_j, f'_i < f'_j, f'_i = f_j$ , and  $f'_j < f_j$ .
- If  $d_i < f_i$ ,



$$L'_i, L'_j \leq L_i$$

$S'$  has the same or better latency than  $S$ .

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

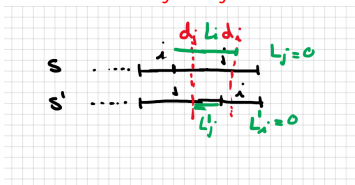
Huffman code

Approximation algorithms



# Inversions and exchange argument: Cont.

- Let  $L_i, L_j$  and  $L'_i, L'_j$  be the lateness of jobs  $i$  and  $j$  in  $S$  and  $S'$ , respectively. Recall  $d_j < d_i$ .
- Let  $f_i, f_j$  and  $f'_i, f'_j$  be the finish times of jobs  $i$  and  $j$  in  $S$  and  $S'$ , respectively.
- We have  $f_i < f_j, f'_i < f'_j, f'_i = f_j$ , and  $f'_j < f_j$ .
- if  $f_i \leq d_i < d_j \leq f_j$ ,



$$L'_i = 0 \text{ and } L'_j \leq L_i$$

$S'$  has the same or better latency than  $S$ .

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Inversions and exchange argument: Cont.

Therefore, in all the three cases, the swapping does not increase the maximum lateness of the schedule.

End Proof

## Definitions

### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

prefix codes

Huffman code

### Approximation algorithms

# Correctness of **LatenessUrgent**

## Theorem

*Algorithm **LatenessUrgent** solves correctly the LATENESS MINIMIZATION problem. in  $O(n \log n)$  time*

## Proof.

According to the design, the schedule  $S$  produced by **LatenessUrgent** has no inversions and no idle steps.

Assume  $\hat{S}$  is an optimal schedule. We can assume that it has no idle steps.

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Correctness of LatenessUrgent

- If  $\hat{S}$  has 0 inversions,  $S$  sorts jobs by deadlines and  $\hat{S} = S$ .
- Otherwise,  $\hat{S}$  has an inversion on two adjacent jobs.  
Let  $i, j$  be an adjacent inversion.  
As we have seen, exchanging  $i$  and  $j$  does not increase lateness but it decreases the number of inversions.  
As  $\hat{S}$  is optimal, the new schedule is also optimal but has one inversion less.
- Repeating, if needed the interchange of adjacent inversions, we will reach an optimal schedule with no inversions. Therefore,  $S$  is optimal.

End Proof

## Definitions

## Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

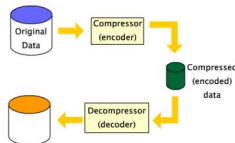
## Optimal prefix codes

data compression  
prefix codes  
Huffman code

## Approximation algorithms

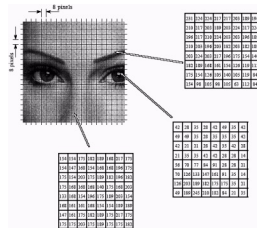
# DATA COMPRESSION

Given as input a text  $\mathcal{T}$  over a finite alphabet  $\Sigma$ . We want to represent  $\mathcal{T}$  with as few bits as possible.



The goal of data compression is to reduce the time to transmit large files, and to reduce the space to store them.

If we are using variable-length encoding we need a system easy to encode and decode.



Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Example.

## Definitions

### Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

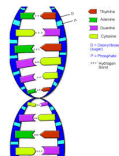
data compression

prefix codes

Huffman code

### Approximation algorithms

AAACAGTTGCAT ... GGTCCCTAGG  
130.000.000



- *Fixed-length encoding*:  $A = 00$ ,  $C = 01$ ,  $G = 10$  and  $T = 11$ . Needs 260Mbites to store.
- *Variable-length encoding*: If  $A$  appears  $7 \times 10^8$  times,  $C$  appears  $3 \times 10^6$  times,  $G$   $2 \times 10^8$  and  $T$   $37 \times 10^7$ , better to assign a shorter string to  $A$  and longer to  $C$

# Prefix codes

Given a set of symbols  $\Sigma$ , a **prefix code**, is  $\phi : \Sigma \rightarrow \{0, 1\}^+$  (symbols to chain of bits) where for distinct  $x, y \in \Sigma$ ,  $\phi(x)$  is not a prefix of  $\phi(y)$ .

- $\phi(A) = 1$  and  $\phi(C) = 101$  then  $\phi$  is **not** a prefix code.
- $\phi(A) = 1, \phi(T) = 01, \phi(G) = 000, \phi(C) = 001$  is a prefix code.
- Prefix codes easy to decode (left-to-right):

$\underbrace{000}_G \underbrace{1}_A \underbrace{01}_T \underbrace{1}_A \underbrace{001}_C \underbrace{1}_A \underbrace{01}_T \underbrace{000}_G \underbrace{001}_C \underbrace{01}_T$

## Definitions

## Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

## Optimal prefix codes

data compression  
**prefix codes**  
Huffman code

## Approximation algorithms

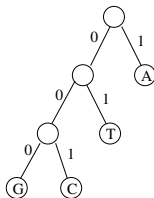
# Prefix tree

We can identify an encoding with prefix property with a labeled binary tree.

A **prefix tree**  $T$  is a binary tree with the following properties:

- One leaf for symbol,
- Left edge labeled 0 and right edge labeled 1,
- Labels on the path from the root to a leaf specify the code for the symbol in that leaf.

$\Sigma$	code
A	1
T	01
G	000
C	001



## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms



# The encoding length

- Given a text  $S$  on  $\Sigma$ , with  $|S| = n$ , and a prefix code  $\phi$ ,  $B(S)$  is the length of the encoded text.
- For  $x \in \Sigma$ , define the **frequency** of  $x$  as

$$f(x) = \frac{\text{number occurrences of } x \in S}{n}$$

*Note:*  $\sum_{x \in \Sigma} f(x) = 1$ .

- We get the formula,

$$B(S) = \sum_{x \in \Sigma} n f(x) |\phi(x)| = n \sum_{x \in \Sigma} f(x) |\phi(x)|.$$

- $\alpha(S) = \sum_{x \in \Sigma} f(x) |\phi(x)|$  is the **average number of bits** per symbol or **compression factor**.

Definitions

Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# The encoding length

- In terms of the prefix tree of  $\phi$ , the length of a codeword  $|\phi(x)|$  is the depth of the leaf labeled  $x$  in  $T$  ( $d_T(x)$ ).
- Thus,  $\alpha(T) = \sum_{x \in \Sigma} f(x) d_T(x)$ .

## Definitions

### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

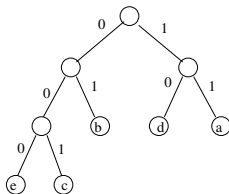
**prefix codes**

Huffman code

### Approximation algorithms

# Fixed versus variable length codes: Example.

- Let  $\Sigma = \{a, b, c, d, e\}$  and let  $S$  be a text over  $\Sigma$  with frequencies:  
 $f(a) = .32, f(b) = .25, f(c) = .20, f(d) = .18, f(e) = .05$
- If we use a fixed length  $\phi$  code, we need  $\lceil \lg 5 \rceil = 3$  bits, we get compression 3.
- Consider the prefix-code  $\phi_1$ :



$$\alpha = .32 \cdot 2 + .25 \cdot 2 + .20 \cdot 3 + .18 \cdot 2 + .05 \cdot 3 = 2.25$$

- In average,  $\phi_1$  reduces the bits per symbol over the fixed-length code from 3 to 2.25, about 25%

Definitions

Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

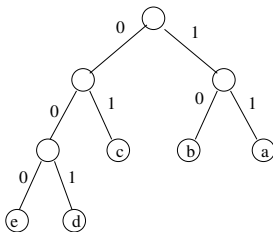
Optimal prefix codes

data compression  
prefix codes  
Huffman code

Approximation algorithms

# Fixed versus variable length codes: Example.

Is 2.25 the maximum compression? Consider the prefix-code  $\phi_2$ :



$$\alpha = .32 \cdot 2 + .25 \cdot 2 + .20 \cdot 2 + .18 \cdot 3 + .05 \cdot 3 = 2.23$$

is that the best? (the maximum compression using a prefix code)

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Optimal prefix code.

Given a text, an **optimal prefix code** is a prefix code that minimizes the total number of bits needed to encode the text, i.e.,  $\alpha$ .

Intuitively, in the prefix tree of an optimal prefix code, symbols with high frequencies should have small depth and symbols with low frequency should have large depth.

Before describing the algorithm we analyze some properties of optimal prefix trees.

## Definitions

### Fractional Knapsack

- Some selection criteria
- Highest  $v/w$
- 0-1 Knapsack

### Scheduling

- Interval scheduling
- Weighted activity selection
- Minimizing lateness

### Optimal prefix codes

- data compression
- prefix codes**
- Huffman code

### Approximation algorithms

# A property of optimal prefix trees.

A binary tree  $T$  is **full** if every interior node has two sons.

## Lemma

*The prefix tree describing an optimal prefix code is full.*

## Proof.

- Let  $T$  be the prefix tree of an optimal code, and suppose it contains a  $u$  with a unique son  $v$ .
- If  $u$  is the root, construct  $T'$  by deleting  $u$  and using  $v$  as root. Otherwise, let  $w$  be the father of  $u$ . Construct  $T'$  by deleting  $u$  and connecting directly  $v$  to  $w$ .
- In both cases  $T'$  is a prefix tree and all the leaves in the subtree rooted at  $v$  reduce its height by 1 in  $T'$ .
- $T'$  yields a code with less bits, so  $T$  is not optimal.

### Definitions

#### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

#### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

#### Optimal prefix codes

data compression  
prefix codes  
Huffman code

#### Approximation algorithms

# Greedy approach: Huffman code

Greedy approach due to David Huffman (1925-99) in 1952, while he was a PhD student at MIT



Wish to produce a labeled binary full tree, in which the leaves are as close to the root as possible. Moreover symbols with low frequency will be placed deeper than the symbol with high frequency.

## Definitions

### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

prefix codes

**Huffman code**

### Approximation algorithms

# Greedy approach: Huffman code

- Given the frequencies  $f(x)$  for every  $x \in \Sigma$
- The algorithm keeps a dynamic sorted list in a priority queue  $Q$ .
- Construct a tree in bottom-up fashion
  - Insert symbols as *leaves* with key  $f$ .
  - Extract the two first elements of  $Q$  and join them by a new *virtual node* with key the sum of the  $f$ 's of its children. Insert the new node in  $Q$ .
- When  $Q$  has size 1, the resulting tree will be the prefix tree of an optimal prefix code.

## Definitions

## Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

## Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

## Optimal prefix codes

data compression

prefix codes

Huffman code

## Approximation algorithms



# Huffman Coding: Construction of the tree.

## Huffman $\Sigma, S$

Given  $\Sigma$  and  $S$  {compute the frequencies  $\{f\}$ }

Construct priority queue  $Q$  of leaves for  $\Sigma$ , ordered by increasing  $f$

```
while  $Q.size() > 1$  do  
    create a new node  $z$   
     $x = \text{Extract-Min}(Q)$   
     $y = \text{Extract-Min}(Q)$   
    make  $x, y$  the sons of  $z$   
     $f(z) = f(x) + f(y)$   
    Insert  $(Q, z, f(z))$   
end while  
 $\phi = \text{Extract-Min}(Q)$ 
```

If  $Q$  is implemented with a Heap, takes time  $O(n \lg n)$ .

Definitions

Fractional  
Knapsack

Some selection  
criteria  
Highest  $v/w$   
0-1 Knapsack

Scheduling

Interval scheduling  
Weighted activity  
selection  
Minimizing lateness

Optimal prefix  
codes

data compression  
prefix codes  
Huffman code

Approximation  
algorithms

# Example

Consider the text: *for each rose, a rose is a rose, the rose*

with  $\Sigma = \{\text{for/ each/ rose/ a/ is/ the/ ,/ b}\}$

Frequencies:

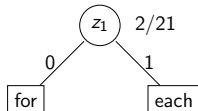
$f(\text{for}) = 1/21$ ,  $f(\text{rose}) = 4/21$ ,  $f(\text{is}) = 1/21$ ,

$f(\text{a}) = 2/21$ ,  $f(\text{each}) = 1/21$ ,  $f(,) = 2/21$ ,

$f(\text{the}) = 1/21$ ,  $f(\text{b}) = 9/21$ .

Priority Queue:

$Q = ((\text{for}:1/21), (\text{each}:1/21), (\text{is}:1/21), (\text{a}:2/21), (, :2/21), (\text{the}:2/21),$   
 $(\text{rose}:4/21), (\text{b}: 9/21))$



Then,  $Q = ((\text{is}:1/21), (\text{a}:2/21), (, :2/21), (\text{the}:2/21), (z_1:2/21), (\text{rose}:4/21),$   
 $(\text{b}:9/21))$

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

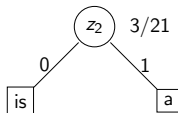
prefix codes

Huffman code

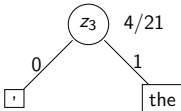
Approximation  
algorithms

# Example.

$Q=((\text{is}:1/21), (\text{a}:2/21), (',:2/21), (\text{the}:2/21), (z_1:2/21), (\text{rose}:4/21), (\text{b}:9/21))$



Then,  $Q=((',:2/21), (\text{the}:2/21), (z_1:3/21), (z_2:3/21), (\text{rose}:4/21), (\text{b}:9/21))$



Then,  $Q=((z_1:2/21), (z_2:3/21), (\text{rose}:4/21), (z_3:4/21), (\text{b}:9/21))$

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

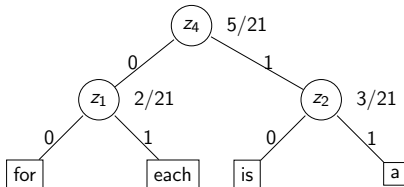
### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# Example.

$Q=((z_1:2/21), (z_2:3/21), (rose:4/21), (z_3:4/21), (b:9/21))$



Then,  $Q=((rose:4/21), (z_3:4/21), (z_4:5/21), (b:9/21))$

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

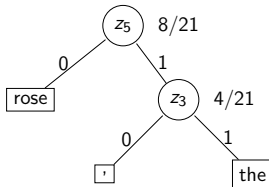
prefix codes

Huffman code

Approximation algorithms

# Example.

$$Q=((\text{rose}:4/21), (z_3:4/21), (z_4:5/21), (b:9/21))$$



Then,  $Q=((z_4:5/21), (z_5:8/21), (b:9/21))$

## Definitions

## Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

## Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

## Optimal prefix codes

data compression

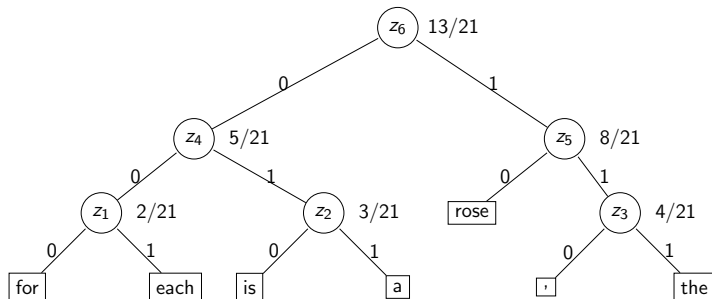
prefix codes

**Huffman code**

## Approximation algorithms

# Example.

$$Q=((z_4:5/21), (z_5:8/21), (b:9/21))$$



$$\text{Then, } Q=((b:9/21),(z_6:13/21))$$

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest  $v/w$   
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

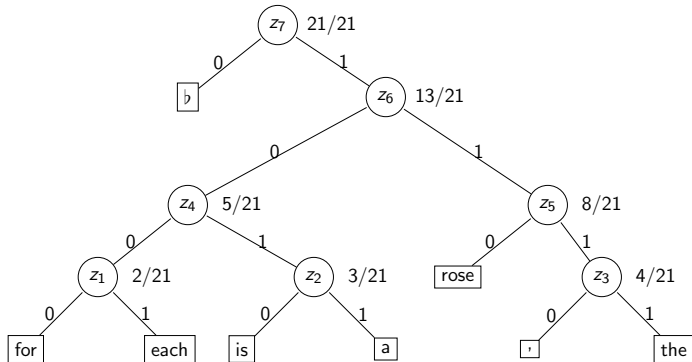
### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# Example.

$$Q=((b:9/21),(z_6:13/21))$$



$$\text{Then, } Q=((z_7:21/21))$$

## Definitions

### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

prefix codes

Huffman code

### Approximation algorithms

# Example

- Therefore *for each rose, a rose is a rose, the rose* is Huffman coded as

10000100101101110010110110010100101101101110011110110

- The solution is not unique!
- The encoded length is 53, and compression is  $53/21 = 2.523 \dots$
- With a fixed size code, we need 4 bits per symbol, length 84 bits instead of 53.
- Why does the Huffman's algorithm produce an optimal prefix code?

## Definitions

## Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

## Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

## Optimal prefix codes

data compression  
prefix codes  
Huffman code

## Approximation algorithms



# Correctness

## Theorem (Greedy property)

*Let  $\Sigma$  be an alphabet, and let  $x, y$  be two symbols with the lowest frequency. There is an optimal prefix code  $\phi$  in which  $|\phi(x)| = |\phi(y)|$  and both codes differ only in the last bit.*

## Proof.

Assume that  $T$  is optimal but that  $x$  and  $y$  have not the same code length. In  $T$  there must be two symbols  $a$  and  $b$  siblings at max. depth. Assume  $f(a) \leq f(b)$  and  $f(x) \leq f(y)$ , otherwise sort them accordingly.

We construct  $T'$  by exchanging  $x$  with  $a$  and  $y$  with  $b$ . As  $f(x) \leq f(a)$  and  $f(y) \leq f(b)$  then  $B(T') \leq B(T)$ . So  $T'$  is optimal and verifies the property. □

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Correctness

## Theorem (Optimal substructure)

*Assume  $T'$  is an optimal prefix tree for  $(\Sigma - \{x, y\}) \cup \{z\}$  where  $x, y$  are two symbols with the lowest frequencies, and  $z$  has frequency  $f(x) + f(y)$ . The  $T$  obtained from  $T'$  by making  $x$  and  $y$  children of  $z$  is an optimal prefix tree for  $\Sigma$ .*

## Proof.

Let  $T_0$  be any prefix tree for  $\Sigma$ . We must show  $B(T) \leq B(T_0)$ .

By the previous result, we only need to consider  $T_0$  where  $x$  and  $y$  are siblings, their parent has frequency  $f(x) + f(y)$ .

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# Correctness

- Let  $T'_0$  be obtained by removing  $x, y$  from  $T_0$ . As  $T'_0$  is a prefix tree for  $(\Sigma - \{x, y\}) \cup \{z\}$ , then  $B(T'_0) \geq B(T')$ .
- Comparing  $T_0$  with  $T'_0$  we get,

$$B(T_0) = B(T'_0) + f(x) + f(y),$$

$$B(T) = B(T') + f(x) + f(y) = B(T).$$

- Putting together the three identities, we get  $B(T) \leq B(T_0)$ .

End Proof

Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# More on Huffman codes

Huffman is optimal under assumptions:

- The compression is **lossless**, i.e. *uncompressing the compressed file yield the original file.*
- We must know the alphabet beforehand (characters, words, etc.),
- We must pre-compute the frequencies of symbols, i.e. read the data twice, **which make it very slow for many real applications.**
- A good source for extensions of Huffman encoding compression is the Wikipedia article on it:  
[https://en.wikipedia.org/wiki/Huffman\\_coding](https://en.wikipedia.org/wiki/Huffman_coding).

Definitions

Fractional  
Knapsack

Some selection  
criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity  
selection

Minimizing lateness

Optimal prefix  
codes

data compression

prefix codes

Huffman code

Approximation  
algorithms

# Approximation algorithms

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

- Many times the Greedy strategy yields a **feasible solution** with value which is **near** to the optimum solution.
- In many practical cases, when finding the global optimum is hard, the greedy may yield a *good enough* feasible solution: **An approximation to the optimal solution.**
- An **approximation algorithm** for the problem always computes a close valid output. Heuristics also could yield good solutions, but they do not have a theoretical guarantee of closeness.
- Greedy is one of the algorithmic techniques used to design approximations algorithms.

# Greedy and approximation algorithms

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

- For any optimization problem, let  $c(*)$  be the value of the optimization function, let  $\mathcal{Apx}$  be an algorithm, that for each input  $x$  produces a valid solution  $\mathcal{Apx}(x)$  to  $x$ . Let  $\text{opt}(x)$  be the cost of an optimal solution to  $x$ .
- We want to design a **fast** algorithm that produce solutions **close** to the optimal.
- For a NP-hard problem, we don't know if it has polynomial time algorithms, we want to design algorithms that are fast (polynomial) and that outputs **good** solutions always.

# Approximation algorithm: Formal definition

- For a given optimization problem, let  $\mathcal{A}px$  be an algorithm, that for each input  $x$  produces a valid solution with cost  $\mathcal{A}px(x)$  to  $x$ . Let  $\text{opt}(x)$  be the cost of an optimal solution to  $x$ .
- For  $r > 1$ ,  $\mathcal{A}px$  is an  **$r$ -approximation algorithm** if, for any input  $x$ :

$$\frac{1}{r} \leq \frac{\mathcal{A}px(x)}{\text{opt}(x)} \leq r.$$

- $r$  is called **the approximation ratio**.
- Given an optimization problem, for any input  $x$ , we require
  - in a MAX problem,  $\mathcal{A}px(x) \leq \text{opt}(x) \leq r\mathcal{A}px(x)$ .
  - in a MIN problem,  $\text{opt}(x) \leq \mathcal{A}px(x) \leq r\text{opt}(x)$ .

## Definitions

### Fractional Knapsack

Some selection criteria  
Highest v/w  
0-1 Knapsack

### Scheduling

Interval scheduling  
Weighted activity selection  
Minimizing lateness

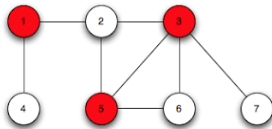
### Optimal prefix codes

data compression  
prefix codes  
Huffman code

### Approximation algorithms

# An easy example: VERTEX COVER problem

Recall the problem of Vertex cover: Given a graph  $G = (V, E)$  with  $|V| = n, |E| = m$  find the minimum set of vertices  $S \subseteq V$  such that it covers every edge of  $G$ .



**GreedyVC** for  $I: G = (V, E)$

$E' = E, S = \emptyset,$

**while**  $E' \neq \emptyset$  **do**

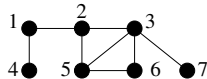
    Pick  $e \in E'$ , say  $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

**end while**

**return**  $S$



Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

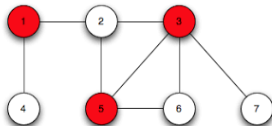
Huffman code

Approximation algorithms



# An easy example: VERTEX COVER problem

Given a graph  $G = (V, E)$  with  $|V| = n, |E| = m$  find the minimum set of vertices  $S \subseteq V$  such that it covers every edge of  $G$ .



**GreedyVC**  $G = (V, E)$

$E' = E, S = \emptyset,$

**while**  $E' \neq \emptyset$  **do**

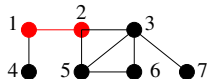
    Pick  $e \in E'$ , say  $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

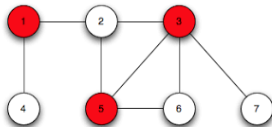
**end while**

**return**  $S$



# An easy example: VERTEX COVER problem

Given a graph  $G = (V, E)$  with  $|V| = n, |E| = m$  find the minimum set of vertices  $S \subseteq V$  such that it covers every edge of  $G$ .



**GreedyVC**  $G = (V, E)$

$E' = E, S = \emptyset,$

**while**  $E' \neq \emptyset$  **do**

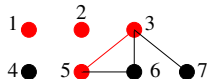
    Pick  $e \in E'$ , say  $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

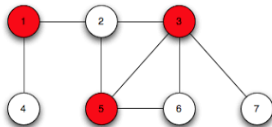
**end while**

**return**  $S$



# An easy example: VERTEX COVER problem

Given a graph  $G = (V, E)$  with  $|V| = n, |E| = m$  find the minimum set of vertices  $S \subseteq V$  such that it covers every edge of  $G$



**GreedyVC**  $G = (V, E)$

$E' = E, S = \emptyset,$

**while**  $E' \neq \emptyset$  **do**

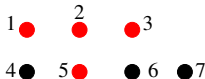
    Pick  $e \in E'$ , say  $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

**end while**

**return**  $S$



# An easy example: Vertex cover

## Theorem

**GreedyVC** runs in  $O(m + n)$  steps. Moreover, if  $S$  is solution computed on input  $G$ ,  $|S| \leq 2\text{opt}(G)$ .

## Proof.

- The edges selected among by **GreedyVC** do not share any vertex.
- Therefore, an optimal solution must have at least one of the two endpoints of each edge while **GreedyVC** takes both.
- So,  $|S| \leq 2\text{opt}(G)$ .



Definitions

Fractional Knapsack

Some selection criteria

Highest v/w

0-1 Knapsack

Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

Optimal prefix codes

data compression

prefix codes

Huffman code

Approximation algorithms

# An easy example: Vertex cover

The decision problem for Vertex Cover: given  $G$  and  $k$ , does  $G$  have a vertex cover with  $k$  or less vertices?, is NP-complete.

Moreover, unless  $P=NP$ , vertex cover can not be approximated within a factor  $r \leq 1.36$

## Definitions

### Fractional Knapsack

Some selection criteria

Highest  $v/w$

0-1 Knapsack

### Scheduling

Interval scheduling

Weighted activity selection

Minimizing lateness

### Optimal prefix codes

data compression

prefix codes

Huffman code

### Approximation algorithms