CSCI 470: Greedy Algorithms: An activity-selection problem, Knapsack problem

Vijay Chaudhary

November 22, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. Greedy Algorithms

2. An activity Selection Problem

3. Knapsack problem

Greedy Algorithms

Greedy Algorithms

An activity Selection Problem

An activity Selection Problem

- Suppose we have a set $S = \{a_1, a_2, ..., a_n\}$ of n proposed ***activities*** that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.
- Each activity a_i has a ***start time*** s_i and a ***finish time*** f_i, where $0 \leq s_i < f_i < \infty$.
- If selected, activity a_i takes place during the half-open time interval $[s_i, f_i)$.

Activity Selection

- Activities a_i and a_j are ***compatible*** if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. That is, a_i and a_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$.
- in the ***activity-selection problem***, we wish to select a maximum-size subset of mutually compatible activities.
- We assume that the activities are sorted in monotonically increasing order of finish time:

$$f_1 \leq f_2 \leq f_3 \leq \ldots f_{n-1} \leq f_n \tag{1}$$

Activity Selection: Example

(We shall see later the advantage that this assumption provides.) For example, consider the following set S of activities:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- For this example, the subset $\{a_3, a_9, a_{11}\}$ consists of mutually compatible activities.
- $\{a_1, a_4, a_8, a_{11}\}$ or $\{a_2, a_4, a_9, a_{11}\}$ are the largest subset of mutually compatible activities.

Optimal substructure of the activity-selection problem

- Let us denote that S_{ij} the set of activities that start after activity a_i finishes and that finish before activity a_j starts.

Optimal substructure of the activity-selection problem

- Let us denote that S_{ij} the set of activities that start after activity a_i finishes and that finish before activity a_j starts.
- Let A_{ij} be a maximum set of mutually compatible activities in S_{ij}, which includes some activity a_k.

Optimal substructure of the activity-selection problem

- Let us denote that S_{ij} the set of activities that start after activity a_i finishes and that finish before activity a_j starts.
- Let A_{ij} be a maximum set of mutually compatible activities in S_{ij}, which includes some activity a_k.
- With a_k in an optimal solution, we are left with two subproblems:

Optimal substructure of the activity-selection problem

- Let us denote that S_{ij} the set of activities that start after activity a_i finishes and that finish before activity a_j starts.
- Let A_{ij} be a maximum set of mutually compatible activities in S_{ij}, which includes some activity a_k.
- With a_k in an optimal solution, we are left with two subproblems:
  - finding mutually compatible activities in the set S_{ik} (activities that start after activity a_i finishes and that finish before activity a_k starts)

Optimal substructure of the activity-selection problem

- Let us denote that S_{ij} the set of activities that start after activity a_i finishes and that finish before activity a_j starts.
- Let A_{ij} be a maximum set of mutually compatible activities in S_{ij}, which includes some activity a_k.
- With a_k in an optimal solution, we are left with two subproblems:
  - finding mutually compatible activities in the set S_{ik} (activities that start after activity a_i finishes and that finish before activity a_k starts)
  - and finding mutually compatible activities in the set S_{kj} (activities that start after activity a_k finishes and that finish before activity a_j starts).

Optimal substructure

- Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$ so that A_{ik} contains the activities in A_{ij} that finish before a_k starts and A_{kj} contains the activities in A_{ij} that start after a_k finishes.

Optimal substructure

- Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$ so that A_{ik} contains the activities in A_{ij} that finish before a_k starts and A_{kj} contains the activities in A_{ij} that start after a_k finishes.
- That gives us $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$, and so the maximum-size set A_{ij} of mutually compatible activities in S_{ij} consists of $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ activities.

Optimal substructure

- Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$ so that A_{ik} contains the activities in A_{ij} that finish before a_k starts and A_{kj} contains the activities in A_{ij} that start after a_k finishes.
- That gives us $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$, and so the maximum-size set A_{ij} of mutually compatible activities in S_{ij} consists of $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ activities.
- The optimal solution to A_{ij} must also include optimal solutions to the two subproblems for S_{ik} and S_{kj}.

Recurrence of optimal substructure

- $c[i, j] = c[i, k] + c[k, j] + 1$
- $$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset, \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq \emptyset. \end{cases}$$
- We could then develop a recursive algorithm and memoize it, or we could work bottom-up and fill in table entries as we go along.

Making the greedy choice

- What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems?

Making the greedy choice

- What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems?
- Intuition suggests that we should choose an activity that leaves the resource available for as many other activities as possible.

Making the greedy choice

- What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems?
- Intuition suggests that we should choose an activity that leaves the resource available for as many other activities as possible.
- Now, of the activities we end up choosing, one of them must be the first one to finish.

Making the greedy choice

- What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems?
- Intuition suggests that we should choose an activity that leaves the resource available for as many other activities as possible.
- Now, of the activities we end up choosing, one of them must be the first one to finish.
- Our intuition tells us, therefore, to choose the activity in S with the earliest finish time, since that would leave the resource available for as many of the activities that follow it as possible.

Making the greedy choice

- If we make the greedy choice, we have only one remaining subproblem to solve: finding activities that start after a_1 finishes.

Making the greedy choice

- If we make the greedy choice, we have only one remaining subproblem to solve: finding activities that start after a_1 finishes.
- Why don't we have to consider activities that finish before a_1 starts?

Making the greedy choice

- If we make the greedy choice, we have only one remaining subproblem to solve: finding activities that start after a_1 finishes.
- Why don't we have to consider activities that finish before a_1 starts?
- We have that $s_1 < f_1$, and f_1 is the earliest finish time of any activity, and therefore no activity can have a finish time less than or equal to s_1.

Making the greedy choice

- If we make the greedy choice, we have only one remaining subproblem to solve: finding activities that start after a_1 finishes.
- Why don't we have to consider activities that finish before a_1 starts?
- We have that $s_1 < f_1$, and f_1 is the earliest finish time of any activity, and therefore no activity can have a finish time less than or equal to s_1.
- Thus, all activities that are compatible with activity a_1 must start after a_1 finishes.

Making the greedy choice

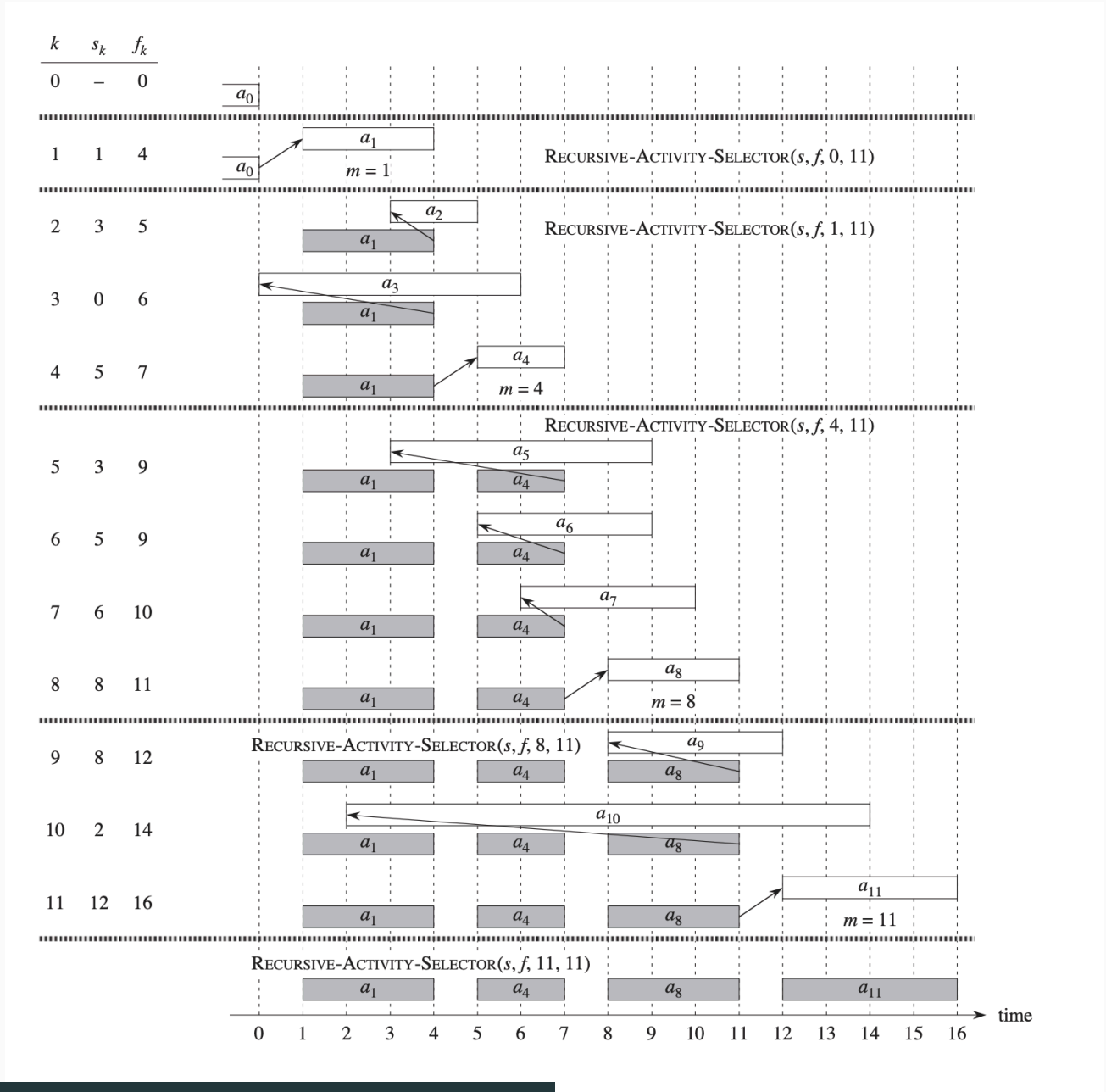
- Let $S_k = \{a_i \in S : s_i \geq f_k\}$ be the set of activities that start after activity a_k finishes.
- If we make the greedy choice of activity a_1, then S_1 remains as the only subproblem to solve.
- Optimal substructure tells us that if a_1 is in the opitmal solution, then an optimal solution to the original problem consists of activity a_1 and all activities in an optimal solution to the subproblem S_1.
- There is a proof that shows this intuition is actually true in the book pg. 418 (Theorem 16.1). *Left to the readers to go over the proof.*

A recursive greedy algorithm

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

```
1  m = k + 1
2  while m ≤ n and s[m] < f[k] // find the first activity in S_k to finish
3        m = m + 1
4  if m ≤ n
5        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

Illustration



An iterative greedy algorithm

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5        if s[m] ≥ f[k]
6              A = A ∪ {a_m}
7              k = m
8  return A
```

iterative greedy algorithm

- The variable k indexes the most recent addition to A, corresponding to the activity a_k in the recursive version.
- Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A.
-

$$f_k = \max\{f_i : a_i \in A\}. \tag{2}$$

Elements of the greedy strategy

We design greedy algorithms according to the following sequence of steps:

- Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
- Demonstrate optimal substructure by showing that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

Greedy-choice property

- We can assemble a globally optimal solution by making locally optimal (greedy) choices. In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.
- In dynamic programming, we make a choice at each step, but the choice usually depends on the solutions to subproblems. Consequently, we typically solve dynamic-programming problems in a bottom-up manner, progressing from smaller subproblems to larger subproblems.
- In a greedy algorithm, we make whatever choice seems best at the moment and then solve the subproblem that remains. The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to subproblems.

Greedy-choice property

- Unlike dynamic programming, which solves the subproblems before making the first choice, a greedy algorithm makes its first choice before solving any subproblems.
- A dynamic programming algorithm proceeds bottom up, whereas a greedy strategy usually progresses in a top-down fashion, making one greedy choice after another, reducing each given problem instance to a smaller one.

Knapsack problem

0-1 Knapsack problem vs Fractional Knapsack problem

- The 0-1 knapsack problem is the following. A thief robbing a store finds n items. The ith item is worth v_i dollars and weighs w_i pounds, where v_i and w_i are integers. The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W.
- In the fractional knapsack problem, the setup is the same, but the thief can take fractions of items, rather than having to make a binary (0-1) choice for each item.

Knapsack problem

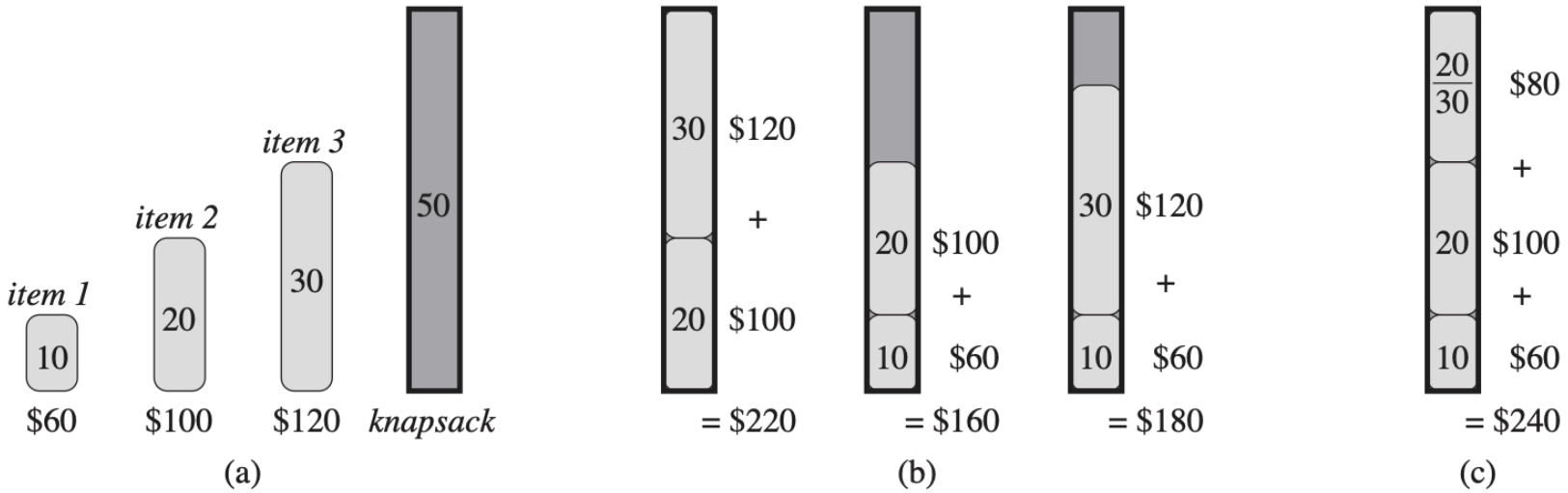


Figure 16.2 An example showing that the greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

Knapsack problem

- 0-1 Knapsack problem cannot be solved using a greedy approach, while fractional knapsack problem can be solved greedily.