# Greedy Algorithms

Jared Saia
University of New Mexico

# Outline

- Greedy Algorithm Intro
- Activity Selection
- Knapsack

# Greedy Algorithms

*"Greed is Good" - Michael Douglas in Wall Street*

- A greedy algorithm always makes the choice that looks best at the moment
- Greedy algorithms do not always lead to optimal solutions, but for many problems they do
- In the next week, we will see several problems for which greedy algorithms produce optimal solutions including: activity selection, fractional knapsack.
- When we study graph theory, we will also see that greedy algorithms can work well for computing shortest paths and finding minimum spanning trees.

# Activity Selection

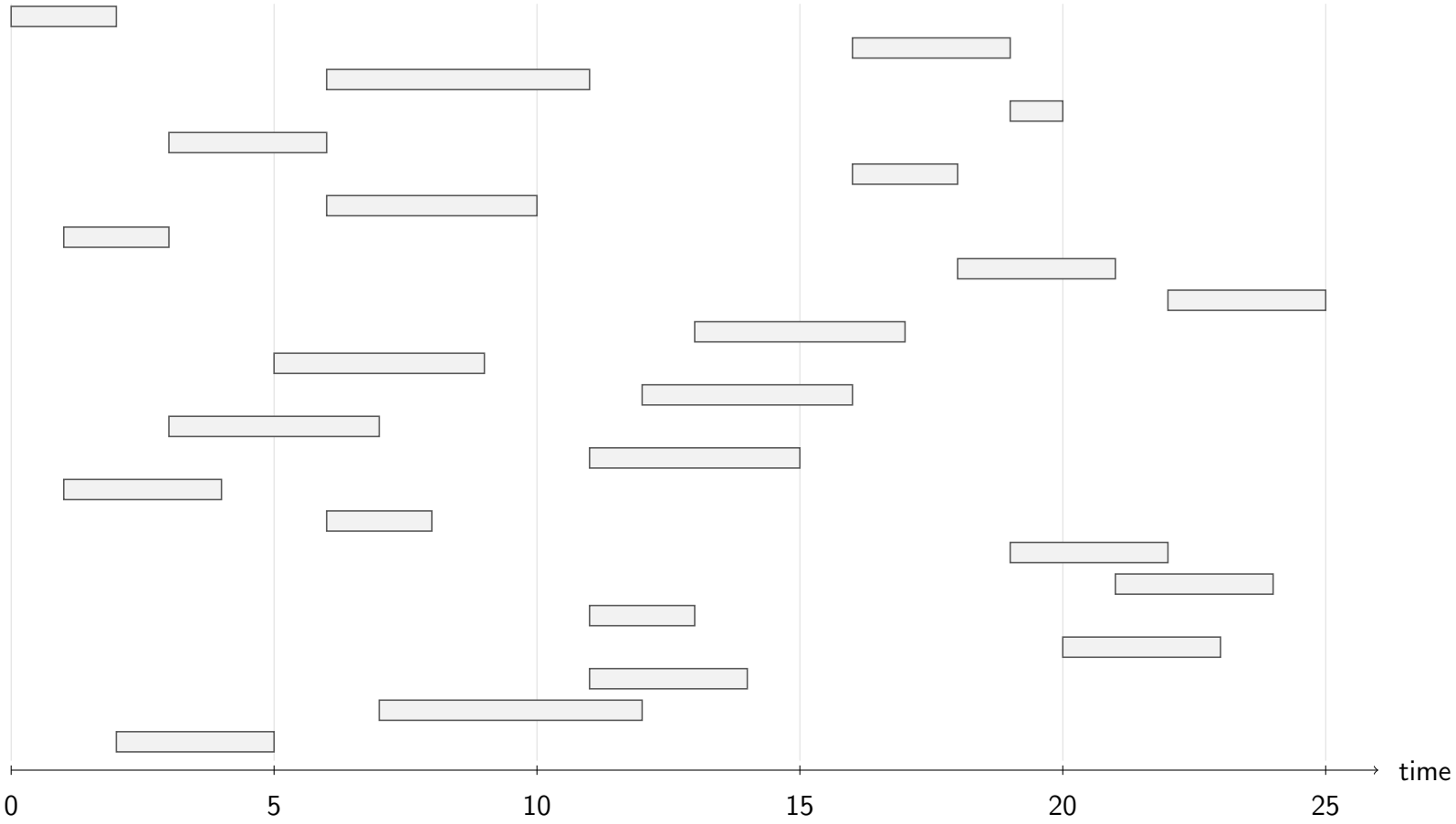
- You are given a list of programs to run on a single processor
- Each program has a start time and a finish time
- However the processor can only run one program at any given time, and there is no preemption (i.e. once a program is running, it must be completed)
- Your goal is to schedule as many activities as possible.

# Another Motivating Problem

- Suppose you are at a film fest, all movies look equally good, and you want to see as many complete movies as possible
- This problem is also exactly the same as the activity selection problem.

# Activity Intervals

Each bar shows one activity, from its start to its finish.



# Ideas

- There are many ways to optimally schedule these activities
- Brute Force: examine every possible subset of the activities and find the largest subset of non-overlapping activities
- Q: If there are $n$ activities, how many subsets are there?
- The book also gives a DP solution to the problem

# Greedy Activity Selector

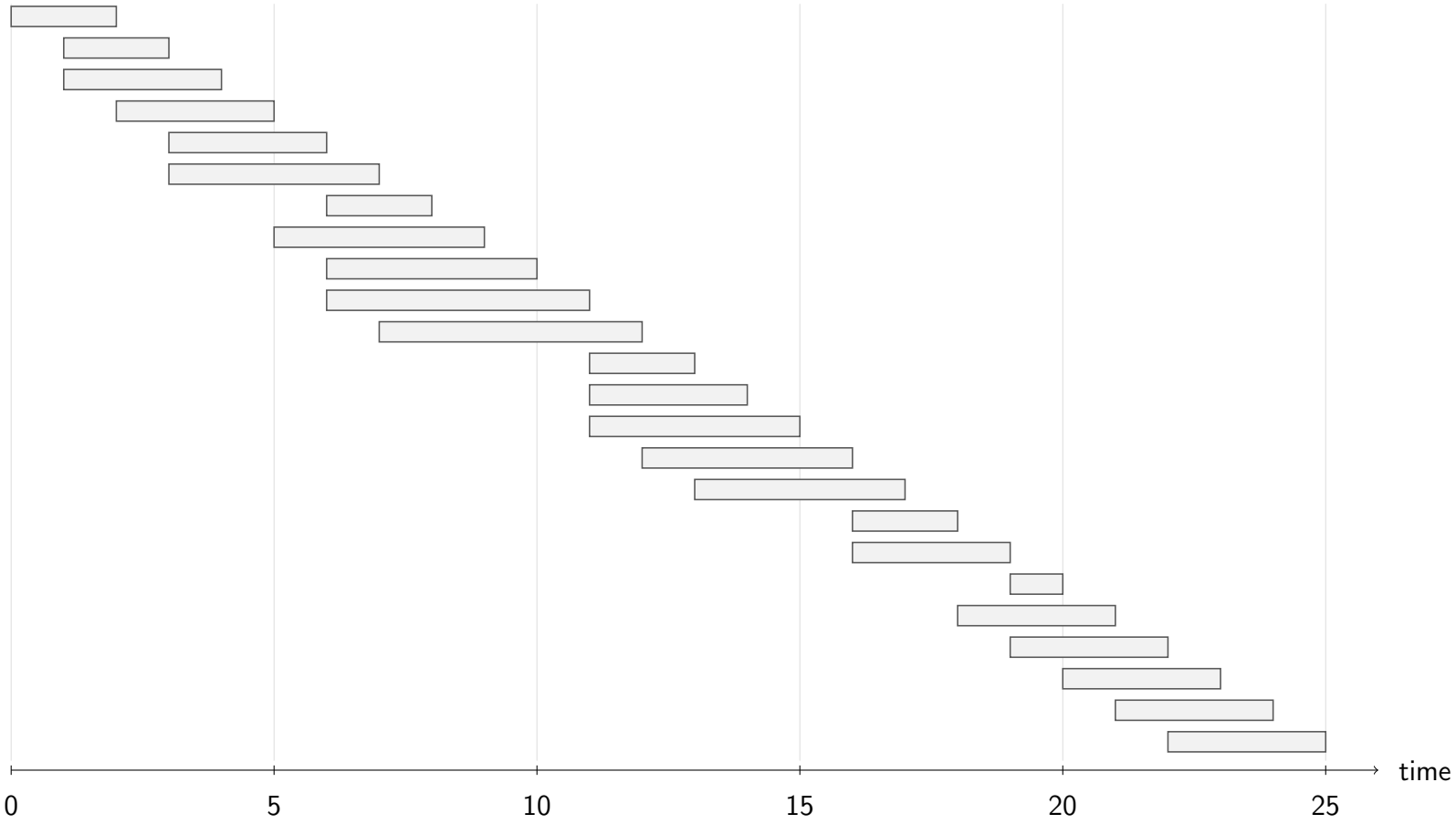
**Activity-Selector**$(\mathcal{S})$

Sort $\mathcal{S}$ by nondecreasing finish time
$A \leftarrow \emptyset$; $t \leftarrow -\infty$
**for each** activity $a$ **in sorted order do**
  **if** $s(a) \geq t$ **then**
    $A \leftarrow A \cup \{a\}$
    $t \leftarrow f(a)$
**return** $A$

Here $s(a)$ and $f(a)$ are the start and finish times of activity $a$.

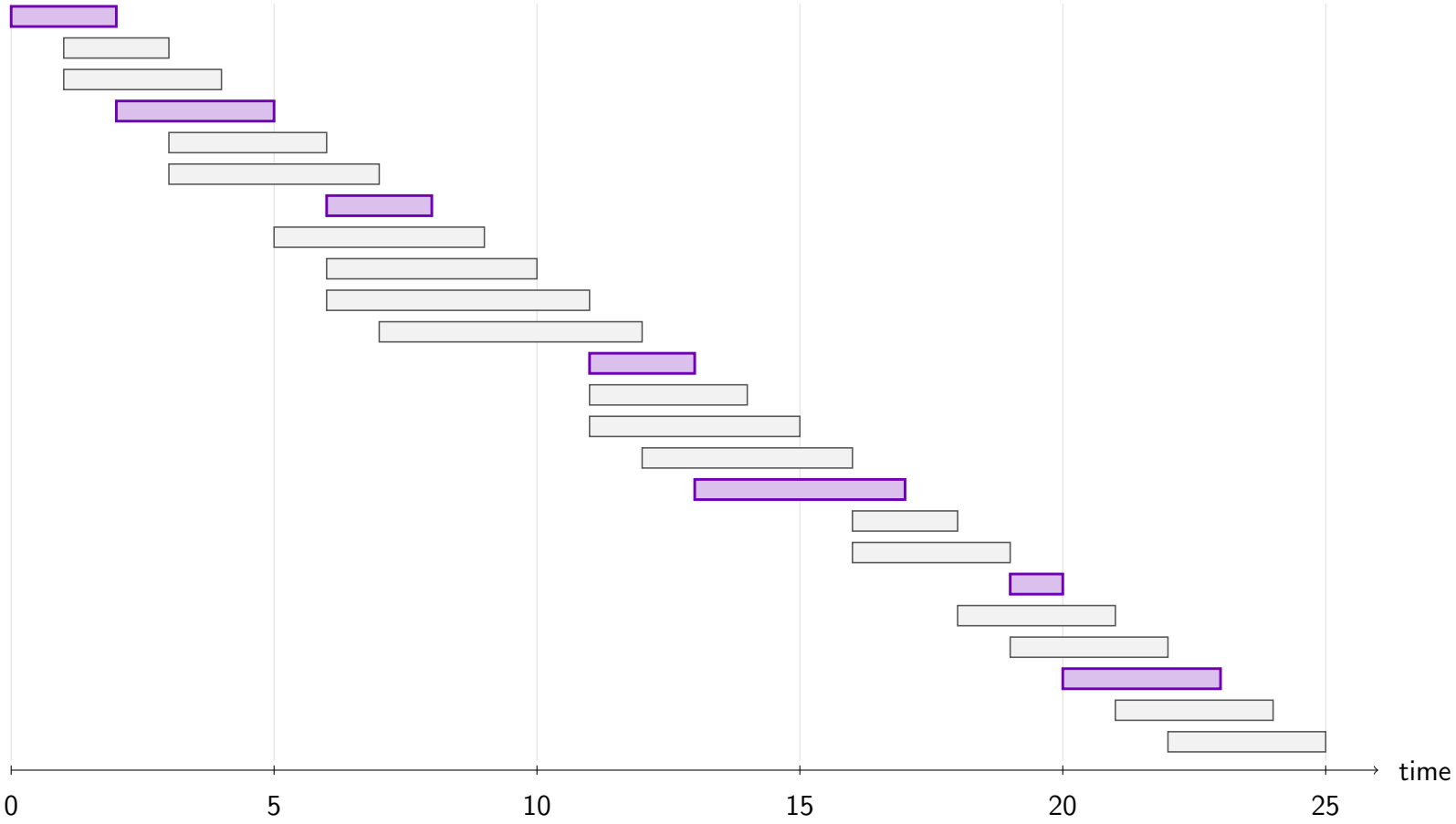
# Sorted by Finish Time

The same activities, now in nondecreasing order of finish time.



# Greedy Activity Selection

Purple bars mark the seven activities selected by greedy.



# Analysis

- Let $n$ be the total number of activities
- The algorithm first sorts the activities by finish time taking $O(n \log n)$
- Then the algorithm visits each activity exactly once, doing a constant amount of work each time. This takes $O(n)$
- Thus total time is $O(n \log n)$

# Greedy Is Optimal

Assume $s(a) < f(a)$ for each activity; touching endpoints are allowed.

- **Claim:** Earliest-finish-time greedy selects a maximum-size set of non-overlapping activities.
- Write the greedy schedule as $A = (a_1, \ldots, a_r)$ and any feasible schedule as $B = (b_1, \ldots, b_m)$, both in chronological order.
- We will show $r \geq m$ by an *exchange argument*: replace activities in $B$ with greedy choices, preserving feasibility and size.

# The Exchange Step

- Let $j$ be the first position where the schedules differ, with $a_j$ and $b_j$ both present. Their first $j-1$ activities agree.
- The activity $b_j$ fits after the common prefix. Thus greedy chooses $a_j$ with $f(a_j) \leq f(b_j)$; ties are allowed.
- The activity $a_j$ fits after the common prefix. It also fits before any following activity, since

$$f(a_j) \leq f(b_j) \leq s(b_{j+1}) \qquad \text{if } j < m.$$

- Thus replacing $b_j$ by $a_j$ keeps $B$ feasible, preserves its size, and extends its agreement with $A$.

# Completing the Exchange Proof

- Repeat the exchange until one schedule is a prefix of the other. Each exchange advances the first differing position.
- If $B$ ends first, then $m \leq r$, as desired.
- If $A$ ends first with $r < m$, the next activity $b_{r+1}$ in the modified $B$ is compatible with all of $A$.
- Greedy could therefore select another activity, contradicting its termination. Hence $r \geq m$ for every feasible $B$, so $A$ is optimal.

# Greedy pattern

- **The problem has a solution that can be given some numerical value**. The "best" (optimal) solution has the highest/lowest value.
- **The solutions can be broken down into steps.** The steps have some order and at each step there is a choice that makes up the solution.
- **The choice is based on what's best at a given moment.** Need a criterion that will distinguish one choice from another.
- Finally, need to **prove** that the solution that you get by making these local choices is indeed optimal

# Activity Selection Pattern

- The value of the solution is the number of non-overlapping activities. The best solution has the highest number.
- The sorting gives the order to the activities. Each step is examining the next activity in order and decide whether to include it.
- In each step, the greedy algorithm chooses the activity which extends the length of the schedule as little as possible

# Greedy versus DP

- A greedy rule commits to a local choice. We must prove that this choice preserves an optimal solution.
- Dynamic programming compares the alternatives in a recurrence and reuses solutions to subproblems.
- For *fractional knapsack*, choosing the greatest value per pound first is optimal. For *0–1 knapsack*, this rule can fail.
- With integer weights and positive integer capacity $W$, there exists a DP solution to 0–1 knapsack that takes $O(nW)$ time. This is *pseudopolynomial*: it depends on $W$, rather than the number of bits needed to encode $W$.

# 0–1 Knapsack

The problem:

- A thief finds $n$ items. Item $i$ has value $v_i \geq 0$ and weight $w_i > 0$, both integers.
- The knapsack has an integer capacity $W \geq 0$.
- The goal is to take a load of maximum total value, with total weight at most $W$.
- Which items should the thief take?

Each item is either taken or not taken; fractional amounts are not allowed.

# Fractional Knapsack

- In this variant, the thief may take a fraction $x_i \in [0, 1]$ of each item. That fraction contributes weight $x_i w_i$ and value $x_i v_i$.
- An item in 0–1 knapsack is like a gold ingot; in fractional knapsack, it is like gold dust.

$$\text{Maximize } \sum_{i=1}^{n} x_i v_i \qquad \text{subject to } \sum_{i=1}^{n} x_i w_i \leq W.$$

# Fractional Greedy

1. Compute the value per pound $\rho_i = v_i/w_i$ for each item.
2. Sort so that $\rho_1 \geq \rho_2 \geq \cdots \geq \rho_n$; break ties arbitrarily.
3. In sorted order, the thief takes as much as possible of each item without exceeding the knapsack's capacity.

After the capacity is filled, all remaining fractions are zero.

# Fractional Runtime

- Sorting $n$ items by value per pound takes $O(n \log n)$ time.
- The scan takes $O(n)$ time. The total is $O(n \log n)$.
- We will prove optimality by comparing greedy with any feasible fractional solution.
- The same density ordering can fail for 0–1 knapsack.

# Three Knapsack Outcomes

Capacity $W = 5$; the items are sorted by value per pound.

| Item $i$ | 1 | 2 | 3 |
|---|---|---|---|
| Weight $w_i$ | 1 | 2 | 3 |
| Value $v_i$ | 3 | 5 | 6 |
| Value per pound $\rho_i$ | 3 | 5/2 | 2 |

- **0–1 density greedy:** items 1 and 2, value $3 + 5 = 8$.
- **0–1 optimum:** items 2 and 3, value $5 + 6 = 11$.
- **Fractional greedy:** items 1 and 2, plus 2/3 of item 3, value $3 + 5 + (2/3)6 = 12$, using all 5 pounds.

# Fractional Optimality

We prove optimality by a direct comparison with any feasible solution.

- The items are sorted so that $\rho_1 \geq \cdots \geq \rho_n \geq 0$, where $\rho_i = v_i/w_i$ and $w_i > 0$.
- Greedy chooses fractions $x_i \in [0, 1]$ and obtains value $V = \sum_i x_i v_i$.
- Any feasible solution $B$ chooses fractions $x'_i \in [0, 1]$ with $\sum_i x'_i w_i \leq W$ and obtains value $V' = \sum_i x'_i v_i$.
- Our goal is to show $V - V' \geq 0$.

# The Cutoff Item

- If $\sum_i w_i \leq W$, greedy takes every item and is optimal because every value is nonnegative.
- Otherwise, greedy fills the knapsack: $\sum_i x_i w_i = W$. Let $k$ be the first index with $x_k < 1$.
- Then $x_i = 1$ for $i < k$, and $x_i = 0$ for $i > k$.

# The Key Inequality

Show that $(x_i - x_i')\rho_i \geq (x_i - x_i')\rho_k$ in each case.

- Q1: $i < k$. Use $x_i = 1$, $x_i' \leq 1$, and $\rho_i \geq \rho_k$.
- Q2: $i = k$. What happens to the two sides?
- Q3: $i > k$. Use $x_i = 0$, $x_i' \geq 0$, and $\rho_i \leq \rho_k$.

For Q3, pay attention to the sign of $x_i - x_i'$.

# Comparing Total Values

Multiply the key inequality by $w_i > 0$ and sum over all items:

$$\begin{aligned} V - V' &= \sum_{i=1}^{n} (x_i - x_i')w_i\rho_i \\ &\geq \sum_{i=1}^{n} (x_i - x_i')w_i\rho_k \\ &= \rho_k \left( \sum_{i=1}^{n} x_i w_i - \sum_{i=1}^{n} x_i' w_i \right) \\ &= \rho_k \left( W - \sum_{i=1}^{n} x_i' w_i \right) \geq 0. \end{aligned}$$

# Why the Final Bound Holds

- In the case $\sum_i w_i > W$, greedy uses exactly the capacity: $\sum_i x_i w_i = W$.
- Feasibility of $B$ gives $\sum_i x'_i w_i \leq W$.
- Since $\rho_k \geq 0$,

$$\rho_k \left( \sum_i x_i w_i - \sum_i x'_i w_i \right) \geq 0.$$

- Thus $V \geq V'$ for every feasible fractional solution $B$. Together with the all-items-fit case, this proves greedy is optimal.