Greedy Algorithms

Algorithms

Greedy Algorithms

- Many algorithms run from stage to stage
  - At each stage, they make a decision based on the information available
- A Greedy algorithm makes decisions
  - At each stage, using locally available information, the greedy algorithm makes an optimal choice

- Sometimes, greedy algorithms give an overall optimal solution
- Sometimes, greedy algorithms will not result in an optimal solution but often in one good enough

Divisible Items Knapsack Problem

- Given a set of items S
  - Each item has a weight $w(x)$
  - Each item has a value $v(x)$
- Select a subset $M \subset S$
  - Constraint: $\sum_{x \in M} w(x) < W$
  - Objective Function: $\sum_{x \in M} v(x) \longrightarrow \max$

Divisible Items Knapsack Problem

- Order all items by impact
  - $\textbf{impact}(x) = \dfrac{v(x)}{w(x)}$
- In order of impact (highest first), ask whether you want to include the item
  - And you include it if the sum of the weights of the items already selected is smaller than W

Optimal Rental

- Set of activities $S = \{a_1, a_2, \ldots, a_n\}$
  - Each activity has a start time and a finish time
    - $0 \le s_i < f_i < \infty$
  - Each activity needs to use your facility
  - Only one activity at a time
  - Make the rental agreements that maximize the number of rentals

Optimal Rental

- Two activities a_i and a_j are compatible iff
  - $[s_i, f_i) \cap [s_j, f_j) = \varnothing$

- This means that activity $i < j$ finishes before activity j

Optimal Rental

- Example:

i	1	2	3	4	5	6	7	8	9	10
si	1	3	0	2	6	5	6	15	18	19
fi	6	7	9	12	13	15	18	19	20	21

- A compatible set is $\{A_1, A_5, A_8, A_{10}\}$
- Another compatible set is $\{A_3, A_9\}$

Optimal Rental

- Optimal rental with a dynamic programming algorithm
  - Subproblems: Define S_{ik} to be the set of activities that start after a_i finishes and finish before a_k starts

i	1	2	3	4	5	6	7	8	9	10
si	1	3	0	2	6	5	6	15	18	19
fi	6	7	9	12	13	15	18	19	20	21

$$S_{1,8} = \{a_5\}$$

Optimal Rental

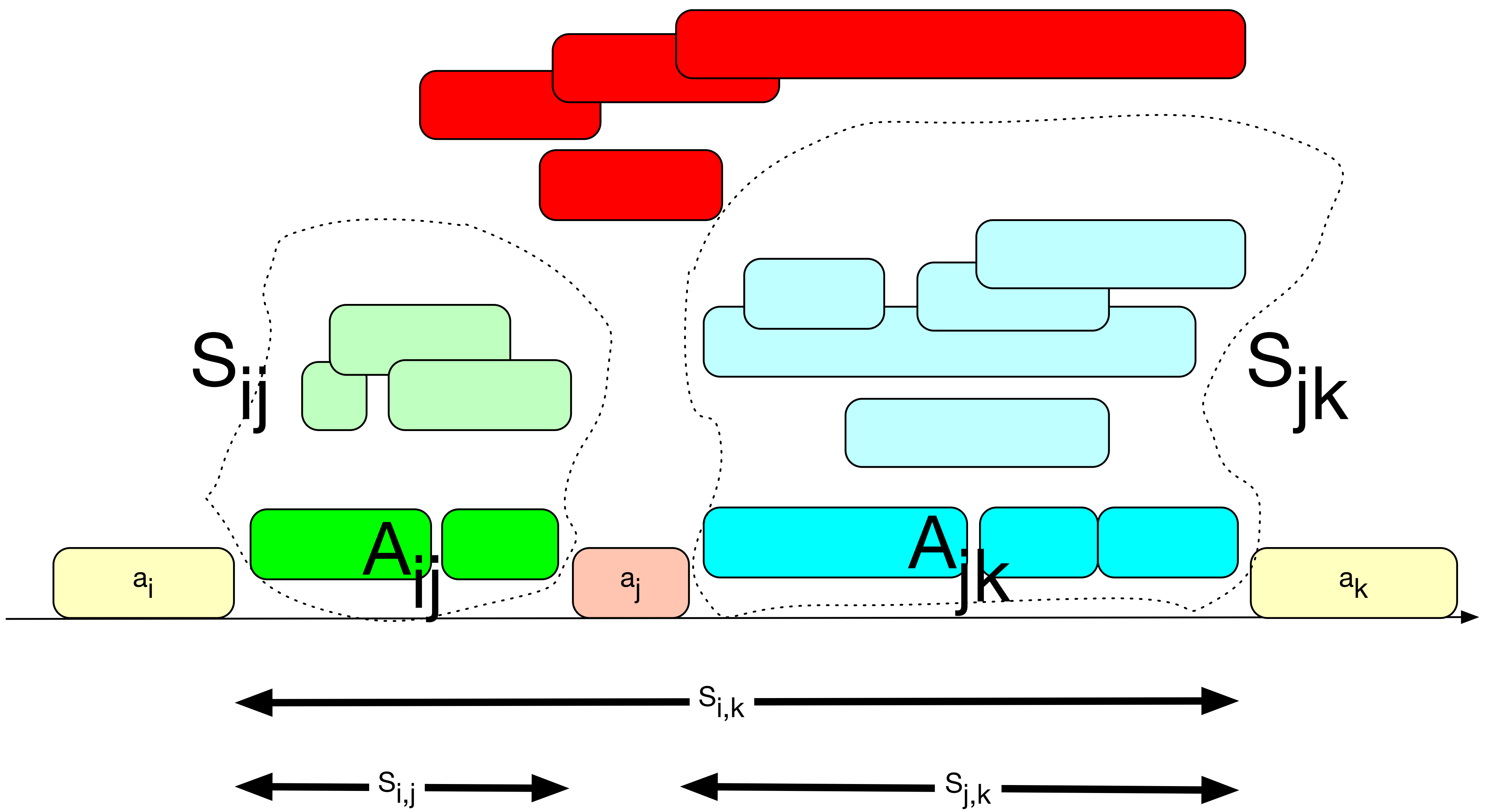
- We want to find an optimal rental plan for S_{ik}
  - Assume that there is an optimal solution that contains activity $a_j \in S_{i,k}$
  - By selecting a_j, we need to decide what to do with the time before a_j starts and after a_j finishes
  - These sets are S_{ij} and S_{jk}

Optimal Rental

- Assume that a_j is part of an optimal solution $A_{i,k}$ for $S_{i,k}$
  - Then $A_{i,k}$ is divided into the ones that end before a_j and the ones that start after a_j
  - $A_{i,j} = A_{i,k} \cap S_{i,k}$ $\quad A_{j,k} = A_{i,k} \cap S_{j,k}$

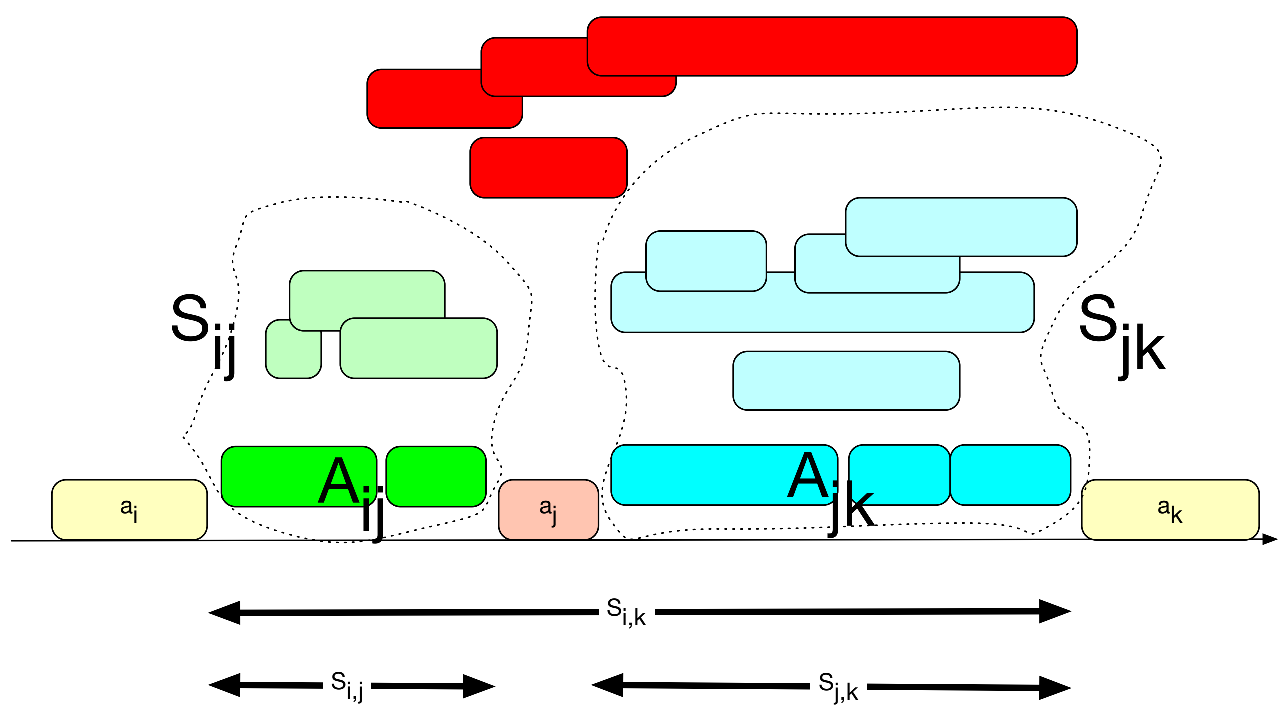
$$A_{i,k} = A_{i,j} \cup \{a_j\} \cup A_{j,k}$$

Optimal Rental



Optimal Rental

- Clearly, $A_{i,j}$ is an optimal solution for $S_{i,j}$
- $A_{j,k}$ is an optimal solution for $S_{j,k}$
- For if not, we could construct a better solution for $S_{i,k}$



Optimal Rental

- We can therefore solve recursively the problem for $S_{i,k}$ by looking at all possible activities for a_j
  - Define $C[i,k] =$ Max number of compatible activities in $S_{i,k}$
  - Then:

$$C[i,k] = \max(0, \max\left(C[i,j] + C[j,k] + 1 \,|\, a_j \in S_{i,k}\right))$$

    - The 0 is necessary because there might be no activity in $S_{i,k}$

Optimal Rental

- The recursion leads to a nice dynamic programming problem

$$C[i,k] = \max(0, \max\left(C[i,j] + C[j,k] + 1 \mid a_j \in S_{i,k}\right))$$

Optimal Rental

- But can we do better?

Optimal Rental

- Start out with the initial problem
  - Select the activity that finishes first
    - this would be a_1
  - This leaves most space for all other activities
    - Call S_1 the set of activities compatible with a_1
      - These are those starting after a_1
    - Similarly, call S_k the set of activities starting after a_k

Optimal Rental

- Theorem: For any non-empty problem S_k let a_m be the activity with the smallest end time. Then a_m is contained in an optimal solution

- Proof:
  - Let A_k be a solution
    - i.e. the maximum sized compatible subset in S_k
    - Let $a_1 \in A_k$ be the activity with earliest finish time
    - If $a_m = a_1$ then we are done

Optimal Rental

- Theorem: For any non-empty problem S_k let a_m be the activity with the smallest end time. Then a_m is contained in an optimal solution

- Proof:
  - Otherwise replace a_1 with a_m in A_k
    - $A'_k = A_k - \{a_1\} \cup \{a_m\}$
    - Since a_m is the first to finish, this is a set of compatible activities
    - Therefore, there exists an optimal solution with a_m

Optimal Rental

- Result of the Theorem:
    - We can find an optimal solution (but not necessarily all optimal solutions) by always picking the first one to finish.

Optimal Rental

- Example

i	1	2	3	4	5	6	7	8	9	10
si	1	3	0	2	6	5	6	15	18	19
fi	6	7	9	12	13	15	18	19	20	21

- Select a_1
- Exclude a_2, a_3, and a_4 as incompatible
- Choose a_5, a_8, and a_{10} for the complete solution

Greedy Algorithms

- Greedy algorithms
  - Determine the optimal substructure
  - Develop a recursive solution
  - Show that making the greedy choice is best
  - Show that making the greedy choice leads to a similar subproblem
  - Obtain a recursive algorithm
  - Convert the recursive algorithm to an iterative algorithm

Coin Change Problem

- How to make change for a given amount using minimum number of coins with denominations $\{1, a_2, a_3, \ldots, a_n\}$
  - Some sets of coins allow a greedy solution
    - Always choose biggest coin smaller than amount
    - $\{1,5,10,20,50\}$
    - $\{1,5,10,50,100,200,1000\}$
  - Others are not regular, i.e. the greedy solution is not always best
    - $\{1,6,7\}$
    - Why is this not regular?

$$12 = 7 + 1 + 1 + 1 + 1 + 1 = 6 + 6$$

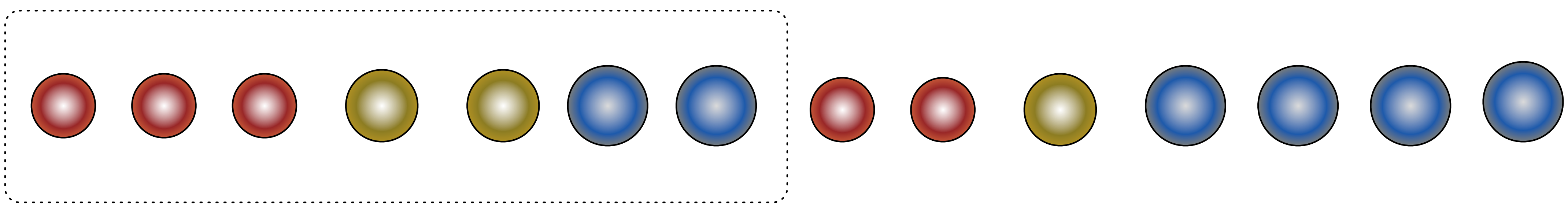
Coin Change Problem

- Dynamic Programming approach
  - Common subproblem structure?

Coin Change Problem

- Let $x_1 + x_2a_2 + x_3a_3 + \ldots + x_na_n = M$ be an optimal way to make change with $x_1 + x_2 + x_3 + \ldots + x_n$ coins.
- Then this is also an optimal way to make change for N

$$y_1 + y_2a_2 + y_3a_3 + \ldots + y_na_n = N, \quad y_1 \le x_1, y_2 \le y_2, \ldots, y_n \le x_n$$



- Proof?

Coin Change Problem

- Because of the common sub-problem property, we can use dynamic programming
- Easiest organized by limiting the number of coins
  - Let the values of the coins be

$$1 = a_1 < a_2 < a_3 < \ldots < a_n$$

  - For all amounts $m \leq M$ let $d_k(m)$ be the minimum number of coins using denominations $a_1, a_2, \ldots, a_k$

Coin Change Problem

- What is the recursive formula?

Coin Change Problem

$$d_k(m) = \min\{d_{k-1}(m - a_k) + 1, d_{k-1}(m - 2a_k) + 2, d_{k-1}(m - 3a_k) + 3, \ldots d_{k-1}(m - \lfloor \frac{m}{a_k} \rfloor a_k) + \lfloor \frac{m}{a_k} \rfloor\}$$

Coin Change Problem

- Solve the coin problem for $\{1,6,7\}$ and amount 25