Greedy Algorithms

A special case of dynamic programming
- Where the number of subproblems to consider is 1
- Where the choice to be made can be determined independent of the subproblems
- Still uses concept of optimal subproblems
- A series of locally optimal choices lead to a globally optimal solution
- Only works for certain problems

Greedy Algorithm proofs

Simplest—Show following two properties
- Greedy choice
  - There exists an optimal algorithm that makes the greedy choice
- Optimal substructure
  - Show any optimal solution exhibits optimal substructure
  - Show greedy algorithm makes first choice consistent with an optimal solution and remainder of problem exhibits optimal substructure

Lower Bound
- Show a lower bound for any solution
- Show greedy algorithm meets that lower bound

Relationship between Dynamic Programming and Greedy

Interval scheduling
- Set S of jobs. Job \( A_j \) starts at \( s_j \) and finishes at \( f_j \)
- Goal: Find maximum subset of non-overlapping jobs.
Interval scheduling

Dynamic programming
- Define $S_{ij}$ as the set of jobs which start at $fi$ and finish at $sj$ (compatible with i and j). $S_{0,n+1}=S$
- $OPT(i, j)$ is the optimal solution for the set $S_{ij}$ (sorted by finish order).
- $OPT(i, j) =$
  - 0 if $i \geq j$
  - $\max_{k \in S_{ij}} (OPT(i, k) + OPT(k, j) + 1)$
- Note two subproblems
Note that the optimal solution for $S_{ij}$ includes the job with earliest finish time, $A_m$
- Look at an optimal scheduling $S^*$ of $S_{ij}$, and look at the job $A_k$ in that optimal scheduling with the earliest finish time. Replace $A_k$ with the job with the strictly earliest finish time from $S_{ij}$. It is compatible with all the other jobs in $S^*$ ($f_m \preceq f_i$)

Simplified Dynamic programming
- $OPT(i, j)$ is the optimal solution for the set $S_{ij}$ (sorted by finish order)
- $OPT(i, j) =$
  - 0 if $i \geq j$
  - $OPT(f_{i+1}, j) + 1$
- Now only one subproblem and choice is forced!
Even simpler
- $OPT(i)$ is the optimal solution for the set $S_{i,n+1}$
- $OPT(i) =$
  - 0 if $i \geq n+1$
  - $OPT(f_{i+1}) + 1$

Interval scheduling

Iterative Algorithm:
- Sort jobs by finish time so that $f_1 \preceq f_2 \preceq \ldots \preceq f_n$
- Solution = {} for $i = 1$ to $n$
  - if job $i$ doesn’t overlap with jobs in Solution
    - Solution = Solution U {job $i$}
return Solution

Runtime
- $O(n \log n)$ for sorting
- $O(n)$ for loop
- $O(n \log n)$ overall

Selecting Breakpoints

Selecting breakpoints.
- Road trip from La Jolla to Berkeley along fixed route.
- Refueling stations, $s_i$, at certain points along the way.
- Fuel capacity = $C$.
- Goal: makes as few refueling stops as possible.

Greedy algorithm. Go as far as you can before refueling.
Selecting Breakpoints

Optimal substructure
- Given an optimal solution with breakpoints $b_1...b_k$, create a subproblem $S' = S - \text{stations } s_1,...,s_{b_1}$ from the original problem.
- $b_2,...,b_k$ must be an optimal solution to $S'$

Greedy-choice property
- There exists an optimal solution where the first breakpoint is at the maximum station $s_C$.

Since the selecting breakpoints problem has these two properties, the greedy algorithm for choosing breakpoints produces an optimal solution.

Interval Partitioning

Interval partitioning.
- Lecture $j$ starts at $s_j$ and finishes at $f_j$.
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses 4 classrooms to schedule 10 lectures.

Interval Scheduling

Greedy algorithm
- Go through lectures (in order of start time)
- If an empty lecture hall is available, schedule the lecture
- If not, create a new lecture hall

In detail
- Sort lectures by start time
  numClassrooms = 0
  for $j = 1$ to $n$
    if lecture $j$ is compatible with some classroom $k$
      schedule lecture $j$ in classroom $k$
    else
      numClassrooms++
      schedule lecture $j$ in classroom numClassrooms

Interval Scheduling proof of correctness

Lower bound on optimal algorithm is depth of input
- Depth = maximum number of simultaneous lectures

Greedy algorithm uses depth classrooms
- Classroom $d$ is opened because there are $d-1$ other lectures going on already when we want to schedule job $j$.
- These $d-1$ other lectures started no later than $s_j$.
- At time $s_j + \epsilon$, there are $d$ lectures going on.
- Any optimal algorithm will need $d$ classrooms.
Matroids

A matroid is an ordered pair \( M = (S, I) \) satisfying

- \( S \) is a finite set
- \( I \) is the independent subsets of \( S \).
  - \( I \) is nonempty
  - Any subset of an element of \( I \) is in \( I \) (hereditary)
  - Note that \( \emptyset \) is in \( I \)
- \( M \) satisfies the exchange property
  - For two elements, \( A, B \), of \( I \) where \( |A| < |B| \), there exists an element \( x \) in \( B-A \) such that \( A \cup \{x\} \) is in \( I \)

Example matroid: graphic matroid

- Given \( G = (V, E) \)
  - \( S = E \)
  - A set of edges is independent iff they don’t form a cycle in \( G \)
    - Clearly, hereditary
  - \( M_g = (S_g, I_g) \) satisfies the exchange property
    - Two forests \( A, B \), \( |A| < |B| \). \( B \) has fewer trees. There exists one tree in \( B \) corresponding to two trees in \( A \)....

Weighted Matroids

Greedy algorithm for weighted matroids

- \( A = \{\} \)
  - Sort \( S \) in decreasing order
  - foreach \( x \) in \( S \)
    - if \( A \cup \{x\} \) is in \( I \) then
      - \( A = A \cup \{x\} \)

Greedy-choice property

- Let \( A = (x) \), the highest weight element such that \( \{x\} \) is independent.
- Use exchange property to add elements from \( B \), an optimal subset.

Optimal substructure property

- Choose \( (x) \) from \( S \). Consider \( M' = (S', I') \), the contraction of \( M \) by \( x \):
  - \( S' \) = set of \( y \) such that \( (x, y) \) are in \( S \)
  - \( I' \) = set of \( B \) in \( S' - x \) such that \( B \cup \{x\} \) is in \( I' \)
- If \( A \) (containing \( x \)) is optimal MIS of \( M \), then \( A - (x) \) is optimal MIS of \( M' \)
- If \( A' \) is optimal MIS of \( M' \), then \( A \cup \{x\} \) is optimal MIS of \( M \)

Task-scheduling: \( n \) unit-time tasks \( a_1, a_n \), \( n \) deadlines \( d_1, d_n \), \( n \) profits, \( w_1, w_n \)

- Determine ordering of tasks that maximizes total profits for met deadlines
- Let \( A = \) maximum independent set of tasks that can all be completed on time.
- Algorithm
  - Sort \( a_1, a_n \) by decreasing profit
  - \( A = \{\} \)
  - for \( i = 1 \) to \( n \)
    - if \( A \cup \{a_i\} \) can be completed by deadlines
      - \( A = A \cup \{a_i\} \)
  - result = elements of \( A \) sorted by deadline time followed by remaining elements of \( a_1, a_n \) in any order