EECS 336: Lecture 3: Introduction to Algorithms

Dynamic Programming Weighted Interval Scheduling

Reading: 6.0-6.3

Last Time:

- philosophy
- computational tractability
- runtime analysis & big-oh

Today:

- Dynamic Programming (a derivation)
- Weighted interval scheduling

Dynamic Programming

"divide problem into small number of subproblems and memoize solution to avoid redundant computation"

Example: Weighted Interval Scheduling

input:

- n jobs $J = \{1, ..., n\}$
- s_i = start time of job i
- f_i = finish time of job i
- v_i = value of job i

compatibility constraint: Only one job can run at once.

output: Schedule $S \subseteq J$ if compatible jobs with maximum total value.

Find a First Decision

"make progress towards a solution"

Idea: job i is either in $\text{OPT}(J)$ or not.

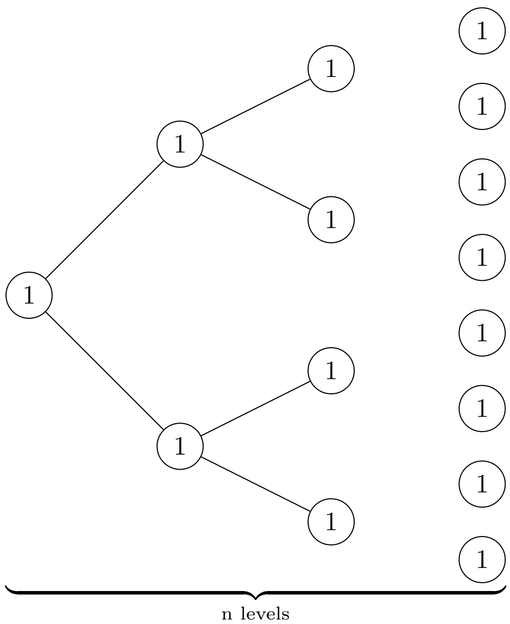
1. let J' = jobs compatible with i in J.
2. let V = value of OPT if "$i \notin \text{OPT}(J)$"

   $= \text{OPT}(J \setminus \{i\})$.
3. let V' = value of OPT if "$i \in \text{OPT}(J)$."

   $= v_i + \text{OPT}(J')$
4. return $\text{OPT}(J) = \max(V, V')$.

Note: subproblems: schedule J' and $J \setminus \{i\}$.

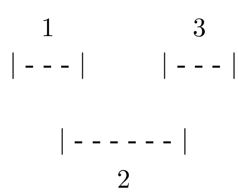
Recurrence: $T(n) = 2T(n-1) + 1$

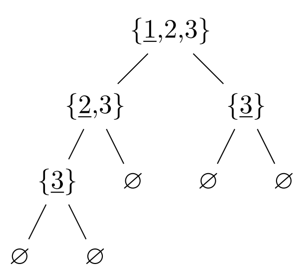


$T(n) = O(2^n)$

Challenge 1: redundant computation

Example:





Note: OPT($\{3\}$) called twice!

Solution: memoize.

"when computing the value of a subproblem save the answer to avoid computing it again"

Result: runtime = # of subproblems $\times$ cost to combine.

Challenge 2: could have too many subproblems. (could be exponential!)

Solution: require "succinct description" of subproblems.

Idea: for interval scheduling, process jobs in order of start time so subproblems suffixes of order.

- sort jobs by increasing start time, $s_1 \leq s_2 \leq ... \leq s_n$.
- let next[i] denote job with earliest start time after i finishes. (if none, set next[i] $= n + 1$.)
- subproblems when processing job 1:
  - $J' = \{\text{next}[i], \text{next}[i] + 1, ..., n\}$
  - $J \setminus \{i\} = \{2, 3, ..., n\}$
- suffix $\{j, ..., n\}$ is succinctly described by "j".

  (only n subproblems)

Recursive Memoized Algorithm

Algorithm: Weighted Interval Scheduling:

1. sort jobs by increasing start time.
2. initialize array next[i].
3. initialize OPT[i] $= \varnothing$ for all i.
4. initialize OPT[$n + 1$] $= 0$.
5. compute OPT(1).

Subroutine: OPT(i)

1. if OPT[i] $\neq \varnothing$, return OPT[i].
2. OPT[i] $\leftarrow \max(v_i + \text{OPT}[\text{next}[i]],\ \text{OPT}[i + 1])$.
3. return OPT[i].

Correctness

"OPT(i)" is correct (by induction on i)

Runtime Analysis

- n subproblems
- constant time to combine
- initialization: sorting & precomputing 'next' array

Runtime: $O(n)+$ initialization $= O(n \log n)$

Iterative DPs

"fill in memoization table from bottom to top"

Algorithm: iterative weighted interval scheduling

1. $\mathrm{OPT}[n+1] = 0$
2. for $i = n$ down to 1:

   $\mathrm{OPT}[i] = \max(v_i + \mathrm{OPT}[\mathrm{next}[i]], \mathrm{OPT}[i+1])$.

Finding Optimal Schedule

"traverse memoization table to find schedule"

Algorithm: schedule

1. $i = 1$
2. while $i < n$:

   if $\mathrm{OPT}[i+1] < v_i + \mathrm{OPT}[\mathrm{next}[i]]$:

   (a) schedule i.
   (b) $i \leftarrow \mathrm{next}(i)$.

   else: $i \leftarrow i + 1$.

Key Ideas of Dynamic Programming

Subproblems must be:

1. succinct (only a polynomial number of them)
2. efficiently combinable.
3. depend on "smaller" subproblems (avoid infinite loops), e.g.,
   - process elements "once and for all"
   - "measure of progress/size."

Seven Part Approach

I. identify subproblem in English

   $\mathrm{OPT}(i)$ = "optimal schedule of $\{i, ..., n\}$ (sorted by starting time)"

II. specify subproblem recurrence (argue correctness)

   $\mathrm{OPT}(i) = \max(\mathrm{OPT}(i+1), v_i + \mathrm{OPT}(\mathrm{next}[i]))$

III. solve the original problem from subproblems

   Optimal Interval Schedule $= \mathrm{OPT}(1)$

IV. identify base case

   $\mathrm{OPT}(n+1) = 0$

V. write iterative DP.

VI. runtime analysis.

   $O(n)$ + initialization $= O(n \log n)$

VII. implement in your favorite language (Python!)