CMPSCI 311: Introduction to Algorithms

Lecture 13: Dynamic Programming 2

Akshay Krishnamurthy

University of Massachusetts

Last Compiled: March 21, 2018

Recap – Dynamic Programming Recipe

- Devise recursive form for solution
- Observe that recursive implementation involves redundant computation. (Often exponential time)
- Design iterative algorithm that solves all subproblems without redundancy.

Recap: Weighted Interval Scheduling

- n shows with start s_i, finish f_i, value v_i
- Find set S of compatible shows with maximum value $\sum_{i \in S} v_i$.

Recurrence $\text{VAL}(m) = \max\{\text{VAL}(p(m)) + v_m, \text{VAL}(m-1)\}$

- With $p(m) = \max\{j : f_j \le s_m\}$
- Rather than solve recursively, solve iteratively from $1, \ldots, n$.

Subset Sum

Problem. Given n jobs where job i requires w_i minutes of time and a budget W.

- Find subset S that maximizes $\sum_{i \in S} w_i$ and has $\sum_{i \in S} w_i \le W$.
- Example: $w_1 = 2, w_2 = 3, w_3 = 5, w_4 = 6, w_5 = 8, W = 12$
- Greedy? Divide and Conquer?

Solution Recurrence

Let O be the optimal solution.

- If $n \notin O$ then O is optimal solution using $\{1, \ldots, n-1\}$.
- If $n \in O$ then O is optimal solution using $\{1, \ldots, n-1\}$ and budget $W - w_n$.

$\text{VAL}(j, W) = \max\{\text{VAL}(j-1, W), w_j + \text{VAL}(j-1, W - w_j)\}$
Unless $W < w_j$, then $\text{VAL}(j, W) = \text{VAL}(j-1, W)$.

Need to track both jobs and remaining budget.

SS Dynamic Program

```
SS-Table(n,W)
    $M[0..n, 0..W] = \texttt{null}$
    $M[0,:] = 0$
    for $j = 1, \ldots, n$ do
        for $w = 0, \ldots, W$ do
            if $w < w_j$ then
                $M[j,w] \leftarrow M[j-1,w]$
            else
                $M[j,w] \leftarrow \max\{M[j-1,w], w_j + M[j-1, w-w_j]\}$
            end if
        end for
    end for
```

Example

$w_1 = 2, w_2 = 2, w_3 = 3, W = 4$

$M[j, w] \leftarrow \max\{M[j-1, w], w_j + M[j-1, w - w_j]\}$

	$w=0$	$w=1$	$w=2$	$w=3$	$w=4$
$j=3$	0	0	2	3	4
$j=2$	0	0	2	2	4
$j=1$	0	0	2	2	2
$j=0$	0	0	0	0	0

Finding Optimal Solution

- Similar to weighted interval scheduling.
- Walk table from $M[n, W]$, following the entry you are based on.

$w_1 = 2, w_2 = 2, \mathbf{w_3 = 3}, W = 4$

	$w=0$	$w=1$	$w=2$	$w=3$	$w=4$
$j=3$	0	0	2	3	**4**
$j=2$	0	0	2	2	**4**
$j=1$	0	0	**2**	2	2
$j=0$	**0**	0	0	0	0

Running Time

- Table has $O(nW)$ entries, each entry requires $O(1)$ computation.
- Finding optimal solution takes $O(n)$ time with table.
- $\Rightarrow O(nW)$ time.
- Not polynomial in size of the input, since W can be specified in $\log_2 W$ bits. *Pseudo-polynomial time*

Next up – Algorithmic problems in Biology

- Protein structure prediction
- Sequence Alignment

Some biology background

- DNA is a string of bases, taking symbols $\{A, C, G, T\}$.
- DNA is often found as paired strings where $A - T, C - G$.
- Example:

```
A  A  T  A  G  C   strand
|  |  |  |  |  |
T  T  A  T  C  G   complement
```

- RNA takes symbols $\{A, C, G, U\}$, but no complement pair.
- Instead RNA pairs with itself, forming a folded molecule.
- Folded structure critical for determining RNA function.

RNA folding

- RNA folds by binding $A - U$ and $C - G$.
- Bases can't bind to more than one other base.
- Want a *stable* configuration: Maximize number of pairings.

Mathematical Model

- RNA is a string $B = b_1 b_2 \dots b_n$ where $b_i \in \{A, C, G, U\}$.
- A *folding* S is a set of pairs $\{(i, j)\}$ where $i, j \in \{1, \dots, n\}$.
- A folding is valid if
  - No sharp turns. $\forall (i, j) \in S$, $|i - j| > 4$.
  - Pairs complement. $\forall (i, j) \in S$, if $b_i = A$ then $b_j = U$, etc.
  - Matching. If $(i, j) \in S$ then $(i, k) \notin S$ for any $k \neq j$.
  - No crossings. If $(i, j), (k, \ell) \in S$, cannot have $i < k < j < \ell$.

Example. AUGAUGGCCAU

RNA Structure Prediction

Problem. Given RNA string B of length n, find valid folding S with maximum number of pairs.

- Consider last base b_n.
  - Either n not paired in OPT.
  - or n paired with some complementary j with $|j - n| > 4$ in OPT.
    - Then what? By no crossing, two subproblems.
- Subproblems are intervals $\{i, \dots, j\}$.

Recursive form

- Let $\mathrm{VAL}(i, j)$ denote maximum number of base pairs in folding on $b_i b_{i+1} \dots b_j$.
- Computing $\mathrm{VAL}(i, j)$.
  - j is not paired $\Rightarrow \mathrm{VAL}(i, j) = \mathrm{VAL}(i, j - 1)$.
  - j is paired with some t, then $\mathrm{VAL}(i, j) = 1 + \mathrm{VAL}(i, t - 1) + \mathrm{VAL}(t + 1, j - 1)$.
- $\mathrm{VAL}(i, j)$ is the maximum of all of these options.
- What is a good order?

Computing VAL

```
Initialize $M[0..n, 0..n]$.
Set $M[i, j] = 0$ for all $i, j$ with $|i - j| \leq 4$.
**for** $k = 5, 6, \dots, n - 1$ **do**
    **for** $i = 1, \dots, n - k$ **do**
        Set $j \leftarrow i + k$.
        Compute $\mathrm{VAL}(i, j)$ using recursive form.
    **end for**
**end for**
```

- Example. AUGAUGCAU
- Running time. $O(n^3)$.
- How to recover the actual folding?

RNA Structure prediction takeaways

- Two new things
  - Dynamic programming over intervals.
  - Each cell depends on $O(n)$ previous cells.

Sequence Alignment

- Biologists use genetic similarity to determine evolutionary relationships.
- But how do we say if two gene sequences are similar or not?
- We *align* them.
- Also used in spell-checkers and search engines.

Sequence Alignment

- For two strings $X = x_1 x_2 \dots x_m, Y = y_1 y_2 \dots y_n$, an alignment M is a matching between $\{1, \dots, m\}$ and $\{1, \dots, n\}$.
- M is valid if
  - Matching. Each element appears in at most one pair in M.
  - No crossings. If $(i,j), (k,\ell) \in S$, the $i < k$ and $j < \ell$.
- Cost of M:
  - Gap penalty. For each unmatched character, you pay δ.
  - Alignment cost. For a match (i,j), you pay $C(x_i, y_j)$.

$$\mathrm{cost}(M) = \delta(n + m - 2|M|) + \sum_{(i,j)\in M} C(x_i, y_j).$$

Sequence Alignment

Problem. Given strings X, Y gap-penalty δ and cost matrix C, find valid alignment of minimal cost.

Example 1 Massachusetts vs Massachussets, $\delta = 0.5$, $C(x,y) = \mathbf{1}[x \neq y]$.

Example 2 Massachusetts vs Massachussets, $\delta = 10$, $C(x,y) = \mathbf{1}[x \neq y]$.

Toward an algorithm

- Try what we did before: Let O be optimal alignment.
  - If $(m,n) \in O$ we can align $x_1 x_2 \dots x_{m-1}$ with $y_1 y_2 \dots y_{n-1}$.
  - If $(m,n) \notin O$ then either m or n must be unmatched (by no crossing).
- Optimal alignment $\mathrm{OPT}(m,n)$ is either,
  - $\mathrm{OPT}(m-1, n-1) \cup \{(m,n)\}$,
  - $\mathrm{OPT}(m-1, n)$, If m unmatched
  - $\mathrm{OPT}(m, n-1)$. If n unmatched

Cost recurrence

Let $\mathrm{cost}(i,j)$ be cost of optimal alignment on $\{1, \dots, i\}, \{1, \dots, j\}$.

$$\mathrm{cost}(i,j) = \min \left\{ \begin{array}{c} C(x_i, y_j) + \mathrm{cost}(i-1, j-1) \\ \delta + \mathrm{cost}(i-1, j) \\ \delta + \mathrm{cost}(i, j-1) \end{array} \right\}$$

And, (i,j) is in optimal alignment if and only if first term is the minimum.