CMPSCI 311: Introduction to Algorithms

Lecture 14: Dynamic Programming 3

Akshay Krishnamurthy

University of Massachusetts

Last Compiled: March 27, 2018

Announcements

- Quiz due tonight
- Homework 4 out
- Hopefully Hw 3 graded by end of week
- Discussion on friday as usual

Recap

Three dynamic programming problems

- Weighted interval scheduling
  - Subproblems are prefixes
- Subset sum
  - Subproblems are prefixes *and* remaining budget
- RNA Folding
  - Subproblems are intervals

Today

- Sequence Alignment
- Shortest paths with negative weights
- All-pairs shortest paths

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

Example. TAIL vs TALE

- For two strings $X = x_1x_2 \ldots x_m, Y = y_1y_2 \ldots y_n$, an alignment M is a matching between $\{1, \ldots, m\}$ and $\{1, \ldots, n\}$.
- M is valid if
  - Matching. Each element appears in at most one pair in M.
  - No crossings. If $(i, j), (k, \ell) \in M$, then $i < k$ and $j < \ell$.
- Cost of M:
  - Gap penalty. For each unmatched character, you pay δ.
  - Alignment cost. For a match (i, j), you pay $C(x_i, y_j)$.

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

Sequence Alignment

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

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

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

Toward an algorithm

- Try what we did before: Let O be optimal alignment.
  - If $(m, n) \in O$ we can align $x_1 x_2 ... x_{m-1}$ with $y_1 y_2 ... y_{n-1}$.
  - If $(m, n) \notin O$ then either x_m or y_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 $\{1, \dots, i\}, \{1, \dots, j\}$.

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

And, (i, j) is in optimal alignment iff first term is the minimum.

Sequence Alignment pseudocode

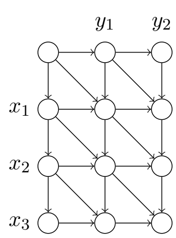
```
align(X,Y)
  Initialize A[0..m,0..n] = null.
  A[i,0] = iδ, A[0,j] = jδ for all i, j.
  for j = 1,...,n do
     for i = 1,...,m do
        v1 = C(x_i, y_j) + A[i−1, j−1].
        v2 = δ + A[i−1, j].
        v3 = δ + A[i, j−1].
        A[i,j] ← min{v1, v2, v3}.
     end for
  end for
```

Example. TALE and TAIL, $\delta = 1, C(x, y) = \mathbf{1}[x \neq y]$.

Example. $\delta = 1$, cost 1 for matching different vowels/consonants, cost 2 for matching vowel with consonant.

Sequence Alignment

- Running time is $O(mn)$.
- Computing optimal matching is easy.
- Related to shortest path in weighted directed graph.



Fact. If $f(i, j)$ is shortest path from $(0, 0)$ to (i, j), then $f(i, j) = \mathrm{cost}(i, j)$.

Sequence Alignment in Linear Space

Question. Can we find optimal alignment in $O(m + n)$ space?

- Current implementation requires $O(mn)$ space.
- Easy to find optimal value in $O(\min\{m, n\})$ space.
  - To compute $\mathrm{cost}(i, \cdot)$ only need to store $\mathrm{cost}(i-1, \cdot)$.
- But how to recover optimal matching afterwards?

Forward and Backward Programs

- $f(i,j)$ is shortest path from $(0,0)$ to (i,j) in alignment graph.
- $g(i,j)$ is shortest path from (i,j) to (m,n),
  - $g(i,j)$ is cost of aligning $x_{i+1} \dots x_m$ with $y_{j+1} \dots y_n$.

$$g(i,j) = \min \left\{ \begin{matrix} C(x_{i+1}, y_{j+1}) + g(i+1, j+1) \\ \delta + g(i+1, j) \\ \delta + g(i, j+1) \end{matrix} \right\}$$

Shortest paths and forward/backward programs.

Fact 1. The length of the shortest path through (i,j) from $(0,0)$ to (m,n) is $f(i,j) + g(i,j)$.

Fact 2. Fix $k \in \{0, \dots, n\}$ and let q minimize $f(q,k) + g(q,k)$. Then the shortest path from $(0,0)$ to (m,n) passes through (q,k).

Divide and Conquer + Dynamic Programming.

```
Seq-Align(X,Y)
  Let m = length(X), n = length(Y).
  If m ≤ 2 or n ≤ 2, compute optimal alignment.
  Compute f(:, n/2) and g(:, n/2) in linear space.
  Let q minimize f(q, n/2) + g(q, n/2). Save (q, n/2).
  Seq-Align(X[0:q],Y[0:n/2])
  Seq-Align(X[q+1:m],Y[n/2+1:n])
```

Running time $O(mn)$ space $O(m+n)$

Running time analysis.

Recurrence.

$$T(m,n) \le cmn + T(q, n/2) + T(m-q, n/2)$$

- If $n = m$ and q always $n/2$, then solves to $O(n^2)$.
- Guess $T(m,n) \le kmn$ and prove by induction.

Sequence Alignment Takeaways

- Standard application of dynamic programming
- Sometimes we can be smart about complexity (e.g., linear space).
- Connection to shortest paths?
- Widely used in the real world!
- Faster alignment seems impossible.

Shortest Paths Revisited

Shortest $s \rightsquigarrow t$ path in directed graph with positive and negative weights?

Problem. Given weighted directed graph $G = (V, E, c)$ where $c_e \in \mathbb{Z}$ with no negative cycles, compute shortest path between s and t.

- Djikstra's? Any other tricks?
- Dynamic programming? What are the subproblems?

Bellman-Ford Algorithm

Fact. If no negative cycles, shortest path has at most $n-1$ edges.

- Let $\text{cost}(i, v)$ be cost of optimal $v \rightsquigarrow t$ path with at most i edges.
- Let P be the optimal $v \rightsquigarrow t$ path using at most $i+1$ edges.
  - If P uses at most i edges, then $\text{cost}(i+1, v) = \text{cost}(i, v)$.
  - Else $P = v \to w \rightsquigarrow t$ where $w \rightsquigarrow t$ path uses at most i edges.

$$\text{cost}(i+1, v) = c_{v,w} + \text{cost}(i, w)$$

Bellman-Ford

$$\text{cost}(i, v) = \min \left\{ \text{cost}(i-1, v), \min_{w \in V} \{c_{v,w} + \text{cost}(i-1, w)\} \right\}$$

Leads to $O(n^3)$ algorithm for shortest paths.

Extensions

- Refined analysis gives $O(mn)$ runtime.
- Can implement in $O(n)$ space.
- Decentralized implementation.

All-pairs shortest paths

- How fast can we compute all shortest paths in a graph?
  - Djikstra's gives $O(nm \log_2 n)$.
  - Bellman-Ford gives $O(n^2 m)$.
  - (new) Floyd-Warshall gives $O(n^3)$.

Problem. Given $G = (V, E, c)$ with non-negative weights, compute $n \times n$ array M where $M[s, t]$ is the cost of shortest $s \rightsquigarrow t$ path.

- What are good subproblems?

Floyd-Warshall algorithm

- Let $\text{cost}(s, t, k)$ be cost of shortest $s \rightsquigarrow t$ path using only vertices $\{1, \ldots, k\}$ as intermediate points.

$$\text{cost}(s, t, k+1) = \min \begin{cases} \text{cost}(s, t, k) \\ \text{cost}(s, k+1, k) + \text{cost}(k+1, t, k) \end{cases}$$

- Running time. $O(n^3)$.
- Recovering paths requires careful book-keeping.