CSCI 355: Algorithm Design and Analysis
8. Dynamic Programming II

- *sequence alignment*
- *Bellman–Ford–Moore algorithm*
- *distance-vector protocols*

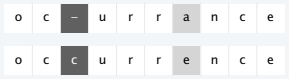
String similarity

Q. How similar are two strings?

Ex. `ocurrance` and `occurrence`.

6 mismatches, 1 gap

0 mismatches, 3 gaps



1 mismatch, 1 gap

Edit distance

Edit distance. [Levenshtein 1966, Needleman–Wunsch 1970]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost = sum of gap and mismatch penalties.

cost = $\delta + \alpha_{CG} + \alpha_{TA}$

assuming $\alpha_{AA} = \alpha_{CC} = \alpha_{GG} = \alpha_{TT} = 0$

Applications. Bioinformatics, spell correction, machine translation, speech recognition, information extraction, ...

```
Spokesperson confirms     senior government adviser was found
Spokesperson said     the senior            adviser was found
```

Editing "*The Gray Lady*"

~~Why~~ Strike at Largest U.S. Wholesale Produce Market ~~Workers Are on Strike: They Want $1-an-Hour Raise~~ Threatens Supply Chain

A Beleaguered Biden ~~Cancels $1.2 Billion in~~ Chips Away at Student Loan ~~Debt for 150,000 Borrowers~~ Debt, Bit by Bit

~~The police entered the compound and~~ Gaza militants fired ~~rubber-tipped bullets. Anger was already building in response to~~ rockets toward Jerusalem and the ~~looming expulsion of several~~ Israeli police fought with Palestinian ~~families from their homes~~ protesters in ~~the city.~~ an escalation of violence after a week of increasing tensions.

BLOSUM matrix for proteins

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	7	-3	-3	-3	-1	-2	-2	0	-3	-3	-3	-1	-2	-4	-1	2	0	-5	-4	-1
R	-3	9	-1	-3	-6	1	-1	-4	0	-5	-4	3	-3	-5	-3	-2	-2	-5	-4	-4
N	-3	-1	9	2	-5	0	-1	-1	1	-6	-6	0	-4	-6	-4	1	0	-7	-4	-5
D	-3	-3	2	10	-7	-1	2	-3	-2	-7	-7	-2	-6	-6	-3	-1	-2	-8	-6	-6
C	-1	-6	-5	-7	13	-5	-7	-6	-7	-2	-3	-6	-3	-4	-6	-2	-2	-5	-5	-2
Q	-2	1	0	-1	-5	9	3	-4	1	-5	-4	2	-1	-5	-3	-1	-1	-4	-3	-4
E	-2	-1	-1	2	-7	3	8	-4	0	-6	-6	1	-4	-6	-2	-1	-2	-6	-5	-4
G	0	-4	-1	-3	-6	-4	-4	9	-4	-7	-7	-3	-5	-6	-5	-1	-3	-6	-6	-6
H	-3	0	1	-2	-7	1	0	-4	12	-6	-5	-1	-4	-2	-4	-2	-3	-4	3	-5
I	-3	-5	-6	-7	-2	-5	-6	-7	-6	7	2	-5	2	-1	-5	-4	-2	-5	-3	4
L	-3	-4	-6	-7	-3	-4	-6	-7	-5	2	6	-4	3	0	-5	-4	-3	-4	-2	1
K	-1	3	0	-2	-6	2	1	-3	-1	-5	-4	8	-3	-5	-2	-1	-1	-6	-4	-4
M	-2	-3	-4	-6	-3	-1	-4	-5	-4	2	3	-3	9	0	-4	-3	-1	-3	-3	1
F	-4	-5	-6	-6	-4	-5	-6	-6	-2	-1	0	-5	0	10	-6	-4	-4	0	4	-2
P	-1	-3	-4	-3	-6	-3	-2	-5	-4	-5	-5	-2	-4	-6	12	-2	-3	-7	-6	-4
S	2	-2	1	-1	-2	-1	-1	-1	-2	-4	-4	-1	-3	-4	-2	7	2	-6	-3	-3
T	0	-2	0	-2	-2	-1	-2	-3	-3	-2	-3	-1	-1	-4	-3	2	8	-5	-3	0
W	-5	-5	-7	-8	-5	-4	-6	-6	-4	-5	-4	-6	-3	0	-7	-6	-5	16	3	-5
Y	-4	-4	-4	-6	-5	-3	-5	-6	3	-3	-2	-4	-3	4	-6	-3	-3	3	11	-3
V	-1	-4	-5	-6	-2	-4	-4	-6	-5	4	1	-4	1	-2	-4	-3	0	-5	-3	7

Sequence alignment

Goal. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a min-cost alignment.

Def. An alignment M is a set of ordered pairs $x_i - y_j$ such that each character appears in at most one pair and no crossings.

$x_i - y_j$ and $x_{i'} - y_{j'}$ cross if $i < i'$, but $j > j'$

Def. The cost of an alignment M is defined as follows:

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	–	G
–	T	A	C	A	T	G
y_1	y_2	y_3	y_4	y_5		y_6

an alignment of CTACCG and TACATG

$M = \{ x_2\text{–}y_1, x_3\text{–}y_2, x_4\text{–}y_3, x_5\text{–}y_4, x_6\text{–}y_6 \}$

Sequence alignment: problem structure

Def. $OPT(i, j)$ = min cost of aligning prefix strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.
Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i - y_j$.
Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$.
(could be negative if characters match)

Case 2a. $OPT(i, j)$ leaves x_i unmatched.
Pay gap for x_i + min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_j$.

Case 2b. $OPT(i, j)$ leaves y_j unmatched.
Pay gap for y_j + min cost of aligning $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_{j-1}$.

optimal substructure property (proof via exchange argument)

Bellman equation.

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \end{cases}$$

Sequence alignment: bottom-up DP algorithm

SEQUENCE-ALIGNMENT$(m, n, x_1, \ldots, x_m, y_1, \ldots, y_n, \delta, \alpha)$

FOR $i = 0$ TO m
  $M[i, 0] \leftarrow i\delta$.
FOR $j = 0$ TO n
  $M[0, j] \leftarrow j\delta$.

FOR i = 1 TO m
  FOR j = 1 TO n
    $M[i, j] \leftarrow \min \{ \alpha_{x_i y_j} + M[i-1, j-1],$
      $\delta + M[i-1, j],$
      $\delta + M[i, j-1] \}.$

already computed

RETURN $M[m, n]$.

Sequence alignment: traceback

		S	I	M	I	L	A	R	I	T	Y
	0	2	4	6	8	10	12	14	16	18	20
I	2	4	1	3	2	4	6	8	7	9	11
D	4	6	3	3	4	4	6	8	9	9	11
E	6	8	5	5	6	6	6	8	10	11	11
N	8	10	7	7	8	8	8	8	10	12	13
T	10	12	9	9	9	10	10	10	10	9	11
I	1							2	9	11	11
T	1							4	11	8	11
Y	1							4	13	10	7

S	I	M	I	–	L	A	R	I	T	Y
–	I	–	–	D	E	N	T	I	T	Y

Sequence alignment: analysis

Theorem. The DP algorithm computes the edit distance (and an optimal alignment) of two strings of lengths m and n in $\Theta(mn)$ time and space.

Pf.

- The algorithm computes edit distance.
- We can perform a traceback to extract the optimal alignment itself. ▪

Theorem. [Backurs–Indyk 2015] If we can compute the edit distance of two strings of length n in $O(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$, then we can solve SAT with n variables and m clauses in $poly(m)\, 2^{(1-\delta)n}$ time for some constant $\delta > 0$.

which would disprove SETH (strong exponential time hypothesis)

Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)*

Arturs Backurs[†] MIT Piotr Indyk[†] MIT

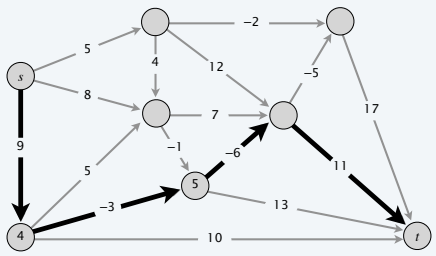
CSCI 355: Algorithm Design and Analysis
8. Dynamic Programming II

- *sequence alignment*
- *Bellman–Ford–Moore algorithm*
- *distance-vector protocols*

Shortest paths with negative weights

Shortest-path problem. Given a digraph $G = (V, E)$, with arbitrary edge lengths ℓ_{vw}, find shortest path from source node s to destination node t.

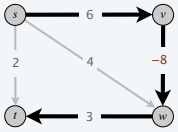
assume there exists a path from every node to t



length of shortest s↝t path = 9 − 3 − 6 + 11 = 11

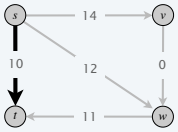
Shortest paths with negative weights: failed attempts

Dijkstra. May not produce shortest paths when edge lengths are negative.



Dijkstra selects the vertices in the order s, t, w, v
But shortest path from s to t is $s \to v \to w \to t$.

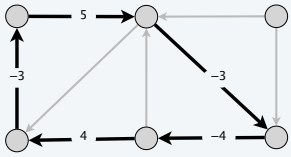
Reweighting. Adding a constant to every edge length does not necessarily make Dijkstra's algorithm produce shortest paths.



Adding 8 to each edge weight changes the shortest path from $s \to v \to w \to t$ to $s \to t$.

Negative cycles

Def. A negative cycle is a directed cycle for which the sum of its edge lengths is negative.

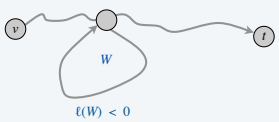


a negative cycle W: $\ell(W) = \sum_{e \in W} \ell_e < 0$

Shortest paths and negative cycles

Lemma 1. If some $v \rightsquigarrow t$ path contains a negative cycle, then there does not exist a shortest $v \rightsquigarrow t$ path.

Pf. If there exists such a cycle W, then we can build a $v \rightsquigarrow t$ path of arbitrarily negative length by detouring around W as many times as we desire. ▪

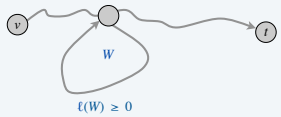


Shortest paths and negative cycles

Lemma 2. If G has no negative cycles, then there exists a shortest $v \rightsquigarrow t$ path that is simple (and has $\leq n-1$ edges).

Pf.

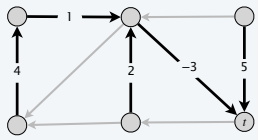
- Among all shortest $v \rightsquigarrow t$ paths, consider one that uses the fewest edges.
- If that path P contains a directed cycle W, then we can remove the portion of P corresponding to W without increasing its length. ▪



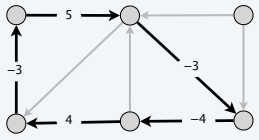
Shortest-paths and negative-cycle problems

Single-destination shortest-paths problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw} (but no negative cycles) and a distinguished node t, find a shortest $v \rightsquigarrow t$ path for every node v.

Negative-cycle problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw}, find a negative cycle (if one exists).



shortest-paths tree



negative cycle

Shortest paths with negative weights: dynamic programming

Def. $OPT(i, v)$ = length of shortest $v \rightsquigarrow t$ path that uses $\leq i$ edges.

Goal. $OPT(n-1, v)$ for each v. ← by Lemma 2, if no negative cycles, there exists a shortest $v \rightsquigarrow t$ path that is simple

Case 1. Shortest $v \rightsquigarrow t$ path uses $\leq i-1$ edges.

- $OPT(i, v) = \mathrm{OPT}(i-1, v)$. ← optimal substructure property (proof via exchange argument)

Case 2. Shortest $v \rightsquigarrow t$ path uses exactly i edges.

- if (v, w) is first edge in shortest such $v \rightsquigarrow t$ path, incur a cost of ℓ_{vw}.
- Then, select best $w \rightsquigarrow t$ path using $\leq i-1$ edges.

Bellman equation.

$$OPT(i, v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = t \\ \infty & \text{if } i = 0 \text{ and } v \neq t \\ \min\left\{ OPT(i-1, v),\ \min\limits_{(v,w) \in E} \{ OPT(i-1, w) + \ell_{vw} \} \right\} & \text{if } i > 0 \end{cases}$$

Shortest paths with negative weights: implementation

SHORTEST-PATHS(V, E, ℓ, t)

FOREACH node $v \in V$:
  $M[0, v] \leftarrow \infty$.
$M[0, t] \leftarrow 0$.
FOR i = 1 TO $n - 1$
  FOREACH node $v \in V$:
    $M[i, v] \leftarrow M[i-1, v]$.
    FOREACH edge $(v, w) \in E$:
      $M[i, v] \leftarrow \min \{ M[i, v],\ M[i-1, w] + \ell_{vw} \}$.

Shortest paths with negative weights: implementation

Theorem 1. Given a digraph $G = (V, E)$ with no negative cycles, the DP algorithm computes the length of a shortest $v \rightsquigarrow t$ path for every node v in $\Theta(mn)$ time and $\Theta(n^2)$ space.

Pf.

- Table requires $\Theta(n^2)$ space.
- Each iteration i takes $\Theta(m)$ time since we examine each edge once. ▪

Finding the shortest paths.

- Approach 1: Maintain $successor[i, v]$ that points to next node on a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.
- Approach 2: Compute optimal lengths $M[i, v]$ and consider only edges with $M[i, v] = M[i-1, w] + \ell_{vw}$. Any directed path in this subgraph is a shortest path.

Shortest paths with negative weights: efficient implementation

Space optimization. Maintain two 1D arrays (instead of 2D array).

- $d[v]$ = length of a shortest $v \rightsquigarrow t$ path that we have found so far.
- $successor[v]$ = next node on a $v \rightsquigarrow t$ path.

Performance optimization. If $d[w]$ was not updated in iteration $i - 1$, then no reason to consider edges entering w in iteration i.

Bellman–Ford–Moore: efficient implementation

BELLMAN–FORD–MOORE(V, E, c, t)

FOREACH node $v \in V$:
  $d[v] \leftarrow \infty$.
  $successor[v] \leftarrow null$.
$d[t] \leftarrow 0$.
FOR $i = 1$ TO $n - 1$
  FOREACH node $w \in V$:
    IF ($d[w]$ was updated in previous pass)
      FOREACH edge $(v, w) \in E$:
        IF ($d[v] > d[w] + \ell_{vw}$)
          $d[v] \leftarrow d[w] + \ell_{vw}$.
          $successor[v] \leftarrow w$.
  IF (no $d[\cdot]$ value changed in pass i) STOP.

pass i
$O(m)$ time

Bellman–Ford–Moore: analysis

Lemma 3. For each node v : $d[v]$ is the length of some $v \rightsquigarrow t$ path.
Lemma 4. For each node v : $d[v]$ is monotone non-increasing.

Lemma 5. After pass i, $d[v] \le$ length of a shortest $v \rightsquigarrow t$ path using $\le i$ edges.

Pf. [by induction on i]

- Base case: $i = 0$.
- Assume true after pass i.
- Let P be any $v \rightsquigarrow t$ path with $\le i + 1$ edges.
- Let (v, w) be the first edge in P and let P' be a subpath from w to t.
- By inductive hypothesis, at the end of pass i, $d[w] \le \ell(P')$ because P' is a $w \rightsquigarrow t$ path with $\le i$ edges.
- After considering edge (v, w) in pass $i + 1$:

and by Lemma 4, $d[w]$ does not increase

$$\begin{aligned} d[v] &\le \ell_{vw} + d[w] \\ &\le \ell_{vw} + \ell(P') \\ &= \ell(P) \quad \blacksquare \end{aligned}$$

and by Lemma 4, $d[v]$ does not increase

Bellman–Ford–Moore: analysis

Theorem 2. Assuming no negative cycles, Bellman–Ford–Moore computes the lengths of the shortest $v \rightsquigarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf. Lemma 2 + Lemma 5. ▪

shortest path exists and has at most $n-1$ edges

after i passes, $d[v] \le$ length of shortest path that uses $\le i$ edges

Remark. Bellman–Ford–Moore is typically faster in practice.

- Edge (v, w) considered in pass $i + 1$ only if $d[w]$ updated in pass i.
- If shortest path has k edges, then algorithm finds it after $\le k$ passes.

Single-source shortest paths with negative weights

year	worst case	discovered by
1955	$O(n^4)$	Shimbel
1956	$O(m\, n^2\, W)$	Ford
1958	$O(m\, n)$	Bellman, Moore
1983	$O(n^{3/4}\, m \log W)$	Gabow
1989	$O(m\, n^{1/2} \log(nW))$	Gabow–Tarjan
1993	$O(m\, n^{1/2} \log W)$	Goldberg
2005	$O(n^{2.38}\, W)$	Sankowsi, Yuster–Zwick
2016	$\tilde{O}(n^{10/7} \log W)$	Cohen–Mądry–Sankowski–Vladu

single–source shortest paths with weights between –W and W

CSCI 355: Algorithm Design and Analysis
8. Dynamic Programming II

- *sequence alignment*
- *Bellman–Ford–Moore algorithm*
- *distance-vector protocols*

Distance-vector routing protocols

Communication network.

- Node ≈ router.
- Edge ≈ direct communication link.
- Length of edge ≈ latency of link. ← non-negative, but Bellman–Ford–Moore used anyway!

Dijkstra's algorithm. Requires global information of network.

Bellman–Ford–Moore. Uses only local knowledge of neighboring nodes.

Synchronization. We don't expect routers to run in lockstep. The order in which each edges are processed in Bellman–Ford–Moore is not important. Moreover, algorithm converges even if updates are asynchronous.

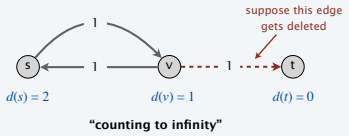
Distance-vector routing protocols

Distance-vector routing protocols. ["routing by rumor"]

- Each router maintains a vector of shortest-path lengths to every other node (distances) and the first hop on each path (directions).
- Algorithm: each router performs n separate computations, one for each potential destination node.

Ex. RIP, Xerox XNS RIP, Novell's IPX RIP, Cisco's IGRP, DEC's DNA Phase IV, AppleTalk's RTMP.

Caveat. Edge lengths may change during algorithm (or fail completely).



"counting to infinity"

Path-vector routing protocols

Link-state routing protocols. not just the distance and first hop

- Each router stores the whole path (or network topology).
- Based on Dijkstra's algorithm.
- Avoids "counting-to-infinity" problem and related difficulties.
- Requires significantly more storage.

Ex. Border Gateway Protocol (BGP), Open Shortest Path First (OSPF).